Emergence, Complexity and Computation ECC

Andrew Adamatzky *Editor*

Automata, Universality, Computation

Tribute to Maurice Margenstern

Springer

Emergence, Complexity and Computation

Volume 12

About this Series

The Emergence, Complexity and Computation (ECC) series publishes new developments, advancements and selected topics in the fields of complexity, computation and emergence. The series focuses on all aspects of reality-based computation approaches from an interdisciplinary point of view especially from applied sciences, biology, physics, or Chemistry. It presents new ideas and interdisciplinary insight on the mutual intersection of subareas of computation, complexity and emergence and its impact and limits to any computing based on physical limits (thermodynamic and quantum limits, Bremermann's limit, Seth Lloyd limits...) as well as algorithmic limits (Gödel's proof and its impact on calculation, algorithmic complexity, the Chaitin's Omega number and Kolmogorov complexity, non-traditional calculations like Turing machine process and its consequences,...) and limitations arising in artificial intelligence field. The topics are (but not limited to) membrane computing, DNA computing, immune computing, quantum computing, swarm computing, analogic computing, chaos computing and computing on the edge of chaos, computational aspects of dynamics of complex systems (systems with self-organization, multiagent systems, cellular automata, artificial life,...), emergence of complex systems and its computational aspects, and agent based computation. The main aim of this series it to discuss the above mentioned topics from an interdisciplinary point of view and present new ideas coming from mutual intersection of classical as well as modern methods of computation. Within the scope of the series are monographs, lecture notes, selected contributions from specialized conferences and workshops, special contribution from international experts.

More information about this series at http://www.springer.com/series/10624

Andrew Adamatzky
Editor

Automata, Universality, Computation

Tribute to Maurice Margenstern

Springer

Editor
Andrew Adamatzky
Unconventional Computing Centre
University of the West of England
Bristol
United Kingdom

ISSN 2194-7287 ISSN 2194-7295 (electronic)
Emergence, Complexity and Computation
ISBN 978-3-319-09038-2 ISBN 978-3-319-09039-9 (eBook)
DOI 10.1007/978-3-319-09039-9

Library of Congress Control Number: 2014945747

Springer Cham Heidelberg New York Dordrecht London

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

[Springer International Publishing AG Switzerland] is part of Springer Science+Business Media
(www.springer.com)

Preface

A Tribute to Maurice Margenstern

A few figures witness Maurice Margenstern's scientific accomplishments. He authored or co-authored 210 papers (some more being on the way...). His 54 co-authors are from all over the world: France, Belgium, Switzerland, Germany, Austria, Italy, Spain, Ireland, United Kingdom, Finland, Hungary, Moldavia, Romania, Russia, Israel, China, Malaysia, Japan, Chili, USA. Besides his research papers, he wrote 6 books, organized and edited the proceedings of 8 international conferences, supervised 5 students.

And, most importantly, he pioneered new areas of research at the intersection of mathematics and computer science.

How did Maurice become the scientist we so much appreciate?

Maurice was born in Paris on June 6, 1947. After high school he was accepted in prestigious French "grandes écoles". Resigning from the École Polytechnique he entered the École Normale Supérieure de Cachan. After he passed the French "agrégation de mathématiques", he got an academic position at the mathematics department of the University of Paris-Sud (Orsay) in 1970.

Attracted by constructive mathematics, then a very marginal subject in France, he went in the early 70's to Leningrad to study with Nicolaï A. Shanin, then the head of the famous Russian school in constructive mathematics founded by A.A. Markov. This is where he worked on his thesis on *Constructive topological properties of spaces of almost periodic functions.*

Then his scientific inclinations gradually moved from constructive mathematics to computability, a subject then much revived after Matijasevich's famous result on Hilbert's 10th problem. In the late 80's he joined the computer science laboratory LITP (now LIAFA) in Paris and started his amazing investigation of the simplest (resp. most complex) machines with an undecidable (resp. decidable) halting problem. Despite the previous works by Yuri Rogogine in Moldavia and Ludmila Pavlotskaia in Russia, this topic was then rather confidential. Maurice is to be credited for putting it up as one of the important themes in computability. He did so via his own contributions and via a triennial international conference *Machines, Computations and Universality* he set up in 1995 and which still goes on. I remember how Josef Gruska (speaking as the last chairman of the first conference, held in Paris) stressed how much energy Maurice had to show in order to create from scratch such a successful international event.

Maurice passed his Habilitation in 1994 "*Study of the frontier between decidability and undecidability of the halting problem of small or constrained Turing machines.*" He was then appointed as a professor in computer science by the university of Lorraine at Metz. There he created the LITA laboratory and spent much energy to make it a solid research center. Some years ago, he was promoted "professeur de classe exceptionnelle", the highest rank in French Academia.

In the early 2000's Maurice developed the subject of computation with cellular automata in the hyperbolic world (dimension 2, 3 or 4) and brought solutions to long open difficult tiling problems. Besides these technical achievements, Maurice proved that hyperbolic computability allows to overcome known unfeasibility results of the Euclidean world.

The classical tool in hyperbolic geometry is the theory of groups but Maurice invented completely new tools: combinatorial ones. As is usual with new ideas, experts in hyperbolic geometry showed reluctance towards such an innovative approach and it was a real relief for Maurice when Donald Knuth wrote to him to express that he found these new tools "quite fascinating" and will refer to them in the Art of Computer Programming.

Of course, the two themes presented above do not exhaust the subjects Maurice investigated but they witness his creative talent and impressive energy.

Maurice recently retired and is now emeritus professor at the university of Lorraine. Having no more to teach and manage a research team, Maurice simply spends more time to continue producing prominent works.

Having had the privilege to know him since the mid 70's, it is my pleasure to say, in the name of all contributors of this volume: Maurice, thank you for what you have accomplished and let's continue our friendship and scientific collaboration.

Serge Grigorieff
Paris

Contents

Chapter 1
The Common Structure of the Curves Having a Same Gauss Word

Bruno Courcelle

Abstract. *Gauss words* are finite sequences of letters associated with self-intersecting closed curves in the plane. (These curves have no "triple" self-intersection). These sequences encode the order of intersections on the curves. We characterize, up to homeomorphism, all curves having a given Gauss word. We extend this characterization to the n-tuples of closed curves having a given n-tuple of words, that we call a *Gauss multiword*. These words encode the self-intersections of the curves and their pairwise intersections. Our characterization uses decompositions of strongly connected graphs in 3-edge-connected components and algebraic terms formalizing these decompositions.

1.1 Introduction

Many geometric configurations can be represented by finite combinatorial objects, up to appropriate equivalence relations, like homeomorphism in the case of embeddings of graphs in surfaces. *Gauss words* are sequences of letters intended to describe the *self-intersections* of closed curves in the plane (with no triple intersection): each crossing is named by a letter, and a word with two occurrences of each letter is obtained by following the curve and writing the letter seen at each crossing. This definition raises the following questions:

(1) What are these words ?

(2) Which curves can be uniquely reconstructed, up to homeomorphism, from the corresponding word ?

(3) What is the common structure of all curves having a same associated word ?

Gauss words are characterized in several articles by Lovasz and Marx [12], Rosenstiehl [16] and de Fraysseix and Ossona de Mendez [8] to name a few. These

Bruno Courcelle
LaBRI, Bordeaux University and CNRS, France
e-mail: courcell@labri.fr

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_1

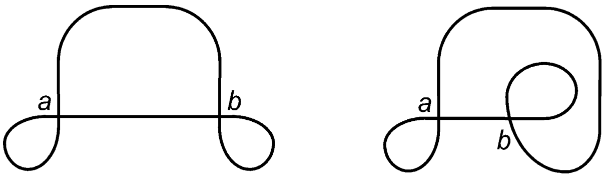


Fig. 1.1 Curves with Gauss word *aabb*

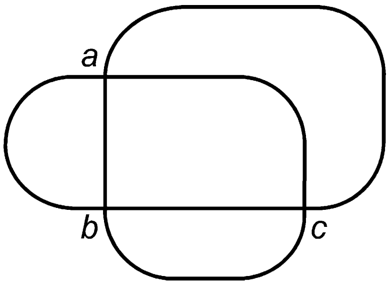


Fig. 1.2 Curve with Gauss word *abcabc*

works answer Question (1). Some of them are reviewed in the book by Godsil and Royle [10].

We will address Questions (2) and (3). A word is *unambiguous* if it characterizes a unique curve up to homeomorphism. Otherwise, it is *ambiguous*. Figure 1.1 shows two curves that are not related by any homeomorphism of the plane or of the sphere but yield the same ambiguous Gauss word *aabb*. On the other hand, the word *abcabc* is unambiguous and Figure 1.2 shows a (the) corresponding curve.

It is natural and convenient to extend the definitions and the corresponding questions to tuples of curves, described by tuples of words such that any letter has exactly two occurrences in one word or one occurrence in two of them. Figure 1.3 shows three curves with corresponding unambiguous *Gauss multiword* $(abcd, aecf, bfde)$.

We will give in Theorem 24 a natural characterization of Gauss multiwords different from those of [8, 12, 16]. By means of Proposition 14, it yields a linear-time recognition algorithm. This algorithm provides also a tuple of corresponding curves if there exists one.

For answering Questions (2) and (3), we will use several notions. First we observe that intersecting and self-intersecting closed curves without triple intersections are nothing but plane 4-regular graphs. We recall that a *map* is a graph equipped, at each vertex, with a circular order of edges around it. This order is called a *rotation* (see the book by Mohar and Thomassen [14], Chapter 3). Every embedding of the graph in an oriented surface yields a map. Intersecting closed curves can be described up to homeomorphism by 4-regular planar maps (we omit here some technical details).

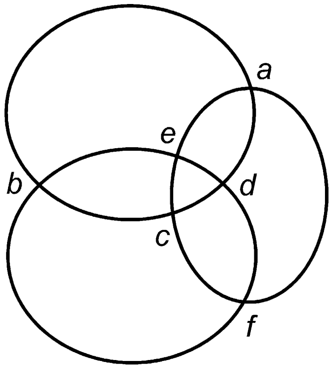


Fig. 1.3 Three intersecting circles

Second, we use a single combinatorial object, constructed from an n-tuple W of words where each letter has two occurrences, from which the planar maps representing the n-tuples of curves with associated multiword W can be defined. This object is a 4-regular graph *with transitions*, that is, equipped, at each vertex v, with a pairing of the half-edges, called *darts*, incident with v. We call it a *t-graph.* In its embeddings in surfaces, two paired darts must form a line that crosses the one formed by the two other incident darts. See Figure 1.5 for an example. (Pairings are represented by thickenings of certain darts. For example, edges a and b are paired at their common vertex, and so are b and c). A t-graph is essentially a 4-regular map in which we forget the orientation of the surface around each vertex (we only retain how darts alternate around a vertex). Hence, a t-graph contains more information than the underlying graph and less than a map of this graph. A t-graph with an underlying planar graph may have no plane embedding that satisfies the "crossing condition" on pairs of opposite darts (see the right part of Figure 1.10). We will prove in Theorem 15 that, if the underlying graph of a planar t-graph is loop-free and 3-edge-connected, then this t-graph has a unique plane embedding, up to homeomorphism.

Third, we start our investigation with tuples of oriented curves (see Figure 1.4). The corresponding t-graphs are directed and each vertex has 2 incoming edges and 2 outgoing edges. We say that they are (2,2)-*regular.* Figure 1.4 shows the difference between oriented and nonorienting curves regarding the ambiguity of Gauss multiwords. It shows two plane embeddings of a (2,2)-regular t-graph associated with the Gauss multiword $(abcd, bc, ad)$ that are not homeomorphic by any homeomorphism of the sphere. (To check this, just compare the directions of the edges incident with the two faces bordered by 4 edges). However, forgetting the directions of the edges yields two homeomorphic embeddings. Hence the multiword $(abcd, bc, ad)$ is unambiguous for representing intersections of nonoriented closed curves, but it is ambiguous for representing intersections of oriented curves. It is actually easier to

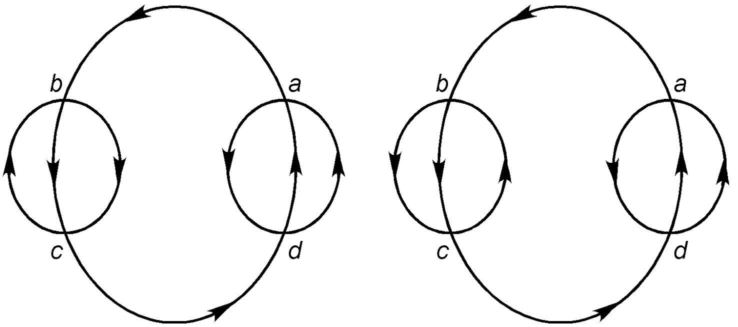


Fig. 1.4 Two nonhomeomorphic embeddings of a same t-map

characterize the unambiguous Gauss multiwords representing oriented curves and we will start by this case.

Our key tool is a decomposition of strongly connected (directed) graphs in 3-edge-connected components that we have studied in [5] under the name of *atomic decomposition*. Figure 1.9 in Section 2.2 below shows an atomic decomposition: the graphs $G_1,...,G_9$ are undecomposable, they are the *atoms* of the decomposed graph.This decomposition works for t-graphs and for strongly connected maps. From the atomic decomposition of the t-graph associated with a Gauss multiword W, and its expression by an algebraic term over a binary operation that composes disjoint graphs, we can describe all tuples of oriented curves (up to homeomorphism) whose associated Gauss multiword is W. Then, we extend this characterization to solve the original questions concerning nonoriented curves. The basic definitions and facts are given and proved in Sections 2.1 and 3. This article can be read independently of [5].

The article is organized as follows. Section 1 reviews definitions about graphs and maps. Section 2 reviews the decomposition of strongly connected graphs in 3-edge-connected components. Section 3 examines the corresponding decompositions of planar graphs and maps. Section 4 develops the applications to the curves in the plane described by given multiwords, and Theorems 24, 26 and 36 answer Questions (1-3) for Gauss multiwords that describe oriented or nonoriented curves. Section 5 reviews open questions. For the reader's convenience, an appendix reviews the various equivalence and isomorphism notions used in this article.

1.2 Definitions

All graphs and related objects (t-graphs, maps) will be finite. By saying that $(e_1,...,e_k)$ is a *circular sequence*, we mean that it can also be specified as $(e_{i+1},...,e_k,$

$e_1,...,e_{i-1})$ and that its properties and associated constructions do not depend on the initial element e_1. (See also Section 4.1.1).

1.2.1 Graphs

1.2.1.1 Terminology and Notation

A directed graph G is a triple $(V_G, E_G, vert_G)$ consisting the set of vertices V_G, the set of edges E_G (with $V_G \cap E_G = \emptyset$) and a mapping $vert_G : E_G \to V_G \times V_G$ that defines incidences. If $vert_G(e) = (x,y)$, we say that x is the *tail* of e, denoted by $\alpha(e)$, that y is its *head*, denoted by $\beta(e)$, we also write $e : x \to_G y$ and we say that x and y are the *ends* of e. If G is undirected, then $vert_G(e)$ is a set $\{x,y\}$ of one or two vertices, called the *ends* of e and we write $e : x -_G y$. In both cases, e is a *loop* if $x = y$. We denote by $Und(G)$ the undirected graph obtained from G by taking as incidence function $vert_{Und(G)}(e) := \{x,y\}$ whenever $vert_G(e) = (x,y)$. Graphs can have loops and multiple edges; we *do not identify* an edge with the pair or the set of its ends.

Walks and paths
Let G be a graph and $x,y \in V_G$. A *walk* from x to y is a sequence $(x_0, e_1, x_1, e_2, ..., e_n, x_n)$ such that $x_0, x_1, ..., x_n \in V_G$, $x_0 = x$, $x_n = y$, $e_1, ..., e_n \in E_G$, $e_i : x_{i-1} \to x_i$ ($e_i : x_{i-1} - x_i$ if G is undirected) for each $i = 1,...,n$, and $e_i \neq e_j$ if $1 \leq i < j \leq n$. It is a *path* if we (also) have $x_i \neq x_j$ for $0 \leq i < j \leq n$, except possibly if $i = 0$ and $j = n$. A walk is *closed* if $x_0 = x_n$. A *directed cycle* is a closed path in a directed graph. A *cycle* is similar in an undirected graph (a cycle with two vertices consists of two parallel edges).

An edge never occurs twice in a walk. A vertex never occurs twice in a path except if $x_0 = x_n$. A walk $(x_0, e_1, x_1, e_2, ..., e_n, x_n)$ of a directed graph can be described without ambiguity by the sequence $(e_1, e_2, ..., e_n)$. A walk, a path or a cycle in a directed graph is said to be *undirected* if its edges can be traversed in any direction (i.e., with $e_i : x_{i-1} \to x_i$ or $e_i : x_i \to x_{i-1}$ in the above definition).

A directed graph is *strongly connected* if, for any two distinct vertices x and y, there is path from x to y. The class of strongly connected graphs is denoted by $\mathcal{SC}$. As usual, a directed graph G is *connected* if $Und(G)$ is connected and similarly for notions defined for undirected graphs.

Subgraphs
We write $G \subseteq H$ (resp. $G \subseteq_i H$) if G is a subgraph (resp. an induced subgraph) of H. If $F \subseteq V_G \cup E_G$, then $G - F$ is the subgraph of G obtained by deleting the edges and vertices in F and the edges incident with a vertex in F. We write it $G - x$ if $F = \{x\}$. If $X \subseteq V_G$, we denote by $G[X]$ the graph $G - (V_G - X)$: it is the induced subgraph of G with vertex set X.

Edge-cuts
A *bridge* in a connected graph is an edge whose removal disconnects the graph. A *cut-pair* is a pair of edges that are not bridges and whose removal disconnects the

graph. A strongly connected graph G has no bridge. Conversely, if an undirected graph H is *2-edge-connected*, i.e., is connected and has no bridge, then it is *Und(G)* for some strongly connected graph G.

We now review some definitions and results from the preliminary sections of [9, 15] and [17]. Let G be strongly connected. If F is a cut-pair of G, and G_1 and G_2 the two connected components of $G - F$, then one edge of F links G_1 to G_2 and the other G_2 to G_1; we say that F *separates* x and y if $x \in V_{G_1}$ and $y \in V_{G_2}$. We define an equivalence relation on V_G by $x \sim y$ if and only if x and y are not separated by any cut-pair, if and only they are linked by 3 edge-disjoint undirected paths (by a classical result by Menger, cf. [7], Theorem 3.3.6). We denote it by $\sim_G$ if G must be specified.

The quotient graph $H := G/\sim$ is defined as follows: its vertices are the equivalence classes $[x]_\sim$ of the vertices x of G; it has edge $e : [x]_\sim \rightarrow_H [y]_\sim$ if and only if $[x]_\sim \neq [y]_\sim$ and $e : u \rightarrow_G v$ where $u \in [x]_\sim$ and $v \in [y]_\sim$. (It is useful to designate in the same way an edge e of G and its "quotient" in H; their end vertices are of course different in G and H). Then $Und(H)(= Und(G)/\sim)$ is a *cactus* (a connected undirected graph whose biconnected components are cycles) without bridges. Since G is strongly connected, these cycles are directed cycles in H. We will say that H is a *strongly connected cactus*, i.e., a strongly connected graph whose biconnected components are directed cycles. Note that a directed graph is a strongly connected cactus if and only if, for every two distinct vertices x and y there is a unique directed path from x to y.

Isomorphisms

An *isomorphism* of $G = (V_G, E_G, vert_G)$ to $G' = (V_{G'}, E_{G'}, vert_{G'})$ is a bijection $h : V_G \cup E_G \rightarrow V_{G'} \cup E_{G'}$ that maps vertices to vertices, edges to edges and preserves incidences, that is : $vert_{G'}(h(e)) = (h(x), h(y))$ (resp. $\{h(x), h(y)\}$) if $vert_G(e) = (x, y)$ (resp. $vert_G(e) = \{x, y\}$). It is a *v-isomorphism* if $V_G = V_{G'}$. In this case, one can consider E_G and $E_{G'}$ as different sets of names used to designate the edges of a graph with vertex set V_G. If $G' = G$, we get the notions of *automorphism* and of *v-automorphism.* The graph of Figure 1.2 has several v-automorphisms. We denote by $G \cong G'$ the existence of an isomorphism between G and G'.

Darts

To discuss plane embeddings it will be useful to split a directed edge e into two *darts*: e^-, e^+ with incidences defined by a function γ such that $\gamma(e^-) = x$ and $\gamma(e^+) = y$ if $e : x \rightarrow y$. We denote by D_G^+ the set of darts e^+, by D_G^- the set of darts e^- and by D_G the set $D_G^+ \cup D_G^-$. It is clear that $D_G^+ \subseteq D_{G'}^+$ and $D_G^- \subseteq D_{G'}^-$ if $G \subseteq G'$. We will only use this notion for directed graphs.

1.2.1.2 Graphs with Transitions

Definition 1: (2,2)-*regular graph*

A (2,2)-*regular graph* is a directed, 4-regular graph, each vertex of which has 2 incoming edges and 2 outgoing edges. We denote by $\mathcal{G}_{2,2}$ the class of connected

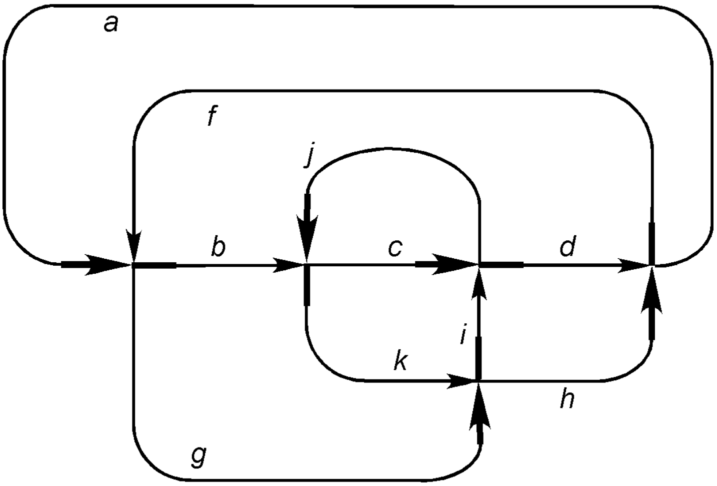


Fig. 1.5 A planar t-map

(2,2)-regular graphs. Each such graph has an *Eulerian tour,* i.e., is covered by a closed walk (*covered* means that the walk goes through all edges). (See [7], Section 1.8, where the proof given for undirected graphs extends easily to directed ones). It is thus strongly connected.

Definition 2: *Graph with transitions*
A (2,2)-*regular graph with transitions* is a pair (G,τ) consisting of a (2,2)-regular graph G and a *transition function* defined as a bijection τ: $D_G^+ \to D_G^-$ such that $\gamma(\tau(d)) = \gamma(d)$ for every $d \in D_G^+$. We say in this case that d and $\tau(d)$ are *opposite* darts. For shortness sake, we will say that there is a *transition from e* to f (or even that (e,f) *is a transition*) if $\gamma(f^-) = \gamma(e^+)$ and $\tau(e^+) = f^-$. If $H = (G,\tau)$, we let $Graph(H)$ denote G, V_H denote V_G and similarly for other items. A *t-graph* is a connected (2,2)-regular graph with transitions. We denote by $\mathcal{G}_{2,2}^t$ the class of t-graphs.

An *isomorphism* or a *v-isomorphism* of t-graphs must respect transitions in an obvious way. (An appendix reviews the different notions of isomorphism and equivalence relation used in this article.)

Definition 3: *Straight walk*
A walk $(x_0,e_1,x_1,e_2,...,e_n,x_n)$ in a (2,2)-regular graph with transitions is *straight* if it is not closed and $\tau(e_i^+) = e_{i+1}^-$ for each $i = 1,...,n-1$ or if it is closed and, in addition to this condition, $\tau(e_n^+) = e_1^-$. Every t-graph is the union of a set of pairwise edge-disjoint closed straight walks, in a unique way.

Figure 1.5 shows a (plane embedding of a) t-graph. The transition function is represented by thickening some darts: two bold darts are related by the transition, and so are two light darts. For example $\tau(a^+) = b^-$ and $\tau(f^+) = g^-$. This graph is covered by the closed straight walks (a,b,c,d) and (f,g,i,j,k,h). Its Eulerian tour (a,b,c,d,f,g,i,j,k,h) is not straight.

1.2.2 Maps

Maps are combinatorial objects that represent embeddings of connected graphs in oriented surfaces, up to orientation preserving homeomorphisms. We review the classical definitions (cf. [14], Chapter 3 for detailed definitions), and we introduce some new notions.

If $\mathcal{E}$ is an embedding of a graph G in a surface, we denote by $\mathcal{E}(u)$ the point representing a vertex u, by $\mathcal{E}(e)$ the line segment representing an edge e and by $\mathcal{E}(W)$ the union of the segments representing the edges of a walk W. As usual, we call *plane* an embedding of a graph in the sphere, and we define a *homeomorphism of plane embeddings* as a homeomorphism of the sphere that maps an embedding onto the other.

Definition 4: *Map*
A *map* is a pair $M = (G,\rho)$ consisting of a connected and directed graph G and a bijection $\rho : D_G \to D_G$ such that, for every d in D_G, the set $\{\rho^i(d) \mid i \geq 0\}$ is the set of darts incident with $\gamma(d)$. This bijection is called the *rotation* of M. We denote G by $Graph(M)$ and D_G by D_M.

From an embedding $\mathcal{E}$ of a connected and directed graph G in an orientable surface, we get a map (G,ρ) by letting $\rho(d)$ be the dart following d in the circular order, "around the vertex $\gamma(d)$" (and according to the orientation of the surface) of the darts incident with $\gamma(d)$. For any two embeddings of G in the sphere with same associated map, there is an orientation preserving homeomorphism of the sphere that maps $\mathcal{E}$ to $\mathcal{E}'$. (See [14], Theorem 3.2.4 for the proof, and a more general statement concerning orientable surfaces.) A map is *planar* if it is associated with a *plane embedding*, i.e., an embedding in the sphere.

If $\mathcal{E}$ is a plane embedding of a connected and directed graph G with map $M = (G,\rho)$, then $M^{-1} := (G,\rho^{-1})$ is the *symmetric map* of M: it corresponds to an embedding $\mathcal{E}'$ of G that is homeomorphic to $\mathcal{E}$ with a reversal of orientation. We say that two maps M and M' are *equivalent* if $M' = M$ or $M' = M^{-1}$. (They have the same underlying graph).

We denote respectively by $\mathcal{MSC}$ and $\mathcal{M}_{2,2}$ the classes of maps of graphs in $\mathcal{SC}$ and in $\mathcal{G}_{2,2}$.

Definition 5: *Transitions defined from rotations*
Let $M = (G,\rho)$ be a map in $\mathcal{M}_{2,2}$. It is a *t-map* if the mapping τ defined by $\tau(d) := \rho^2(d)$ for $d \in D_G^+$ is a transition function. The associated t-graph is $Graph^t(M) := (G,\tau)$. We let $\mathcal{M}_{2,2}^t$ denote the class of t-maps. Two t-maps M and N are *t-equivalent* if $Graph^t(M) = Graph^t(N)$, which we denote by $M \sim^t N$. Clearly, $M \sim^t M^{-1}$. Figure

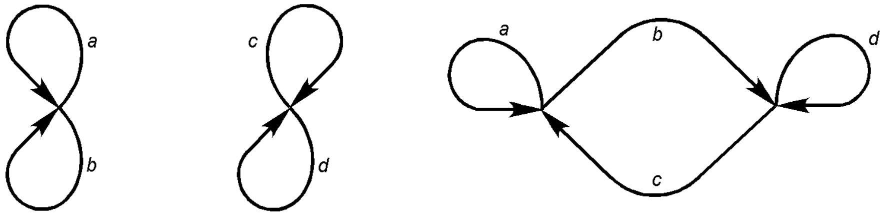


Fig. 1.6 Three planar maps $M, N, P \in \mathcal{M}_{2,2}$

1.4 shows two t-maps M and N that are t-equivalent but not equivalent: they are different but $N \neq M^{-1}$. (Another similar example can be obtained from Figure 1.1).

In Figure 1.6, the planar map M at the left is a t-map because $\rho(a^+) = b^+$, $\rho(b^+) = b^-$ and $\rho(b^-) = a^-$, hence $\rho^2(a^+) = b^-$ and $\rho^2(b^+) = a^-$, so that ρ^2 is a transition function. The map N in the middle is not because $\rho(c^+) = c^-$ and $\rho(c^-) = d^+$, so that $\rho^2(c^+) = d^+$. (We will denote by **8** the corresponding graph in $\mathcal{G}_{2,2}$). Similarly, the rightmost map P is not either because $\rho^2(d^+) = b^+$. The planar map of Figure 1.5 is a t-map.

Definition 6 : *Planar t-graph*
A plane embedding of a t-graph G *respects the transition* if the corresponding map M is such that $Graph^t(M) = G$. A t-graph G is *planar* if it has a plane embedding, hence, if and only if it is $Graph^t(M)$ for some planar t-map M.

We denote by $\mathcal{PSC}$, $\mathcal{PG}_{2,2}$, $\mathcal{PG}^t_{2,2}$, $\mathcal{PMSC}$, $\mathcal{PM}_{2,2}$ and $\mathcal{PM}^t_{2,2}$ the classes of graphs, t-graphs and maps that belong respectively to $\mathcal{SC}$, $\mathcal{G}_{2,2}$, $\mathcal{G}^t_{2,2}$, $\mathcal{MSC}$, $\mathcal{M}_{2,2}$ and $\mathcal{M}^t_{2,2}$ and are planar. These classes of graphs and maps are related by inclusions and by the mappings *Graph* and *Graph^t*. We have in particular :

$$\begin{tikzcd}
\mathcal{M}_{2,2} \arrow[r, phantom, "\supseteq"] \arrow[d] & \mathcal{M}^t_{2,2} \arrow[d] \\
\mathcal{G}_{2,2} & \mathcal{G}^t_{2,2}. \arrow[l]
\end{tikzcd}$$

This square diagram shows that:

$$Graph(M) = Graph(Graph^t(M)) \text{ for every } M \in \mathcal{M}^t_{2,2}.$$

These relations and facts extend to planar graphs and maps. Hence, we have the similar square diagram with analogous meaning:

$$\begin{tikzcd}
\mathcal{PM}_{2,2} \arrow[r, phantom, "\supseteq"] \arrow[d] & \mathcal{PM}^t_{2,2} \arrow[d] \\
\mathcal{PG}_{2,2} & \mathcal{PG}^t_{2,2}. \arrow[l]
\end{tikzcd}$$

The symmetrization mapping $M \mapsto M^{-1}$ preserves each of the classes $\mathcal{MSC}$, $\mathcal{M}_{2,2}$, $\mathcal{M}^t_{2,2}$, $\mathcal{PMSC}$, $\mathcal{PM}_{2,2}$ and $\mathcal{PM}^t_{2,2}$.

1.3 Atoms of Graphs and Maps

As recalled in Section 1.1.1, every strongly connected graph G has a canonical structure of strongly connected cactus whose vertices are its 3-edge-connected components. We call *atoms* these components and *atomic decomposition* the global cactus structure. For our applications to Gauss words, it is useful to express these decompositions by algebraic terms based on an appropriate binary operation that composes strongly connected graphs. This operation also composes maps. We review some definitions and show that they are also applicable to t-graphs and to the graphs, t-graphs and maps of the classes of Definition 6.

1.3.1 Circular Composition of Directed Graphs and Maps

We define an operation that composes directed graphs and maps. We will actually use it mainly to *decompose* these objects.

Definition 7 : *Circular composition*
We let G_1, G_2 be disjoint directed graphs and $e_i \in E_{G_i}$ for $i = 1,2$. We define $G_1 \boxplus_{e_1,e_2} G_2$ as the graph H such that $V_H := V_{G_1} \cup V_{G_2}$, $E_H := E_{G_1} \cup E_{G_2}$ and the incidence function $vert_H$ is defined as follows:

$$vert_H(e_1) := (\alpha(e_1), \beta(e_2)),$$
$$vert_H(e_2) := (\alpha(e_2), \beta(e_1)),$$
$$vert_H(e) := vert_{G_i}(e) \text{ if } e \in E_{G_i} - \{e_i\}, i = 1,2.$$

We call $G_1 \boxplus_{e_1,e_2} G_2$ a *circular composition of* G_1 and G_2. If $G_1, ..., G_k$ are pairwise disjoint and e_i is an edge of G_i for each i, we let :

$$\boxplus_{e_1,...,e_k}(G_1, ..., G_k) := (...(G_1 \boxplus_{e_1,e_2} G_2) \boxplus_{e_2,e_3} G_3)...) \boxplus_{e_{k-1},e_k} G_k$$
$$= \boxplus_{e_1,...,e_{k-1}}(G_1, ..., G_{k-1}) \boxplus_{e_{k-1},e_k} G_k.$$

Here are examples. We let G, H, K be the respective graphs of the maps M, N, P of Figure 1.6 : we have $K = G \boxplus_{b,c} H$. The left part of Figure 1.7 shows graphs $G_1, ..., G_4$ with distinguished edges $e_1, ..., e_4$ (note that e_2 is a loop). The right part shows $\boxplus_{e_1,...,e_4}(G_1, G_2, G_3, G_4)$. This figure explains the terminology.

The following properties are easy to check from definitions.

Proposition 8 : Let G_1, G_2, G_3 be pairwise disjoint directed graphs and $e_i \in E_{G_i}$ for $i = 1,2,3$. We have the following equalities (in each case, both handsides are defined):

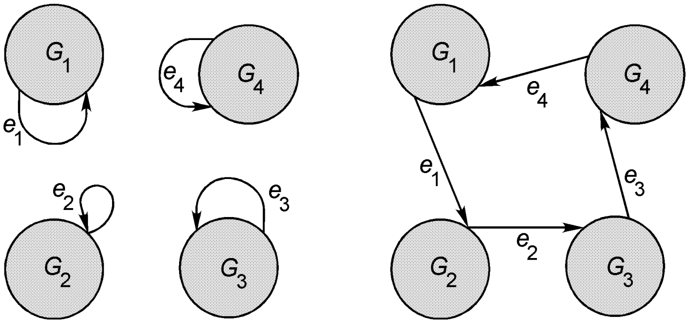


Fig. 1.7 The graph to the right is a circular composition of $G_1,...,G_4$

(1) $G_1 \boxplus_{e_1,e_2} G_2 = G_2 \boxplus_{e_2,e_1} G_1$,
(2) $(G_1 \boxplus_{e_1,e_2} G_2) \boxplus_{e_2,e_3} G_3 = G_1 \boxplus_{e_1,e_3} (G_2 \boxplus_{e_2,e_3} G_3)$,
(3) $(G_1 \boxplus_{e_1,e_2} G_2) \boxplus_{f,e_3} G_3 = G_1 \boxplus_{e_1,e_2} (G_2 \boxplus_{f,e_3} G_3)$ if $f \in E_{G_2} - \{e_2\}$.

From (1) and (2) we get the following *circular associativity* :
$(G_1 \boxplus_{e_1,e_2} G_2) \boxplus_{e_2,e_3} G_3 = (G_2 \boxplus_{e_2,e_3} G_3) \boxplus_{e_3,e_1} G_1$.

The graph $H = \boxplus_{e_1,...,e_4}(G_1,G_2,G_3,G_4)$ at the right of Figure 1.7 can be expressed as $(G_1 \boxplus_{e_1,e_2} G_2) \boxplus_{e_2,e_4} (G_3 \boxplus_{e_3,e_4} G_4)$.

It is easy to check that $G_1 \boxplus_{e_1,e_2} G_2$ is strongly connected if G_1 and G_2 are so. However, the connectedness of G_1 and G_2 does not imply that of $G_1 \boxplus_{e_1,e_2} G_2$: just take $G_1 = e_1$ and $G_2 = e_2$. Furthermore, $\{e_1,e_2\}$ is a cut-pair of $G_1 \boxplus_{e_1,e_2} G_2$. Conversely, if G is strongly connected and has a cut-pair $\{e_1,e_2\}$, then $G = G_1 \boxplus_{e_1,e_2} G_2$ where G_1 and G_2 are strongly connected. This decomposition step will be applied to G_1 and G_2 if possible, and to the graphs resulting from their decompositions.

Definition 9: *Circular composition of t-graphs and maps*
(a) We recall that $\mathcal{G}_{2,2} \subseteq \mathcal{SC}$. If G_1 and G_2 belong to $\mathcal{G}_{2,2}$, then so does any circular composition $G_1 \boxplus_{e_1,e_2} G_2$. If they are t-graphs (i.e., if they belong to $\mathcal{G}^t_{2,2}$), then, we define $H = G_1 \boxplus_{e_1,e_2} G_2$ as the t-graph with vertices and edges as in Definition 7 for graphs, and transition function τ_H defined by :
$\tau_H(e_1^+) := \tau_{G_2}(e_2^+)$,
$\tau_H(e_2^+) := \tau_{G_1}(e_1^+)$,
$\tau_H(d) := \tau_{G_i}(d)$ if $d \in D^+_{G_i} - \{e_i^+\}, i = 1,2$.

Hence, H belongs to $\mathcal{G}^t_{2,2}$ because it is strongly connected and the degree conditions at each vertex are satisfied.

(b) Similarly, for pairwise disjoint strongly connected maps M_1 and M_2, we define a map $N = M_1 \boxplus_{e_1,e_2} M_2$ with underlying graph $H = Graph(M_1) \boxplus_{e_1,e_2} Graph(M_2)$ and rotation ρ_N defined as follows:

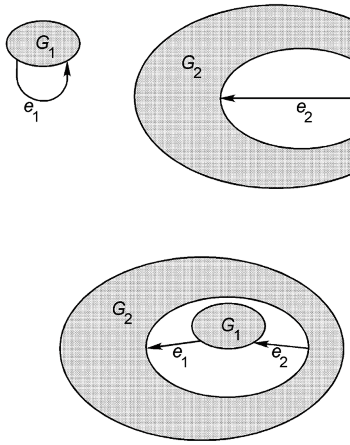


Fig. 1.8 Composition of planar maps

$$\begin{aligned}
&\rho_N(e_1^+) := \rho_{M_2}(e_2^+),\\
&\rho_N(e_2^+) := \rho_{M_1}(e_1^+),\\
&\rho_N(d) := e_1^+ \text{ if } d \in D_{M_2} - \{e_2^+\} \text{ and } \rho_{M_2}(d) = e_2^+,\\
&\rho_N(d) := e_2^+ \text{ if } d \in D_{M_1} - \{e_1^+\} \text{ and } \rho_{M_1}(d) = e_1^+,\\
&\rho_N(d) := \rho_{M_i}(d) \text{ if } d \in D_{G_i}\ , i = 1,2 \text{ and the above cases do not apply.}
\end{aligned}$$

Then, H is strongly connected and N is a map. If furthermore M_1 and M_2 belong to $\mathcal{M}_{2,2}$ or $\mathcal{M}^t_{2,2}$, then N belongs to the same class.

For the maps of Figure 1.6, we have the equality $P = M \boxplus_{b,c} N$. Figure 1.7 shows the circular composition of four planar maps represented in the plane, with distinguished edges drawn ”on the outer face”. Figure 1.8 shows the composition $G_1 \boxplus_{e_1,e_2} G_2$ of two planar maps G_1 and G_2, represented similarly, but with e_2 not on the outer face.

The equational properties of circular composition stated in Proposition 8 are also valid for t-graphs and maps.

In Definitions 7 and 9, circular composition is a partial operation because of the disjointness condition on the arguments. We will use it to *decompose given maps and graphs*, that is, for given H, we will try to find graphs (or maps) G_1 and G_2 such that $H = G_1 \boxplus_{e_1,e_2} G_2$. Hence, the disjointness condition does not raise any difficulty.

By the following proposition, the results about decompositions of strongly connected graphs extend to the classes of graphs, t-graphs and maps of Definition 6.

Proposition 10: (1) If G, H_1, H_2 are directed graphs such that $G = H_1 \boxplus_{e_1,e_2} H_2$, then G is strongly connected (resp. is in $\mathcal{G}_{2,2}$) if and only if H_1 and H_2 are strongly connected (resp. are in $\mathcal{G}_{2,2}$). Furthermore, if G, H_1, H_2 are strongly connected, then G is planar if and only if H_1 and H_2 are planar.

(2) If G is a t-graph such that $Graph(G) = H_1 \boxplus_{e_1,e_2} H_2$ for some directed graphs H_1 and H_2, then there are unique transition functions τ_1, τ_2 such that $G = G_1 \boxplus_{e_1,e_2} G_2$ and for each i, $G_i = (H_i, \tau_i) \in \mathcal{G}^t_{2,2}$. Furthermore, G is planar if and only if G_1 and G_2 are planar.

(3) If M is a strongly connected map (resp. a map in $\mathcal{M}_{2,2}$ or a t-map) such that $Graph(M) = H_1 \boxplus_{e_1,e_2} H_2$ for some directed graphs H_1 and H_2, then there are unique rotations ρ_1, ρ_2 such that $M = N_1 \boxplus_{e_1,e_2} N_2$ where, for each i, $N_i = (H_i, \rho_i)$ is a map (resp. is in $\mathcal{M}_{2,2}$ or in $\mathcal{M}^t_{2,2}$). Furthermore, M is planar and strongly connected if and only if N_1 and N_2 are planar and strongly connected.

Proof: (1) Let G, H_1, H_2 be directed graphs such that $G = H_1 \boxplus_{e_1,e_2} H_2$. Every closed walk of G that goes through e_1 must also go through e_2. It follows then that G is strongly connected if and only if H_1 and H_2 are so. The ”local” conditions on edges for membership in $\mathcal{G}_{2,2}$ are easy to check.

Let G, H_1, H_2 be strongly connected. If H_1 and H_2 are planar, then a plane embedding of G can be built from plane embeddings of H_1 and H_2 (cf. Figures 1.7 and 1.8). Conversely, assume that $G = H_1 \boxplus_{e_1,e_2} H_2$ is planar, with plane embedding $\mathcal{E}$. Let P be a path in G from $x = \alpha(e_1)$ to $y = \beta(e_2)$ (y is a vertex of H_2) that goes through e_1, some edges of H_2 and finally e_2. Then, $\mathcal{E}(P)$ links $\mathcal{E}(x)$ to $\mathcal{E}(y)$. We obtain in this way a plane embedding of H_1. That H_2 is planar is proved similarly. (If we only assume that $H_1 \boxplus_{e_1,e_2} H_2$ is planar and H_2 is connected, then H_1 may not be planar.)

(2) Let G be a t-graph such that $Graph(G) = H_1 \boxplus_{e_1,e_2} H_2$ for some H_1, H_2 in $\mathcal{G}_{2,2}$. The graphs $Graph(G), H_1$ and H_2 are strongly connected. The existence of unique transition functions τ_1, τ_2 such that $G = (H_1, \tau_1) \boxplus_{e_1,e_2} (H_2, \tau_2)$ and $(H_1, \tau_1), (H_2, \tau_2) \in \mathcal{G}^t_{2,2}$ follows easily. From plane embeddings of (H_1, τ_2) and (H_2, τ_2), one can build a plane embedding of G. The converse is proved as in (1).

(3) Let M be a strongly connected map such that $Graph(M) = H_1 \boxplus_{e_1,e_2} H_2$ for some graphs H_1, H_2. These graphs must be strongly connected. The proof goes as for (2). □

1.3.2 Terms Defining Graphs and Maps

Definition 11: *Atoms and terms*
An *atom* is a strongly connected graph that cannot be expressed as $G_1 \boxplus_{e,f} G_2$, equivalently, that has no cut-pair. The latter condition means that it is 3-edge connected, hence that any two vertices are linked by 3 edge-disjoint undirected paths (by a theorem by Menger, cf. [7], Theorem 3.3.6). A t-*atom* (resp. an *atomic* map or t-map) is a t-graph (resp. a map or t-map) that cannot be expressed as the circular composition of two t-graphs or maps. If the context makes things clear, we will call them also atoms. By Proposition 10 they are the t-graphs, maps or t-maps whose underlying graphs are atoms.

The graph $Graph(M) = Graph(N)$ where M,N are at the left of Figure 1.6 and the graph of Figure 1.5 are atoms.

Let $\mathcal{A}$ be a set of pairwise disjoint graphs, t-graphs or maps included in one of the classes of Definition 6. A *term over* $\mathcal{A}$ is a term t built with cicular composition and elements of $\mathcal{A}$ used as constants, each of them having at most one occurrence in the term. It defines a graph, a t-graph or a map, depending on the types of the elements of $\mathcal{A}$. We denote this object by $val(t)$: it is the *value* of t. We also say that *t defines $val(t)$*. We say that *t uses* $\mathcal{A}$ if each element of $\mathcal{A}$ has one (and only one) occurrence in t.

Every strongly connected graph, t-graph or strongly connected map G is defined by a term over atoms (of the corresponding type): if it is defined by a term t using $G_1,...,G_k$ such that G_i is not an atom, then $G_i = H \boxplus_{e,f} K$ and G_i can be replaced in t by $H \boxplus_{e,f} K$. Proposition 8 shows that different terms (even over atoms) can have the same value. We will say that two such terms are *equivalent*. For example, the following two terms define the graph shown in Figure 1.9:

$$t = \boxplus_{e_1,e_2,e_3}(G_1,[(G_2 \boxplus_{h_2,h_7} G_7) \boxplus_{f_2,f_4} (\boxplus_{g_4,g_5,g_6}(G_4,G_5,G_6))],G_3)$$
$$t' = G_7 \boxplus_{h_7,h_2} [\boxplus_{e_1,e_2,e_3}(G_1,G_2,G_3) \boxplus_{f_2,f_4} (\boxplus_{g_4,g_5,g_6}(G_4,G_5,G_6))].$$

Theorem 12 : There exists a linear-time algorithm that constructs, for every strongly connected graph, its set of atoms and a term over atoms that defines it.

Proof: As observed in Definition 11, every strongly connected graph G is defined by a term over atoms.

The linear-time algorithm of [17] constructs the vertex sets of the 3-edge-connected components of $Und(G)$, i.e., the equivalence classes $[x]_\sim$ for $x \in V_G$, hence, also in linear time, the strongly connected cactus H underlying the atomic decomposition of G.

We denote by $A(G)$ the set $\{[x]_\sim \mid x \in V_G\}$. The graph H has vertex set V_H equal to (or in an implementation, in bijection with) $A(G)$. Its set of edges E_H consists of the edges of G whose ends are in different members of $A(G)$. If e links x to y in G, then it links $[x]_\sim$ to $[y]_\sim$ in H if $[x]_\sim \neq [y]_\sim$.

From this cactus we now build a term t_G over atoms as desired. We use an induction on its size.

If H has a unique vertex, then G is an atom and t_G is this atom.

Otherwise, Lemma 2.1 of [9] proves that $\{e,f\}$ is a cut-pair of G if and only if it is one of H. Let us choose a vertex $[x]_\sim$ and edges $e : [x]_\sim \longrightarrow [y]_\sim$ and $f : [z]_\sim \longrightarrow [x]_\sim$ that belong to a same cycle of H. Then $G = G_1 \boxplus_{e,f} G_2$ where G_1 and G_2 are defined as follows:

X_1 is the union of the sets $[u]_\sim$ that are the vertices of the connected component of $H - \{e,f\}$ that contains $[x]_\sim$,
$X_2 := V_G - X_1$,
G_1 is $G[X_1]$ augmented with edge $e : \alpha_G(e) \longrightarrow \beta_G(f)$,
G_2 is $G[X_2]$ augmented with edge $f : \alpha_G(f) \longrightarrow \beta_G(e)$.

Hence, we can take $t_G := t_{G_1} \boxplus_{e,f} t_{G_2}$. The terms t_{G_1} and t_{G_2} can be constructed from the cactuses H_1 and H_2 of G_1 and, respectively, G_2 that, by Lemma 2.1 of [9],

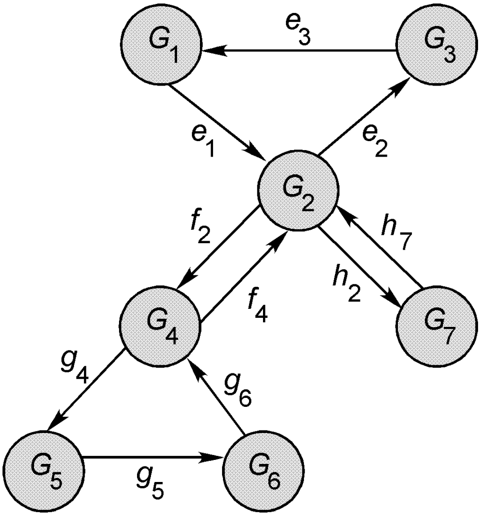


Fig. 1.9 The graph defined by term t

can be obtained as follows in constant time from H, $e : [x]_\sim \longrightarrow [y]_\sim$ and $f : [z]_\sim \longrightarrow [x]_\sim$:

> H_1 is the connected component of $H - \{e, f\}$ that contains $[x]_\sim$ and H_2 is the other connected component of $H - \{e, f\}$ augmented with the edge $f : [z]_\sim \longrightarrow [y]_\sim$ if $[z]_\sim \neq [y]_\sim$. (The edge e is not an edge of H_1 because after its redirection towards $\beta_G(f)$, it is an edge of $G_1[[x]_\sim]$ and similarly for f in H_2 if $[z]_\sim = [y]_\sim$.)

It follows that t_G can be constructed in linear time. □

The set of atoms of t_G is the same for all such terms and is called *the set of atoms of G*. The *atomic decomposition* of G is the cactus whose vertices are its atoms. Figure 1.9 shows an example.

Theorem 13: Let $\mathcal{C}$ be any one of the classes $\mathcal{PSC}$, $\mathcal{G}_{2,2}$,$\mathcal{PG}_{2,2}$, $\mathcal{G}^t_{2,2}$, $\mathcal{PG}^t_{2,2}$, $\mathcal{MSC}$, $\mathcal{PMSC}$, $\mathcal{M}_{2,2}$, $\mathcal{PM}_{2,2}$, $\mathcal{M}^t_{2,2}$ or $\mathcal{PM}^t_{2,2}$. Every element of $\mathcal{C}$ is defined by a term over atoms of $\mathcal{C}$. There exists a linear-time algorithm that constructs, for every element of $\mathcal{C}$, a term over atoms that defines it.

Proof: The proof is the same for all cases. Let us do it for, say, $\mathcal{C} = \mathcal{PG}^t_{2,2}$.

Let $G \in \mathcal{C}$. The graph $H = Graph(G)$ is strongly connected, hence defined by a term t using the atoms $H_1, ..., H_k$. By means of an induction on the structure of t, Proposition 10 entails the existence of unique transition functions $\tau_1, ..., \tau_k$ such that $(H_i, \tau_i) \in \mathcal{C}$ and G is defined by a term using $(H_1, \tau_1), ..., (H_k, \tau_k)$. Since $H_1, ..., H_k$ are atoms in $\mathcal{SC}$, $(H_1, \tau_1), ..., (H_k, \tau_k)$ are atoms in $\mathcal{C}$.

For proving the second assertion, consider a term s' using the atoms $G'_1, ..., G'_\ell$ in $\mathcal{C}$ that defines G. We get from it a term s using the atoms $Graph(G'_1), ...,$ $Graph(G'_\ell)$ in $\mathcal{SC}$ that defines H. Hence, $k = \ell$ and $\{H_1, ..., H_k\} = \{Graph(G'_1),$

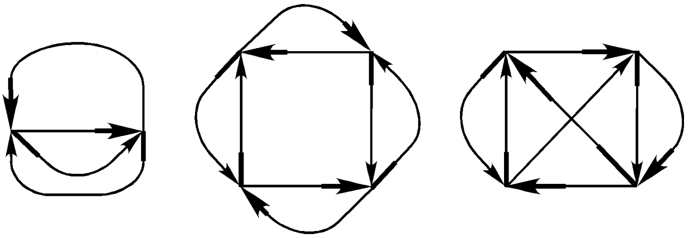


Fig. 1.10 Three atomic t-maps

$..., Graph(G'_k)\}$. Since in Proposition 10(2) the transition functions τ_1, τ_2 are uniquely defined, we have $\{(H_1, \tau_1), ..., (H_k, \tau_k)\} = \{G'_1, ..., G'_k\}$.

The algorithm of Theorem 12 can construct the atoms $H_1, ..., H_k$ of H. By Definition 9, one can get the transition functions $\tau_1, ..., \tau_k$ in linear-time. □

Figure 1.5 shows an atomic planar t-map. Figure 1.6 shows an atomic planar t-map M, and an atomic planar map N that is not a t-map. The first two drawings of Figure 1.10 represent atoms that are planar considered as graphs, t-graphs or maps. The third one shows an atom that is planar as a graph but is not planar as a t-graph or map.

We will need the following proposition of independent interest. A loop e of $G \in \mathcal{G}^t_{2,2}$ has a *transition to itself* if $\tau(e^+) = e^-$.

Proposition 14: There is a linear-time algorithm that checks if a t-graph is planar, and constructs if possible a plane embedding.

Proof: Let $G \in \mathcal{G}^t_{2,2}$. If $Graph(G)$ has 1 or 2 vertices, we check if G is planar by looking at its transition function.

Otherwise, we first eliminate the loops. Let G have a loop e at vertex u. If e has a transition to itself, then G is not planar. Otherwise, we remove e, we fuse the two remaining edges incident with u (hence, we also remove u) and we repeat these removals until there is no more loop or we get a t-graph with 1 or 2 vertices. We obtain a t-graph $G' \in \mathcal{G}^t_{2,2}$ that is planar if and only if G is. If G' has 1 or 2 vertices, we conclude by looking at the transition function.

Otherwise G' has at least three vertices and no loop, and we construct a graph H as follows :

(1) We subdivide each edge e into two edges by putting a new vertex x_e in its ”middle”.

(2) Let u be any vertex. For any two edges e and $f \neq e$ incident with u such that there is no transition at u between e and f, we add an edge between x_e and x_f. We say that we *make G' rigid at u*. We define H by making G' rigid at all its vertices.

It is clear that H is planar if and only if G' is, and that every plane embedding of H yields one of the t-graph G' (that respects its transitions), from which we get easily one of the given t-graph G.

The constructions of G' from G and of H from G' can be done in time $O(|V_G| + |E_G|)$; the size of H defined as $|V_H| + |E_H|$ is also $O(|V_G| + |E_G|)$. The classical linear-time planarity test applied to H gives the answer. This algorithm constructs plane embeddings of H, G' and G when they exist. □

1.4 Planar t-Graphs and t-Maps

In order to answer our initial questions about curves in the plane, we now focus our attention on planar t-graphs and t-maps.

We first recall a basic fact. Let M be a map and $G = Graph(M)$. If P is an undirected path in G from x to y (cf. Section 1.1.1 for definitions) whose edge sequence is $(e_1, ..., e_k)$, if d is the dart of e_1 such that $\gamma(d) = x$ and d' is that of e_k such that $\gamma(d') = y$, then we say that d and d' are the *end darts* of P. Let M be planar, and let P_1, P_2, P_3 be vertex-disjoint undirected paths from x to y with respective end darts d_1, d_2, d_3 that are incident with x and d'_1, d'_2, d'_3 incident with y. If the circular order of d_1, d_2, d_3 (around x) is (d_1, d_2, d_3), then that of d'_1, d'_2, d'_3 around y is (d'_3, d'_2, d'_1) (see [4]). It follows that the circular order of d_1, d_2, d_3 determines that of d'_1, d'_2, d'_3.

We recall from Definition 5 that two maps M and N in $\mathcal{M}^t_{2,2}$ are t-equivalent if and only if $Graph^t(M) = Graph^t(N)$, and that they are equivalent if and only if $M = N$ or $M = N^{-1}$. Two equivalent maps are t-equivalent. For atoms, we have the following converse.

Theorem 15: If two maps M and N in $\mathcal{PM}^t_{2,2}$ are t-equivalent and $Graph(M)$ is an atom, then, they are equivalent. A planar t-atom has a unique plane embedding.

As atoms are 3-edge connected, this result is similar to the fact that a 3-connected planar graph has a unique plane embedding ([7,14]). In the following definition and proofs, we will consider undirected paths and walks in directed graphs, that we will simply call paths and walks.

Definition 16: Let $G \in \mathcal{G}^t_{2,2}$ and let P and Q be two edge-disjoint walks from x to $y \neq x$. We say that they *cross at a vertex u* belonging to P and Q if $u \neq x, u \neq y$ and the two darts of the edges of P that are incident with u are opposite (and so must be the two darts of the edges of Q that are incident with u). If P and Q are vertex-disjoint (except for x and y), they do not cross (at any vertex).

Similarly, we say that *P crosses itself at u* if it is of the form $(..., e, u, e', ..., f, u, f', ...)$ where the darts of e and e' incident with u are opposite (and so must be the darts of f and f' incident with u).

Lemma 17: Let $G \in \mathcal{G}^t_{2,2}$ be atomic. Between any two distinct vertices, there are three edge-disjoint paths that pairwise do not cross.

Proof: If $G = \mathbf{8}$ the assertion is trivially true. Otherwise, since G is atomic, there are three edge-disjoint paths between any two distinct vertices (cf. Definition 11). Let P, Q, R be edge-disjoint paths from x to $y \neq x$ and let p be the total number of vertices at which they cross. Let u be a vertex at which P and Q cross. Since G is (2,2)-regular, u is not on R. Then $P = P_1P_2$ and $Q = Q_1Q_2$ such that the paths P_1 and Q_1 go from x to u, the paths P_2 and Q_2 go from u to y. Then, P_1Q_2 and Q_1P_2 are two walks from x to y. They are still edge-disjoint (and also edge-disjoint with R). The total number of self-crossings of P_1Q_2 and Q_1P_2 and of crossings between P_1Q_2, Q_1P_2 and R is $p-1$. If P_1Q_2 is not a path, we can shorten it into a path P' from x to y by removing some closed subwalks. Similarly, we can shorten Q_1P_2 into a path Q' from x to y. We get a triple P', Q', R of edge-disjoint paths from x to y whose total number of crossings is at most $p-1$ (because the shortenings of P_1Q_2 and Q_1P_2 into P' and Q' can only reduce the number of crossings). By repeating this step, we get three edge-disjoint paths from x to y that pairwise do not cross. □

Proof of Theorem 15: Let M, $N \in \mathcal{PM}^t_{2,2}$ be t-equivalent such that $Graph(M)$ is an atom. If $Graph(M) = \mathbf{8}$, then the result is clear. Otherwise, $G = Graph(M)$ is loop-free because if it has a loop, it has two edges e, f such that $G = H_1 \boxplus_{e,f} H_2$ and is not an atom.

Let $x \in V_G$. If the rotations ρ_M and ρ_N are not the same on the darts incident with x, we have $\rho_M = \rho_{N^{-1}}$ at x because $Graph^t(M) = Graph^t(N)$ and so, there are only two rotations around a vertex that yield a given transition function (cf. Definition 5). We now replace N by N^{-1}, and then ρ_M and ρ_N are the same on the darts incident with x.

We now prove that for every vertex $y \neq x$, the rotations ρ_M and ρ_N are the same on the darts incident with y. We first consider the case where x and y are linked by three vertex-disjoint paths. Then, since the circular order of the corresponding three end darts incident with x is the same in M and in N, the same holds for the circular order of the three darts around y because of the three vertex-disjoint paths. The position of the fourth dart incident with y among the first three is determined by the transition. Since we have $Graph^t(M) = Graph^t(N)$, the rotations ρ_M and ρ_N are the same on the four darts incident with y.

We now consider the general case, where x and y are linked by three edge-disjoint paths P, Q, R that are not vertex-disjoint. By Lemma 17, we can assume that they pairwise do not cross. Let W be the set of vertices different from x and y that belong to two of these paths. We add vertices to subdivide each edge of G that has an end in W and we also add edges between these new vertices, so as to make G rigid at all vertices of W (this notion is defined in the proof of Proposition 14). We obtain a graph H. Thanks to the newly added edges and because P, Q, R do not cross pairwise, x and y are linked in H by three vertex-disjoint paths P', Q', R' that avoid the vertices of W. We identify in a obvious way the darts of G incident with x and y with the corresponding ones in H.

Consider plane embeddings $\mathcal{E}$ and $\mathcal{E}'$ of M and N that respect the transition relations. By adding some line segments to $\mathcal{E}$, one can make it into a plane embedding $\mathcal{E}_1$ of H, and similarly, one can make $\mathcal{E}'$ into a plane embedding $\mathcal{E}'_1$ of H. The circular orders of darts around x are the same in M and N, and thus also in $\mathcal{E}_1$ and $\mathcal{E}'_1$. Since we have three vertex-disjoint paths P',Q',R' between x and y, the corresponding circular orders of their end darts around y are the same in $\mathcal{E}_1$ and $\mathcal{E}'_1$, hence in M and in N. As observed above, and since M and N are t-equivalent, the rotations ρ_M and ρ_N are the same on the four darts incident with y.

We obtain that $M = N$, which gives the result. (We recall that the initial map N may have been replaced by N^{-1} at the beginning.) □

Our objective is now to describe all maps in $\mathcal{PM}^t_{2,2}$ that are t-equivalent to a given map. For doing that we will use terms over atoms.

Lemma 18: Let $M,M',N,N',P,P' \in \mathcal{PM}^t_{2,2}$ be such that $M \sim^t M'$, $M = N \boxplus_{e,f} P$, $M' = N' \boxplus_{e,f} P'$, $Graph(N) = Graph(N')$ and $Graph(P) = Graph(P')$. Then $N \sim^t N'$ and $P \sim^t P'$.

Proof: The transition functions of $Graph^t(N)$ and $Graph^t(M)$ are related as follows:

$$\tau_{Graph^t(N)}(e^+) = \tau_{Graph^t(M)}(f^+),$$
$$\tau_{Graph^t(N)}(d) = \tau_{Graph^t(M)}(d) \text{ for every } d \in D^+_{Graph^t(N)} - \{e^+\},$$

and similarly for M' and N'. Since $Graph^t(M) = Graph^t(M')$, we have $Graph^t(N) = Graph^t(N')$ and similarly, $Graph^t(P) = Graph^t(P')$. Hence $N \sim^t N'$ and $P \sim^t P'$. □

Let t be a term over atomic maps. We define as follows a set $\nabla(t)$ of terms written with atomic maps, the circular composition $\boxplus$ and also the symmetrization $^{-1}$ that is well-defined on maps:

$$\nabla(t) := \{t, t^{-1}\} \text{ if } t \text{ is an atomic map,}$$
$$\nabla(t) := \{s' \boxplus_{e,f} s'' \mid s' \in \nabla(t'), s'' \in \nabla(t'')\} \text{ if } t = t' \boxplus_{e,f} t''.$$

Since for maps M and N we have $(M \boxplus_{e,f} N)^{-1} = M^{-1} \boxplus_{e,f} N^{-1}$, we need not add the terms $(s' \boxplus_{e,f} s'')^{-1}$ to $\nabla(t)$ in the second case. In other words, the operation $^{-1}$ can be used equivalently anywhere in a term or only on atoms.

Theorem 19: Let $M \in \mathcal{PM}^t_{2,2}$ be defined by a term t over atomic t-maps. The maps that are t-equivalent to M are those defined by the terms in $\nabla(t)$.

Proof: We use an induction on the structure of t. If t is an atom, the result follows from Theorem 15.

Otherwise, $t = t' \boxplus_{e,f} t''$. Lemma 18 shows that the t-maps that are t-equivalent to M are those of the form $N \boxplus_{e,f} P$ where $N \sim^t val(t')$ and $P \sim^t val(t'')$. By induction, these maps N and P are those defined respectively by the terms in $\nabla(t')$ and in $\nabla(t'')$. Hence, the t-maps t-equivalent to M are those defined by the terms in $\nabla(t)$. □

Corollary 20: Let $M \in \mathcal{PM}^t_{2,2}$ be defined by a term t using p atomic maps. There are 2^{p-1} pairwise inequivalent maps in $\mathcal{PM}^t_{2,2}$ for the t-graph $Graph^t(M)$. One can determine them from t.

Proof: We first observe that for all M and N in $\mathcal{PM}^t_{2,2}$, the t-maps $M \boxplus_{e,f} N$ and $M \boxplus_{e,f} N^{-1}$ are t-equivalent but not equal. To prove this, we let $P = M \boxplus_{e,f} N$ and $P' = M \boxplus_{e,f} N^{-1}$. Then, $\rho_P(e^+) = \rho_N(f^+)$, $\rho_{P'}(e^+) = \rho_{N^{-1}}(f^+)$ and $\rho_{N^{-1}}(f^+) \neq \rho_N(f^+)$ because the head of f has degree 4 in N. Hence, $P \neq P'$. Furthermore, $P'^{-1} = (M \boxplus_{e,f} N^{-1})^{-1} = M^{-1} \boxplus_{e,f} N$, hence $P \neq P'^{-1}$ by the same argument and $M \boxplus_{e,f} N$ and $M \boxplus_{e,f} N^{-1}$ are not equivalent either.

It follows that two distinct terms t' and t'' in $\nabla(t)$ define different (but t-equivalent) maps. The set $\nabla(t)$ contains 2^p terms that define 2^p maps. Each of these maps Q is equivalent to a unique map Q^{-1} in the same set, and $Q \neq Q^{-1}$ because these maps are 4-regular. Hence, we obtain exactly 2^{p-1} pairwise inequivalent maps. By fixing one atom of a term t and by replacing in all possible ways some of the other atoms M by M^{-1}, we obtain 2^{p-1} terms that define the 2^{p-1} pairwise inequivalent maps in $\mathcal{PM}^t_{2,2}$ for the t-graph $Graph^t(M)$. □

For an example, the multiword $W = (abcd, bc, ad)$ is ambiguous (see the introduction or the next section for the definition) since it is associated with the two triples of curves of Figure 1.4. The t-graph ${}^tGra(W)$ has two atoms, hence, has two inequivalent planar maps, shown in Figure 1.4.

Remark **:** The maps $M \boxplus_{e,f} N$ and $M \boxplus_{e,f} N^{-1}$ are not equivalent because M and N are 4-regular. For comparison, we have $P = P^{-1}$ if P is the (unique) map of a path or a directed cycle. It follows that $M \boxplus_{e,f} N = M \boxplus_{e,f} N^{-1}$ if N is the map of a directed cycle containing the edge f.

1.5 Curves in the Plane

1.5.1 Oriented Curves

1.5.1.1 Definitions and Basic Facts

We make precise some definitions sketched in the introduction.

Circular sequences
We denote by V^* the set of possibly empty sequences of elements of a finite set V, also considered as *words* over V (whence the term "Gauss word"). The empty sequence is denoted by ε. Two sequences u and v are *conjugate*, which we denote by $u \equiv v$, if $u = w_1w_2$ and $v = w_2w_1$ for some $w_1, w_2 \in V^*$. This relation is an equivalence and its classes are the *circular sequences* of elements of V. Their set is denoted by $V^{\circledast}$. We denote by $Conj(u)$ the set of sequences conjugate with u.

The *reversal* (or *mirror image*) of a sequence u is denoted by $\widetilde{u}$. Two sequences u and v are *reversal-conjugate*, which we denote by $u \asymp v$, if $u \equiv v$ or $u \equiv \widetilde{v}$. This is also an equivalence relation. Its equivalence classes are the *reversal-circular* sequences over V and their set is denoted by $V^{\circledcirc}$. We will usually designate an element of $V^{\circledast}$ or $V^{\circledcirc}$ by some sequence of the corresponding equivalence class.

A *double occurrence multiword* over V is a tuple of circular sequences over V where each element of V has two occurrences (either two occurrences in one sequence or one occurrence in two sequences). We denote by $DO_n(V)$ the set of such n-tuples. If $X \subseteq V$ and $W \in DO_n(V)$, we denote by $W \upharpoonright X$ the multiword in $DO_n(X)$ obtained by removing from the sequences composing W the elements not in X. If $W, W' \in DO_n(V)$, we write $W \asymp W'$ if, for each i, the i-th component of W is $\asymp$-equivalent to the i-th component of W'.

Curves in the plane

Let $\mathcal{C}$ be a closed oriented curve in the plane, with finitely many self-intersections but no triple (or more complex) self-intersection. The term *oriented* means that $\mathcal{C}$ is given with a traversal direction. We let V be the set of self-intersection points, or more precisely, a set of letters naming these points. By following the curve from some of these points according to its traversal direction, we get a circular sequence $w(\mathcal{C})$ of elements of V where each element of V has two occurrences (hence, $(w(\mathcal{C})) \in DO_1(V)$). If $\mathcal{C}$ has no self-intersection, i.e., if it is homeomorphic to a circle, then, $w(\mathcal{C}) = \varepsilon$. The sequence $w(\mathcal{C})$ is circular because the starting point is arbitrary. If $\mathcal{C}$ is not oriented, then $w(\mathcal{C}) \in V^{\circledcirc}$ because the traversal direction is also arbitrary. Such sequences are called *Gauss words*, and have been studied in many articles (e.g., [3, 8, 11, 12, 16]).

Some sequences, for instance *abab*, are not Gauss words as one checks easily. (But a curve yielding it can be drawn on the torus). The Gauss word *abcabc* represents the curve of Figure 1.2. A *square*, i.e., a sequence of the form *uu* (where u is a nonempty sequence) is a Gauss word if and only if u has odd length.

We wish to characterize the cases where $\mathcal{C}$ can be reconstructed from $w(\mathcal{C})$, in a unique way up to homeomorphism. This is not always the case : Figure 1.1 shows two curves that do not correspond via any homeomorphism of the sphere but have the same Gauss word *aabb*. We will characterize the Gauss words which correspond to a unique curve, where as usual, unicity is understood up to homeomorphism.

Tuples of curves and graphs

We will consider more generally *finite tuples of intersecting and self-intersec- ting closed curves* on the plane. With such an n-tuple of oriented curves $(\mathcal{C}_1, ..., \mathcal{C}_n)$, we associate the n-tuple of circular sequences $W(\mathcal{C}_1, ..., \mathcal{C}_n) = (w(\mathcal{C}_1), \ ..., w(\mathcal{C}_n)) \in DO_n(V)$ where V is the set of intersection and self-intersection points, or rather a set of names for these points. We call $W(\mathcal{C}_1, ..., \mathcal{C}_n)$ a *Gauss multiword.* Examples are $(\varepsilon, \varepsilon, \varepsilon)$, $(abcd, bfde, \ aecf)$ and $(abcd, abcd)$. The first one corresponds to three disjoint circles, the second one corresponds to the three curves of Figure 1.3 and the third one to two ellipses with 4 intersections. This last example shows that a same sequence may occur twice in a multiword. The 2-tuple (a, a) represents two curves on the torus with a single intersection, but does not represent two curves on the plane, this is a consequence of the following proposition.

Proposition 21: Each component of a Gauss multiword has even length. The number of letters that occur in any two components of a Gauss multiword is even.

Proof: It follows from the Jordan Curve Theorem (see [14], Chapter 2) that any two closed curves without self-intersections (they are homeomorphic to circles) cross at an even number of points. Hence, the two assertions hold for the Gauss multiwords associated with curves without self-intersections. (They are the Gauss multiwords such that each letter occurs in two different components.)

We now prove the general case by induction on the total number of self-intersections of the curves $\mathcal{C}_1,...,\mathcal{C}_n$ that define the considered multiword. We let $W = W(\mathcal{C}_1,...,\mathcal{C}_n)$ and, without loss of generality, we assume that $\mathcal{C}_1$ has a self-intersection named a. Hence $w(\mathcal{C}_1) \equiv auav$. One can delete the self-intersection a by replacing $\mathcal{C}_1$ by $\mathcal{C}'_1$ such that $w(\mathcal{C}'_1) = u\widetilde{v}$. The two assertions hold for W if they hold for $W(\mathcal{C}'_1,\mathcal{C}_2,...,\mathcal{C}_n)$. The result follows. □

t-graphs from multiwords

We will mainly consider *connected* tuples of curves $(\mathcal{C}_1,...,\mathcal{C}_n)$, i.e., tuples whose union is a connected subset of the sphere (in which we define our "plane embeddings"). If $n > 1$, each curve has intersections with other curves. Except if the tuple consists of one circle, the union of $\mathcal{C}_1,...,\mathcal{C}_n$ is a plane embedding $\mathcal{E}$ of a (nonempty) t-graph $G = G(\mathcal{C}_1,...,\mathcal{C}_n) \in \mathcal{PG}^t_{2,2}$ that we now define. We split each curve $\mathcal{C}_i$ into a union of consecutive segments $s_{i,1}, s_{i,2},..., s_{i,n_i}$ whose ends are the intersection and self-intersection points, and we let V be (in bijection with) the set of ends of these segments. We define G as follows:

its vertex set V_G is V,
its edges are the pairs (i,j) such that $1 \leq i \leq n$ and $1 \leq j \leq n_i$,
$vert_G((i,j))$ is the pair (v,w) such that the segment $s_{i,j}$ links v to w,
the transition is defined by : $\tau_G((i,j)^+) := (i,j+1)^-$ if $1 \leq j < n_i$, and
$\tau_G((i,n_i)^+) := (i,1)^-$.

A plane embedding $\mathcal{E}$ of G is defined by $\mathcal{E}((i,j)) := s_{i,j}$. It is clear that G is a planar t-graph (in particular because $(\mathcal{C}_1,...,\mathcal{C}_n)$ is assumed connected) and that $\mathcal{E}$ respects its transition (cf. Definition 6). This t-graph has n closed straight walks $\omega_1,...,\omega_n$: the edges of ω_i are those of the form (i,j) and $\mathcal{E}(\omega_1)$,..., $\mathcal{E}(\omega_n)$ are the curves $\mathcal{C}_1,...,\mathcal{C}_n$.

For the example of Figure 1.3 with trigonometric orientation of the curve, the graph $G(\mathcal{C}_1,\mathcal{C}_2,\mathcal{C}_3)$ has vertices $\{a,b,...,f\}$, edges $a \to b$, $b \to c$, $c \to d$,..., $b \to f$, $f \to d$,..., $a \to e$, $e \to c$,..., $f \to a$ and transitions $(a \to b, b \to c)$, $(b \to c, c \to d)$, ..., $(b \to f, f \to d)$, ... , $(c \to f, f \to a)$.

We have a bijection between the edges of $G(\mathcal{C}_1,...,\mathcal{C}_n)$ and the segments $s_{i,j}$ of the curves $\mathcal{C}_1,...,\mathcal{C}_n$. Every other plane embedding $\mathcal{E}'$ of $G(\mathcal{C}_1,...,\mathcal{C}_n)$ yields a tuple of oriented curves $(\mathcal{C}'_1,...,\mathcal{C}'_n)$ (with $\mathcal{C}'_i = \mathcal{E}'(\omega_i)$) such that $W(\mathcal{C}'_1,...,\mathcal{C}'_n) = W(\mathcal{C}_1,...,\mathcal{C}_n)$. We now show that the t-graph $G(\mathcal{C}_1,...,\mathcal{C}_n)$ can be constructed from $W(\mathcal{C}_1,..., \mathcal{C}_n)$.

Definition 22: (a) A multiword $W \in DO_n(V)$ is *connected* if there is no bipartition (V_1,V_2) of V (with V_1,V_2 possibly empty) such that some components of W are in V_1^* and others in V_2^*. Hence (ε) is connected but $(\varepsilon,w_2,...,w_n)$ is not. If W has no empty component, this is equivalent to the property that the graph (V,R_W) is connected where $R_W := \{(a,b) \mid wabw'$ or bwa is a component of W for some words $w,w'\}$. If $L \subseteq V^*$, $u,v \in L$, we say that u and v are *connected via L* if there exist $z_1,...,z_p$ in L such that $u = z_1, v = z_p$ and for each i, z_i and z_{i+1} have at least one letter in common. Then W is connected if and only if any two distinct components are connected via its set of components. It is clear that $W(\mathcal{C}_1,...,\mathcal{C}_n)$ is connected if and only if $(\mathcal{C}_1,...,\mathcal{C}_n)$ is connected.

(b) A multiword $W = (w_1,w_2,...,w_n) \in DO_n(V)$ is *prime* if it is connected, different from (ε) and there is no bipartition (V_1,V_2) of V (with V_1,V_2 nonempty) and index i such that w_i is conjugate to a word in $V_1^+V_2^+$ and $w_j \in V_1^+ \cup V_2^+$ for every $j \neq i$, $(X^+ := X^* - \{\varepsilon\})$. For example, (aa) and $(abab)$ are prime whereas $aabb$ is not. The multiword $(abcd,abcd)$ is prime whereas $(abcd,ab,cd)$ is not. The prime multiword $(abcd,abef,cdfe)$ corresponds to the curves of Figure 1.3.

(c) *The t-graph associated with a connected multiword.*

Let $W = (w_1,w_2,...,w_n) \in DO_n(V)$ where each w_i is nonempty and written $v_{i,1}...v_{i,n_i}$ with $v_{i,j} \in V$. Each letter of V has occurrences in W. We define a t-graph ${}^tGra(W)$ as follows:

$$V_{{}^tGra(W)} := V,$$
$$E_{{}^tGra(W)} := \{(i,j) \mid 1 \leq j \leq n_i\},$$
$$vert_{{}^tGra(W)}((i,j)) := (v_{i,j},v_{i,j+1}) \text{ if } j < n_i \text{ and}$$
$$vert_{{}^tGra(W)}((i,n_i)) := (v_{i,n_i},v_{i,1}),$$
$$\tau_{{}^tGra(W)}((i,j)^+) := (i,j+1)^- \text{ if } j < n_i \text{ and}$$
$$\tau_{{}^tGra(W)}((i,n_i)^+) := (i,1)^-.$$

If $W' = (w'_1,...,w'_n)$ where $w'_i \equiv w_i$, then ${}^tGra(W')$ is v-isomorphic to ${}^tGra(W)$: some edges (i,j) of ${}^tGra(W)$ are just mapped to (i,j'). Here is an example. For $W = (ab,abcc)$, we have $v_{1,1} = v_{2,1} = a$, $v_{1,2} = v_{2,1} = b$, $v_{2,1} = v_{2,2} = c$. Then ${}^tGra(W)$ is the t-graph G with vertices a,b,c, edges $(1,1): a \to b$, $(1,2): b \to a$, $(2,1): a \to b$, $(2,2): b \to c$, $(2,3): c \to c$ and $(2,4): c \to a$. If $W' = (ab,ccab)$, then ${}^tGra(W')$ is v-isomorphic to ${}^tGra(W)$ by the bijection on edges that exchanges $(2,1)$ and $(2,3)$, and $(2,2)$ and $(2,4)$.

If W is not connected, ${}^tGra(W)$ is defined as above but may not be connected. It is a (2,2)-regular graph with transitions. (It may still be connected if W has empty components).

Lemma 23: A double occurrence multiword W without empty components is prime if and only if ${}^tGra(W)$ is atomic.

Proof: Let $W = (w_1,w_2,...,w_n) \in DO_n(V)$. We assume that it is not prime because of a bipartition (V_1,V_2). Without loss of generality, we assume that $w_1 = uv$ with

$u \in V_1^+$, $v \in V_2^+$, $w_2,...,w_n \in V_1^+ \cup V_2^+$. We let a be the first letter of u, b be its last letter, c be the first letter of v and d be its last letter. We may have $a = b, c = d$ and even $u = a = b$ or $v = c = d$. (An example is (ac, agg, chh) that is not a Gauss multiword). The only edges of $^tGra(W)$ between V_1 and V_2 are $e : b \to c$ and $f : d \to a$. It follows that $Graph(^tGra(W)) = H \boxplus_{e,f} K$ for some graphs H with vertex set V_1 and K with vertex set V_2. Hence, $^tGra(W)$ is not prime.

For the converse, we assume that $G = Graph(^tGra(W)) = H \boxplus_{e,f} K$ for some graphs H and K. Every closed walk of G that goes through e must go through f. The t-graph $^tGra(W)$ is covered by a union of pairwise edge-disjoint closed straight walk. One of them contains e and f and can be written $ePfQ$ where P and Q are paths, respectively in K and in H. The other closed straight walks are either in H or in K. It follows that the bipartition (V_H, V_K) of V_G shows that W is not prime. □

Theorem 24: (1) If $(\mathcal{C}_1,...,\mathcal{C}_n)$ is a connected n-tuple of curves with intersections, then $^tGra(W(\mathcal{C}_1,...,\mathcal{C}_n))$ is v-isomorphic to $G(\mathcal{C}_1,...,\mathcal{C}_n)$ in such a way that the i-th closed straight walk of $^tGra(W(\mathcal{C}_1,...,\mathcal{C}_n))$ is transformed into the i-th closed straight walk of $G(\mathcal{C}_1,...,\mathcal{C}_n)$.

(2) Let $W \in DO_n(V)$ be connected. It is a Gauss multiword if and only if it is (ε) or the t-graph $^tGra(W)$ is planar.

Proof: (1) We let $W(\mathcal{C}_1,...,\mathcal{C}_n) = (w_1,...,w_n)$, with each sequence w_i nonempty and written $(v_{i,1},...,v_{i,n_i})$. We first assume that each segment $s_{i,j}$ of $\mathcal{C}_i$ links $v_{i,j}$ to $v_{i,j+1}$ (and s_{i,n_i} links v_{i,n_i} to $v_{i,1}$). Then, $^tGra(W(\mathcal{C}_1,...,\mathcal{C}_n)) = G(\mathcal{C}_1,...,\mathcal{C}_n)$. Otherwise, a segment $s_{i,j}$ of $\mathcal{C}_i$ may link $v_{i,j'}$ to $v_{i,j'+1}$ and then, $^tGra(W(\mathcal{C}_1,...,\mathcal{C}_n))$ and $G(\mathcal{C}_1,...,\mathcal{C}_n)$ are v-isomorphic.

(2) Consider a plane embedding of the t-graph $^tGra(W)$ such that $W \neq (\varepsilon)$. Its closed straight walks define curves $\mathcal{C}_1,...,\mathcal{C}_n$ such that $W = W(\mathcal{C}_1,...,\mathcal{C}_n)$. Conversely, if W is a Gauss multiword defined as $W(\mathcal{C}_1,...,\mathcal{C}_n)$, then $\mathcal{C}_1,...,\mathcal{C}_n$ form a plane embedding of $^tGra(W)$. □

This theorem answers Question (1) of the introduction for multiwords. Its second assertion yields a natural characterization of Gauss multiwords and a linear-time recognition algorithm based on that of Proposition 14.

Corollary 25: Let $W \in DO_n(V) - \{(\varepsilon)\}$ be a connected Gauss multiword.

(1) The tuples of curves $(\mathcal{C}_1,...,\mathcal{C}_n)$ such that $W(\mathcal{C}_1,...,\mathcal{C}_n) = W$ are obtained from the plane embeddings $\mathcal{E}$ of the t-graph $^tGra(W)$ by defining $\mathcal{C}_i = \mathcal{E}(\omega_i)$ where ω_i is the closed straight walk of $^tGra(W)$ consisting of the edges (i, j).

(2) Two such tuples are homeomorphic if and only the corresponding embeddings of $^tGra(W)$ are homeomorphic.

Proof: (1) Clear from the previous constructions.

(2) If $\mathcal{E}$ and $\mathcal{E}'$ are two homeomorphic embeddings of $^tGra(W)$, the corresponding tuples of curves are homeomorphic.

The converse may look evident, but a careful proof is necessary. Assume that $\varphi : \mathbb{S} \to \mathbb{S}$ is a homeomorphism of the sphere $\mathbb{S}$ to itself that maps $(\mathcal{C}_1,...,\mathcal{C}_n)$

defined from $\mathcal{E}$ to $(\mathcal{C}'_1,...,\mathcal{C}'_n)$ defined from $\mathcal{E}'$. Since the intersection points are designated by the elements of V, we have $\varphi(\mathcal{E}(v)) = \mathcal{E}'(v)$ for each $v \in V$. Furthermore, $\varphi(\mathcal{E}(\omega_i)) = \mathcal{E}'(\omega_i)$ for each i. Hence, for each edge (i,j) we have: $\varphi(\mathcal{E}((i,j))) = \mathcal{E}'((i,j'))$ where j' may be different from j.

Let us examine this case : the edges (i,j) and (i,j') have same tail and head. Since $\mathcal{E}$ and $\mathcal{E}'$ preserve the transitions of ${}^tGra(W)$ and φ maps $\mathcal{E}(\omega_i)$ to $\mathcal{E}'(\omega_i)$, we also have $\varphi(\mathcal{E}((i,j+1))) = \mathcal{E}'((i,j'+1))$ and thus $(i,j+1)$ and $(i,j'+1)$ have also same tail and head. Since every $v \in V$ has at most two occurrences in each walk ω_i, this means that the tails of (i,j) and (i,j') are the two occurrences (in W) of some v, and the same is true for their heads, and for those of $(i,j+1)$ and $(i,j'+1)$. Hence, since W is connected, we have $n = 1$ and $W = (ww)$ for some sequence w, hence is a square.

Hence, if W is not a square, $\varphi(\mathcal{E}((i,j))) = \mathcal{E}'((i,j'))$ implies $j' = j$, and $\mathcal{E}$ and $\mathcal{E}'$ are homeomorphic embeddings of ${}^tGra(W)$. If W is a square, then ${}^tGra(W)$ is a planar t-atom (cf. Figure 1.2) and has a unique plane embedding up to homeomorphism by Theorem 15. So the result holds in both cases. □

The proof of Assertion (2) shows that, unless W is a square, there is a unique v-automorphism of ${}^tGra(W)$ that preserves each closed straight walk. If W is a square there are two such v-automorphisms. Note for comparison that if $W = (w,w)$, then there is a v-automorphism of ${}^tGra(W)$ that exchanges the two closed straight walks.

1.5.1.2 Unambiguous Gauss Multiwords

We say that a Gauss multiword W is *unambiguous* if any two systems of oriented curves that define it are homeomorphic. It is thus unambiguous if and only if it is (ε) or ${}^tGra(W)$ has a unique embedding (by Corollary 25), hence if and only if ${}^tGra(W)$ is an atom by Corollary 20. We now characterize this property in terms of W, which answers Question (2) for oriented curves.

Theorem 26: A connected Gauss multiword W is unambiguous if and only if it is prime or equal to (ε). If it is ambiguous, all tuples $(\mathcal{C}_1,...,\mathcal{C}_n)$ such that $W(\mathcal{C}_1,...,\mathcal{C}_n) = W$ can be determined from the planar maps of the t-graph ${}^tGra(W)$ by means of any term over t-atoms that defines it.

Proof: A connected Gauss multiword $W \neq (\varepsilon)$ is unambigous if and only if ${}^tGra(W)$ is atomic by Corollary 20. Furthermore, ${}^tGra(W)$ is atomic if and only if W is prime by Lemma 23.

If a connected Gauss multiword W is ambiguous, the tuples of curves $(\mathcal{C}_1,...,\mathcal{C}_n)$ such that $W(\mathcal{C}_1,...,\mathcal{C}_n) = W$ are in bijection with the planar maps of the t-graph ${}^tGra(W)$ by Corollary 25. Theorem 19 and Corollary 20 show that these maps can be determined from any term over t-atoms that defines this t-graph. □

If in a Gauss multiword $W = W(\mathcal{C}_1,...,\mathcal{C}_n)$ defined by $(w_1,...,w_n) \in (V^*)^n$ we replace some w_i by $\widetilde{w_i}$, then we obtain the Gauss multiword W' corresponding to the same tuple of curves where the orientation of $\mathcal{C}_i$ is reversed. Hence, the ambiguity of W does not depend on the orientations of the curves it represents. Furthermore, there is a bijective correspondence between the tuples of curves corresponding to W and to W'.

1.5.1.3 Small Components of Multiwords

We now consider tuples of curves such that at least one curve has exactly two intersection points with the others. The associated multiwords have several components and at least one of them has length 2. By "reducing" these multiwords, we will be able to extend Theorem 26 to the description by multiwords of tuples of nonoriented curves.

Definition 27: *Small components and reduction*
(1) Let $W = (w_1, ..., w_n)$ be a Gauss multiword. A component w_i of W of the form ab with $a \neq b$ is a *small component*. Some other component w_j must contain a and b. We define W' from W by removing w_i and the letters a and b from w_j; it is a Gauss multiword because, if W is associated with a tuple of curves $(\mathcal{C}_1, ..., \mathcal{C}_n)$, then W' is associated with the $(n-1)$-tuple obtained from $(\mathcal{C}_1, ..., \mathcal{C}_n)$ by removing $\mathcal{C}_i$. We write this $W \dashrightarrow W'$ and we call this transformation a *reduction step*. It is clear that W is connected if and only if W' is. (Since (ε) is defined as connected, this is true for $W = (ab, ab)$).

(2) A Gauss multiword W is *reducible* if it has small components; if furthermore it is connected, either it is of the form (ab, ab) or it can be written without loss of generality (by reordering its components if necessary):

$$W = (a_1b_1, a_2b_2, ..., a_pb_p, w_1, ..., w_m)$$

where $a_1, b_1, ..., a_p, b_p$, are pairwise distinct and each component $w_1, ..., w_m$ has length at least 4. Each pair a_i, b_i occurs in one of the components $w_1, ..., w_m$. We let $X = \{a_1, b_1, a_2, b_2, ..., a_p, b_p\}$ and we define $Red(W)$ as $(w'_1, ..., w'_m)$ where $w'_i := w_i \upharpoonright (V - X)$ (cf. Section 4.1.1 for notation). Clearly, $W \dashrightarrow^* Red(W)$. However, $Red(W)$ may have small components that we do not remove. For an example $Red((ab, cd, abef, cdef)) = (ef, ef)$. We may have $Red(W) = (\varepsilon)$. If W is connected then $Red(W)$ is so.

Reduction viewed in terms of graphs

Our objective is to show that if W is a connected Gauss multiword that reduces into W' by the removal of a small component ab, then there exists a planar t-atom D with vertices a and b such that (up to technical details):

$${}^tGra(W') = H \boxplus K \text{ and } {}^tGra(W) = H \boxplus D \boxplus K$$

for some unique t-graphs H and K that can be defined from W. (H and/or K may be empty).

Before starting the description of H and K, we define $\mathbf{D}_4(x,y;e,f,g,h)$ as the planar t-atom with vertices x and y, edges $e,h : x \to y$ and $f,g : y \to x$ and transitions $(e,g),(g,e),(f,h)$ and (h,f) (that yield the two closed straight walks (e,g) and (f,h)). Figure 1.11 (right part) shows $\mathbf{D}_4(a,b;g,f',g',e')$.

Let $W \in DO_n(V)$ be a connected Gauss multiword defined as $W(\mathcal{C}_1,...,\mathcal{C}_n)$ where $\mathcal{C}_1$ has two intersections, a and b, with one of the other curves. Without loss of generality (by reordering the tuple and taking a conjugate of w_2), we can assume that $W = (ab, aw'_2bw''_2, w_3,...,w_n)$. We have $W \to W' = (w'_2w''_2, w_3,...,w_n)$.

We first assume that w'_2 and w''_2 are not empty. The curve $\mathcal{C}_1$ separates the sphere into two connected open sets F_1 and F_2. (Our plane embeddings are in the sphere). Let F_1 be the one that contains the intersections represented by the letters of w'_2. Let V_1 and V_2 be the set of letters of the components w_i, $3 \le i \le n$, that are connected, respectively, to w'_2 and to w''_2 via the components of W. The curves whose words are connected to w'_2 (resp. to w''_2) are in F_1 (resp. in F_2), hence, (V_1,V_2) is a bipartition of $V - \{a,b\}$. (If W is not a Gauss multiword, then V_1 and V_2 may not be disjoint : take for instance $W = (ab, acbd, cdee)$).

We can reorder the components of W so that $w'_2, w_3,...,w_p \in V_1^*$ and $w''_2, w_{p+1},...,$ $w_n \in V_2^*$. Note that $W_1 = (w'_2, w_3,...,w_p)$ and $W_2 = (w''_2, w_{p+1},...,w_n)$ are connected Gauss multiwords, respectively $W(\mathcal{C}'_2,\mathcal{C}_3,...,\mathcal{C}_p)$ and $W(\mathcal{C}''_2,\mathcal{C}_{p+1},...,\mathcal{C}_n)$ for some $\mathcal{C}'_2$ and $\mathcal{C}''_2$.

We let $G := {}^tGra(W)$, $G' := {}^tGra(W')$ and $G_i := {}^tGra(W_i)$ for $i = 1,2$. We will denote in the same way the associated underlying graphs. In order to relate precisely G_1 and G_2 to G, we let c and c' be respectively the first and last letter of w'_2, and d and d' be similarly the first and last letter of w''_2. In G, we have the following edges that we name e,e',f,f',g and g' :

$e : c' \to b$, $e' : a \to c$, $f : d' \to a$, $f' : b \to d$, $g : a \to b$ and $g' : b \to a$,

cf. the left part of Figure 1.11.

Then G_1 is $G[V_1]$ augmented with an edge : $c' \to c$ that we name e, and, similarly, G_2 is $G[V_2]$ augmented with $f : d' \to d$. The transitions making them into t-graphs are as follows: $\tau_{G_1}(e^+) := \tau_G(e'^+)$ and $\tau_{G_2}(f^+) := \tau_G(f'^+)$; otherwise, $\tau_{G_1}(x)$ and $\tau_{G_2}(x)$ are $\tau_G(x)$. It is then clear that

$$G' = G_1 \boxplus_{e,f} G_2 \text{ and } G = G_1 \boxplus_{e,e'} \mathbf{D}_4(a,b;g,f',g',e') \boxplus_{f',f} G_2.$$

If $w'_2 \neq \varepsilon$ and $w''_2 = \varepsilon$, then $V_1 \neq \emptyset$, $V_2 = \emptyset$ and G_2 is undefined. In $G := {}^tGra(W)$, we have the following edges :

$e : c' \to b$, $e' : a \to c$, $f' : b \to a, g : a \to b$ and $g' : b \to a$,

with transitions$(e,f'),(f',e'),(g,g')$ and (g',g).

Here, $W' = (w'_2, w_3,...,w_n)$ and we have :

$$G := {}^tGra(W') \boxplus_{e,e'} \mathbf{D}_4(a,b;g,f',g',e').$$

If $w'_2 = w''_2 = \varepsilon$, then $V_1 = V_2 = \emptyset$, $W = (ab,ab), W' = (\varepsilon)$ and :

$${}^tGra(W) = \mathbf{D}_4(a,b;g,f',g',e')$$

for an appropriate naming of the edges. These constructions establish the following:

Proposition 28: Let $W = (ab, aw'_2bw''_2, w_3,...,w_n)$ be a connected Gauss multiword that reduces to $W' = (w'_2w''_2, w_3,...,w_n)$.

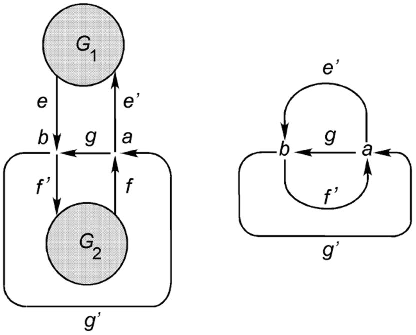


Fig. 1.11 Assertion (1) of Proposition 28

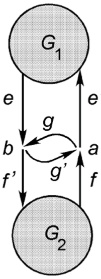


Fig. 1.12 The graph H of Remark 29(1)

(1) If $w_2', w_2'' \neq \varepsilon$, then ${}^tGra(W') = G_1 \boxplus_{e,f} G_2$ and
${}^tGra(W) = G_1 \boxplus_{e,e'} \mathbf{D}_4(a,b;g,f',g',e') \boxplus_{f',f} G_2$,
(2) if $w_2' \neq \varepsilon$ and $w_2'' = \varepsilon$, then
${}^tGra(W) = {}^tGra(W') \boxplus_{e,e'} \mathbf{D}_4(a,b;g,f',g',e')$,
(3) if $w_2' = w_2'' = \varepsilon$, then $n = 2$ and ${}^tGra(W) = \mathbf{D}_4(a,b;g,f',g',e')$,
where G_1, G_2, e, e' etc. are as in the construction. □

Remarks 29:(1) The order in which the edges occur in $\mathbf{D}_4(a,b;g,f',g',e')$ in Proposition 28 is important. Figure 1.12 shows for comparison the graph

$$H = G_1 \boxplus_{e,e'} \mathbf{D}_4(a,b;g,g',f',e') \boxplus_{f',f} G_2.$$

The corresponding Gauss multiword is $W = (baw_2', abw_2'', w_3, ..., w_n)$.

(2) The t-graph ${}^tGra(W')$ has exactly one atom less than ${}^tGra(W)$: this atom is $\mathbf{D}_4(a,b;g,f',g',e')$.

Construction of curves

A *face* of a plane embedding $\mathcal{E}$ of a graph G is a connected component of $\mathbb{S} - \mathcal{E}$. It is an open set and its (topological) border is the union of some line segments representing edges of G.

Lemma 30: A plane embedding of a graph $G = H \boxplus_{e,f} K \in \mathcal{SC}$ has two faces whose borders contain the line segments representing e and f.

Proof: Let $\mathcal{E}$ be a plane embedding of $G = H \boxplus_{e,f} K \in \mathcal{SC}$. The graphs $H - e$ and $K - f$ are connected, disjoint and $G - \{e,f\} = (H - e) \cup (K - f)$. The restriction of $\mathcal{E}$ to $G - \{e,f\}$, denoted by $\mathcal{E} \upharpoonright (G - \{e,f\})$ is the union of $\mathcal{E} \upharpoonright (H - e)$ and $\mathcal{E} \upharpoonright (K - f)$. Then, $\mathcal{E} \upharpoonright (H - e)$ is included in a face F_K of $\mathcal{E} \upharpoonright (K - f)$ whose border we denote by B. Similarly, $\mathcal{E} \upharpoonright (K - f)$ is included in a face F_H of $\mathcal{E} \upharpoonright (H - e)$ whose border we denote by B'. Then $\mathcal{E} \upharpoonright (G - \{e,f\})$ has a face $F = F_H \cap F_K$ whose border is $B \cup B'$. Since $\mathcal{E}(e) \cap F \neq \emptyset$ and $\mathcal{E}(f) \cap F \neq \emptyset$, each of e and f has one end vertex in B and the other in B' (otherwise $\mathcal{E}$ is not plane). It follows that $\mathcal{E}(e)$ and $\mathcal{E}(f)$ divide F into two faces of $\mathcal{E}$ whose borders contain both of them. □

Proposition 31: Let W be a connected Gauss multiword that reduces to W' by deletion of one small component, and $\mathcal{S}'$ be a tuple of oriented curves such that $W' = W(\mathcal{S}')$.

(1) If $W' = (\varepsilon)$, there is a unique way, up to homeomorphism, to extend $\mathcal{S}'$ into a pair $\mathcal{S}$ such that $W(\mathcal{S}) = W$.

(2) If $W' \neq (\varepsilon)$, there are exactly two ways, up to homeomorphism, to extend $\mathcal{S}'$ into a tuple $\mathcal{S}$ such that $W(\mathcal{S}) = W$.

Proof: (1) In this case, $W = (ab, ab)$, $\mathcal{S}'$ is one circle (without self-intersection), and there is a unique way to extend $\mathcal{S}'$ into $\mathcal{S}$ such that $W(\mathcal{S}) = (ab, ab)$ because ${}^t Gra(W)$ is a t-atom.

(2) Assume first that ${}^t Gra(W') = G_1 \boxplus_{e,f} G_2$ (cf. Proposition 28(1)) and consider its embedding $\mathcal{E}'$ defined by $\mathcal{S}'$. One of its curves contains $\mathcal{E}'(e)$ and $\mathcal{E}'(f)$. Note that the edges e and f are the unique ones in ${}^t Gra(W')$ that link respectively V_1 to V_2 (i.e., a vertex of V_1 to a vertex of V_2) and V_2 to V_1 (so that $\mathcal{E}'(e)$ and $\mathcal{E}'(f)$ are defined in a unique way as segments of a line). The curve to be added to $\mathcal{S}'$ must cross these two segments and no other one. By Lemma 30, $\mathcal{E}'$ has two faces F and F' whose borders contain these segments. Let us subdivide these segments by adding two points x_e and x_f representing the two vertices of ${}^t Gra(W)$ not in ${}^t Gra(W')$. The t-graph ${}^t Gra(W)$ has edges g: $x_f \to x_e$ and g': $x_e \to x_f$ with transitions (g, g') and (g', g). We make $\mathcal{E}'$ into a plane embedding $\mathcal{E}$ of ${}^t Gra(W)$ by adding line segments in exactly two possible ways (up to homeomorphism) : we can draw g in F and g' in F' or vice versa. Figure 1.11 shows one possibilty. The other one is obtained by exchanging g and g'.

If now $V_2 = \emptyset$, then $G = {}^t Gra(W') \boxplus_{e,e'} \mathbf{D}_4(a,b; g, f', g', e')$ (Proposition 28(2)). Consider the embedding $\mathcal{E}'$ of ${}^t Gra(W')$ defined by $\mathcal{S}'$. The edge e of ${}^t Gra(W')$ links c to c'. These letters are respectively the first and last ones of w_2', cf. Lemma 30. If W' is not a square, there is a unique segment of some line that is $\mathcal{E}'(e)$, see Corollary 25(2). The new line must intersect it twice, at a and b, in the order c, b, a, c'. Again there are exactly two ways, up to homeomorphism to define it.

If W' is a square (ww), there are two line segments representing edges from c to c'. This seems to yield four ways to extend $\mathcal{S}'$ into $\mathcal{S}$, but actually we get only two, up to homeomorphism, because these two segments correspond to two edges related by a v-automorphism of ${}^{t}Gra(W')$. □

Anticipating on the sequel, we observe that in Case (2), there are two ways to extend $\mathcal{S}'$ into $\mathcal{S}$ because the curves are oriented. If we consider nonoriented curves, we get a unique extension, up to homeomorphism.

1.5.2 Curves without Orientation

We now consider curves $\mathcal{C}_1,...,\mathcal{C}_n$ having no particular orientation. The components of $W(\mathcal{C}_1,...,\mathcal{C}_n)$ are thus reversal-circular: we need not distinguish a sequence from its reversal. For the three nonoriented curves of Figure 1.3, we get $W(\mathcal{C}_1,\mathcal{C}_2,\mathcal{C}_3)$ defined by $(abcd,aecf,bfde)$ as well as by $(abcd,fcea,bfde)$ or $(cdab,afce,debf)$.

Consider again the oriented curves of Figure 1.4. The corresponding multiword $W = (abcd,bc,ad)$ is ambiguous as already observed. However, if we omit the orientations, the two triples of curves are homeomorphic and W is not ambiguous for describing nonoriented curves. The difference between the oriented and nonoriented cases is due to the presence of curves with two intersections because reversing the orientation of such a curve that has two crossings with the others changes the corresponding map in $\mathcal{PM}^t_{2,2}$ but not its undirected version.

If the curves of a tuple $(\mathcal{C}_1,...,\mathcal{C}_n)$ are not oriented, we define $G(\mathcal{C}_1,...,\mathcal{C}_n)$ as $Und(G(\mathcal{C}'_1,...,\mathcal{C}'_n))$ where $\mathcal{C}'_i$ is any orientation of $\mathcal{C}_i$.

Let $(\mathcal{C}_1,...,\mathcal{C}_n)$ and $(\mathcal{C}'_1,...,\mathcal{C}'_n)$ be two tuples of oriented curves such that $W(\mathcal{C}_1,..., \mathcal{C}_n) \asymp W(\mathcal{C}'_1,...,\mathcal{C}'_n)$. (The equivalence $\asymp$ combines conjugacy and reversal.) If they are homeomorphic, then, the tuples of corresponding nonoriented curves are also homeomorphic. Hence if $W(\mathcal{C}_1,...,\mathcal{C}_n) = W$ is ambiguous for nonoriented curves, it is also for oriented ones. The converse is not always true. However, it is for a multiword W provided the nonoriented curves of any tuple that yields it can be equipped with an orientation depending only on W. This fact motivates the following definition.

Definition 32:*Orientable components of multiwords*
(1) A word $w \in V^*$ is *orientable* if $Conj(w) \neq Conj(\widetilde{w})$ ($Conj(w)$ is the set of conjugate words of w, cf. Section 4.1.1). It is easy to see that, if this condition is true for w, then it is also true for every sequence $\asymp$-equivalent to w. A *palindrome* is a sequence w such that $w = \widetilde{w}$. A sequence $\asymp$-equivalent to a palindrome is not orientable. Neither is ab, where $a,b \in V$.

(2) Let $\leq$ be an arbitrary but fixed linear order on V. From it we get a lexicographic linear order on the set of finite subsets of V^*, that we also denote by $\leq$. For every orientable $w \in V^*$, we define:

(2.1) $\mu(w) = w$ if $Conj(w) < Conj(\widetilde{w})$, and

(2.2) $\mu(w) = \widetilde{w}$ if $Conj(\widetilde{w}) < Conj(w)$.

If all components of $W = (w_1, ..., w_n) \in (V^*)^n$ are orientable, we say that W is *orientable* and we define $\mu(W) := (\mu(w_1), ..., \mu(w_n))$.

Lemma 33: A component of a Gauss multiword is orientable except if it is (conjugate to) a palindrome or has length 2.

Proof: In the following proof, a, b, c denote elements of V and u, v, w, w_1 etc. denote elements of V^*.

Fact: A Gauss multiword has no component of the form $aub\widetilde{u}$ or $aua\widetilde{u}$ with $u \neq \varepsilon$. (Otherwise, by constructing the closed straight walk associated with $aub\widetilde{u}$ or $aua\widetilde{u}$ we get a contradiction with planarity). ■

Let w be a nonempty component of a Gauss multiword such that $Conj(w) = Conj(\widetilde{w})$. We prove that it is of the form ab, with $a \neq b$ or is conjugate to a palindrome. We distinguish several cases. Recall that w can be replaced by any conjugate sequence, and that its length is even.

Case 1: w has at least two different letters a, b that have only one occurrence. Then $w = aubv$ and $w \in Conj(\widetilde{w}) = Conj(\widetilde{v}b\widetilde{u}a)$. We must have $w = a\widetilde{v}b\widetilde{u}$, hence $v = \widetilde{u}$. But this contradicts the initially observed fact, except if $u = v = \varepsilon$, which gives $w = ab$.

In the next two cases, all letters in w have two occurrences in this word.

Case 2: $w = uv$ where u and v are sequences over two disjoint nonempty alphabets (or w is conjugate to such a sequence). Then $w \in Conj(\widetilde{w}) = Conj(\widetilde{v}\widetilde{u})$ implies $w = \widetilde{u}\widetilde{v}$, hence $u = \widetilde{u}$ and $v = \widetilde{v}$ are two palindromes. If they have both even length, then $w = u_1\widetilde{u_1}v_1\widetilde{v_1}$. It is conjugate to the palindrome $\widetilde{u_1}v_1\widetilde{v_1}u_1$. If they have odd length, then $w = u_1a\widetilde{u_1}v_1b\widetilde{v_1}$. But a and b belong to disjoint alphabets, hence $a \neq b$ and a has a single occurrence. This case cannot happen.

Hence, the first letter of w differs from the last one. The only remaining possibilities are :

Case 3: $w = auavbxb$ and $w = aubvaxb$.

In the first case, we have $w \in Conj(b\widetilde{x}b\widetilde{v}a\widetilde{u}a)$. We must have $auavbxb = a\widetilde{u}ab\widetilde{x}b\widetilde{v}$, hence, $v = \varepsilon$ and u and x are palindromes. If u and x have even length, then w is conjugate to a palindrome (cf. Case 2). If they have odd length, we have $w = au_1c\widetilde{u_1}abx_1c\widetilde{x_1}b$, hence w is conjugate to $c\widetilde{u_1}abx_1c\widetilde{x_1}bau_1$ which contradicts the initial fact.

Very similar arguments using the initial fact eliminate the case where $w = aubvaxb$. □

If $(\mathcal{C}_1, ..., \mathcal{C}_n)$ is an n-tuple of oriented curves, we denote by $NO(\mathcal{C}_1, ..., \mathcal{C}_n)$ the tuple of nonoriented curves obtained by forgetting the orientations.

Proposition 34: Let W be a connected Gauss multiword that is orientable or has a single component.

(1) W is unambiguous if and only if it is for describing nonoriented curves.

(2) If W is ambiguous, there is a bijection that preserves homeomorphisms between the n-tuples of oriented and of nonoriented curves described by W. Both sets of n-tuples can be described from any term over atoms that defines ${}^{t}Gra(W)$.

Proof: We first assume that $W = (w_1, ..., w_n)$ is orientable.

Claim: There exists a bijection ν between the n-tuples $(\mathcal{C}_1, ..., \mathcal{C}_n)$ of nonoriented curves described by W and the n-tuples of oriented ones described by $\mu(W)$ such that $(\mathcal{C}_1, ..., \mathcal{C}_n) = NO(\nu(\mathcal{C}_1, ..., \mathcal{C}_n))$. Furthermore, if $(\mathcal{D}_1, ..., \mathcal{D}_n)$ is homeomorphic to $(\mathcal{C}_1, ..., \mathcal{C}_n)$, then $\nu(\mathcal{D}_1, ..., \mathcal{D}_n)$ is homeomorphic to $\nu(\mathcal{C}_1, ..., \mathcal{C}_n)$ by the same homeomorphism of the sphere to itself.

Proof of the claim: Let W and $(\mathcal{C}_1, ..., \mathcal{C}_n)$ be as stated. Then $(\mathcal{C}_1, ..., \mathcal{C}_n) = NO(\mathcal{C}''_1, ..., \mathcal{C}''_n)$ where $\mathcal{C}''_1, ..., \mathcal{C}''_n$ are oriented curves such that $W = W(\mathcal{C}''_1, ..., \mathcal{C}''_n)$. Let I be the set of indices i such that $\mu(w_i) = \widetilde{w_i}$ (i.e., Case (2.2) of Definition 32 applies). Then we define $\nu(\mathcal{C}_1, ..., \mathcal{C}_n)$ as the n-tuple $(\mathcal{C}''_1, ..., \mathcal{C}''_n)$ except that we reverse the orientation of $\mathcal{C}''_i$ whenever $i \in I$.

It is then clear that $NO(\nu(\mathcal{C}_1, ..., \mathcal{C}_n)) = (\mathcal{C}_1, ..., \mathcal{C}_n)$. The mapping ν is a bijection because the orientations of the curves in $\nu(\mathcal{C}_1, ..., \mathcal{C}_n)$ are determined from W. The last assertion is also clear. ■

This claim and the remark after Theorem 26 show that there is a bijection between the homeomorphism classes of the n-tuples of nonoriented and of oriented curves described by W, which proves (1) and (2) for orientable multiwords.

It remains to consider the case where $n = 1$ and W is not orientable. By Lemma 33, this means that $W = (w)$ and w is a palindrome. The results hold if $W = (\varepsilon)$. Otherwise, let M be a planar map of ${}^{t}Gra(W)$. The map M' obtained from M by reversing all edge directions is a planar map of ${}^{t}Gra((\widetilde{w}))$ that is v-isomorphic to M^{-1}. Hence, up to homeomorphism, the same oriented and nonoriented curves are described by w and by $\widetilde{w}$ which proves (1) and (2). (Note that ${}^{t}Gra(W)$ is prime, hence that w is unambiguous if and only if $w = aa$.) □

With the hypotheses of this proposition, if p is the number of atoms of the t-graph ${}^{t}Gra(W)$, then the number of pairwise nonhomeomorphic tuples of oriented (resp. nonoriented) curves described by W is 2^{p-1} by Corollary 20. Another consequence of this proposition is that a Gauss word (representing a single curve) is unambiguous for describing nonoriented curves if and only if it is prime. This fact is proved in a completely different way in [3].

We now use the reduction of Definition 27 to handle the cases not covered by Proposition 34.

Lemma 35: Let W be a connected Gauss multiword W that has several components and is not orientable. It is reducible. There is a bijection θ between the homeomorphism classes of the tuples of nonoriented curves described by W and those of oriented curves described by $Red(W)$.

Proof: Let W be as stated. None of its components is a palindrome because it is connected and has several components. It has small components and can be written (ab,ab) or $(a_1b_1,a_2b_2,...,a_pb_p,w_1,..., \ w_m) \in (V^*)^n$ as in Definition 27. In the first case, $Red(W) = (\varepsilon)$ and W describes a circle, so the result holds. Otherwise, the sequences $\mu(w_1),...,\mu(w_m)$ are defined because $w_1,...,w_m$ have length at least 4 and are not palindromes. We let $w'_i := \mu(w_i) \restriction (V - \{a_1,...,a_p,b_1,...,b_p\}$. Hence, $Red(W) \asymp (w'_1,...,w'_m)$.

For every n-tuple of nonoriented curves $(\mathcal{C}_1,...,\mathcal{C}_n)$ such that $W(\mathcal{C}_1,...,\mathcal{C}_n) \asymp W$, we let $(\mathcal{C}'_1,...,\mathcal{C}'_n)$ be an n-tuple of oriented curves such that $NO(\mathcal{C}'_1,...,\mathcal{C}'_n) = (\mathcal{C}_1,...,\mathcal{C}_n)$ and $w(\mathcal{C}'_i) = \mu(w_{i-p})$ for each $i = p+1,...,n$. Hence, the curves $\mathcal{C}'_{p+1},...,\mathcal{C}'_n$ have a canonical orientation based on the orientability of their associated sequences. For the curves $\mathcal{C}'_1,...,\mathcal{C}'_p$, we take any orientation. It is then clear that $W(\mathcal{C}'_{p+1},...,\mathcal{C}'_n) = (w'_1,...,w'_m)$. We define $\theta(\mathcal{C}_1,...,\mathcal{C}_n)$ as $(\mathcal{C}'_{p+1},...,\mathcal{C}'_n)$.

Every m-tuple of oriented curves $(\mathcal{E}_1,...,\mathcal{E}_m)$ such that $W(\mathcal{E}_1,...,\mathcal{E}_m) = (w'_1,..., w'_m)$ is $\theta(\mathcal{C}_1,...,\mathcal{C}_n)$ for some $\mathcal{C}_1,...,\mathcal{C}_n$ such that $W(\mathcal{C}_1,...,\mathcal{C}_n) \asymp W$.

Let $(\mathcal{D}_1,...,\mathcal{D}_n)$ be an n-tuple of nonoriented curves such that $W(\mathcal{D}_1,...,\mathcal{D}_n) \asymp W$. If it is homeomorphic to $(\mathcal{C}_1,...,\mathcal{C}_n)$, then $\theta(\mathcal{D}_1,...,\mathcal{D}_n)$ is homeomorphic to $\theta(\mathcal{C}_1,...,\mathcal{C}_n)$. This is so because the n-tuple $(\mathcal{D}'_1,...,\mathcal{D}'_n)$ associated with $(\mathcal{D}_1,...,\mathcal{D}_n)$ as $(\mathcal{C}'_1,...,\mathcal{C}'_n)$ is with $(\mathcal{C}_1,...,\mathcal{C}_n)$ is homeomorphic to $(\mathcal{C}'_1,...,\mathcal{C}'_n)$ up to the orientations of the curves $\mathcal{D}'_1,...,\mathcal{D}'_p$ that have two intersections with the others.

Note that $NO(\theta(\mathcal{C}_1,...,\mathcal{C}_n)) = (\mathcal{C}_{p+1},...,\mathcal{C}_n)$. Proposition 31 shows that, up to homeomorphism, there is a unique way to extend this $(n-p)$-tuple by nonoriented curves $\mathcal{C}_1,...,\mathcal{C}_p$ in such a way that $W(\mathcal{C}_1,...,\mathcal{C}_n) \asymp W$. This observation shows that if $\theta(\mathcal{D}_1,...,\mathcal{D}_n)$ is homeomorphic to $\theta(\mathcal{C}_1,...,\mathcal{C}_n)$, then $(\mathcal{D}_1,...,\mathcal{D}_n)$ is homeomorphic to $(\mathcal{C}_1,...,\mathcal{C}_n)$. □

Figure 1.13 shows an 11-tuple of nonoriented curves. The small components of the associated multiword W correspond to curves 1 to 6. The reduced multiword $Red(W)$ has small components corresponding to curves 7 and 9.

The following theorem answers Questions (2) and (3) of the introduction for the description of nonoriented curves.

Theorem 36: Let W be a connected Gauss multiword. In the following statements, ambiguity refers to the description of nonoriented curves.

(1) If W has a single component, it is unambigous if and only if it is (ε) or ${}^tGra(W)$ is prime.

(2) If it has several components, it is unambigous if and only if $Red(W) = (\varepsilon)$ or ${}^tGra(Red(W))$ is prime.

(3) In both cases, all corresponding nonoriented curves can be determined from a term over t-atoms that defines ${}^tGra(W)$ in Case (1) or ${}^tGra(Red(W))$ in Case (2).

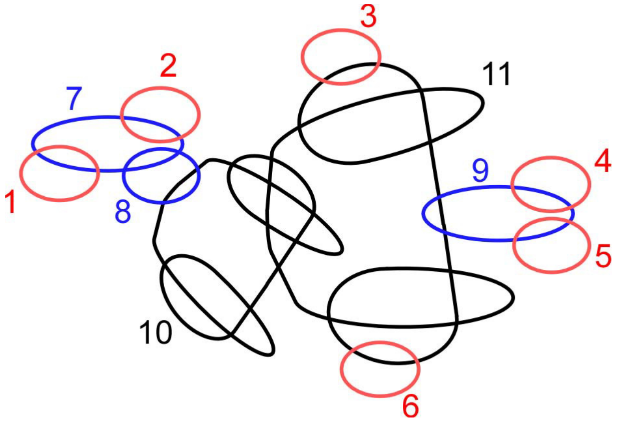


Fig. 1.13 An 11-tuple of nonoriented curves

Proof: (1) holds by Proposition 34(1) and Theorem 26.

(2) and (3) hold by Proposition 34, Lemma 35,and Theorem 26. □

The following algorithm analyses a double occurrence multiword W intended to describe nonoriented curves.

Algorithm 37: *Analysis of a double occurrence multiword.*

1. We first check if it is connected. If it is not, we treat separately each of its connected submultiwords.

2. Unless $W = (\varepsilon)$, we construct the t-graph ${}^tGra(W)$ and we check its planarity.

3. If it is planar, then W is a connected Gauss multiword, and we reduce it into $W' := Red(W)$ (cf. Definition 27; $Red(W) = W$ if W has a single component) by keeping track of the reduction steps.

4. We construct the (planar) t-graph ${}^tGra(W')$ and a term t over atoms that defines it (we use Theorem 13); we let p be the number of atoms used by t.

5. By using Proposition 14, we determine a planar t-map for each t-atom occurring in t.

6. By using Corollary 20, we compute the 2^{p-1} (terms defining the) planar t-maps that represent the different plane embeddings of ${}^tGra(W')$, hence, the different tuples of curves that yield W'.

7. Since we have recorded the sequence of reductions done at Step 3, and by using this sequence backwards, we can compute the 2^{p-1} planar t-maps of ${}^tGra(W)$, hence, the different tuples of curves (extending those we have by step 6) described by W.

Its correctness follows from Theorem 36 and the quoted results.

1.6 Some Open Questions

Knots and knot diagrams

The reader will find the definitions and basic facts in the books [1, 2, 10]. Let us consider oriented links and knots. A (connected) link diagram is nothing but a map in $\mathcal{M}^t_{2,2}$ whose vertices are labelled by **over** or **under**, to indicate the type of crossing. A knot diagram is such a map that has a unique straight walk. We let k map a diagram to the corresponding link or knot. Knots can be composed by a binary (multivalued) operation denoted in [1] by # such that for every two knot diagrams D and D', every edge e of D and every edge f of D', we have $k(D \boxplus_{e,f} D') = k(D)\#k(D')$. It follows that the diagrams of prime knots are atoms. We leave as a research topic to investigate the relationships between the unique factorization of a knot and the atomic decompositions of its diagrams.

Planar t-graphs

A t-graph G contains more information than the underlying graph $Graph(G)$ and less information than a map M in $\mathcal{M}^t_{2,2}$ such that $Graph^t(M) = G$. Planar graphs and planar maps are characterized by finitely many forbidden minors ([7, 14] for graphs, [4, 6] for maps). Our question is:

Are planar t-graphs characterized by finitely many forbidden substructures of some kind?

We have given a natural characterization of Gauss multiwords (Theorem 24(2)), hence also, of Gauss words. However, Gauss words also a characterization in terms of forbiden subwords [12].

Can it be extended to Gauss multiwords?

Many extensions remain to be investigated. First, intersections of curves on other surfaces than the plane (see [11]). Our characterization of Gauss multiwords in terms of the planarity of an associated t-graph extends easily to other surfaces. Since the embeddability of a graph in an arbitrary given surface can be checked in linear time by an algorithm that produces embeddings ([13]), the algorithm of Proposition 14 can be turned into a linear-time algorithm for checking if a double occurrence multiword is associated with curves in a given surface.

Can one extend to other surfaces the characterization (resulting from Corollary 20 and its consequences) of all tuples of curves in a given surface that correspond to a given double occurrence multiword ?

Some other questions:

Can the results of this article be generalized to the description of curves with multiple intersections?

Gauss multiwords are defined from curves that cross, themselves or pairwise. One could also consider curves that have *touching points* i.e., that can share a vertex without crossing at it. This notion is used in [8]. Special labels in double occurrence multiwords could encode touching points.

What are the associated multiwords and when are they unambiguous ?
What are the tuples of curves with crossing and touching points
associated with a given multiword ?

Acknowledgements. I thank E. Gioan and the late M. Las Vergnas for many helpful comments.

References

1. Adams, C.: The knot book. AMS (2004)
2. Bollobas, B.: Modern graph theory. Springer (2001)
3. Chaves, N., Weber, C.: Plombages de rubans et problème des mots de Gauss. Exp. Math. 12, 53–77 (1994)
4. Courcelle, B.: The monadic second-order logic of graphs XII: Planar graphs and planar maps. Theoret. Comput. Sci. 237, 1–32 (2000)
5. Courcelle, B.: The atomic decomposition of strongly connected graphs, Research report (June 2013), `http://hal.archives-ouvertes.fr/hal-00875661`
6. Courcelle, B., Dussaux, V.: Map genus, forbidden maps and monadic second-order logic. The Electronic Journal of Combinatorics 9(1), 40 (2002), `http://www.combinatorics.org/Volume_9/Abstracts/v9i1r40.html`
7. Diestel, R.: Graph theory, 4th edn. Springer (2010), `http://diestel-graph-theory.com`
8. de Fraysseix, H., Ossona de Mendez, P.: On a characterization of Gauss codes. Discrete Comput. Geom. 22, 287–295 (1999)
9. Galil, Z., Italiano, G.: Maintaining the 3-edge-connected components of a graph on-line. SIAM J. Comput. 22, 11–28 (1993)
10. Godsil, C., Royle, G.: Algebraic graph theory. Springer (2001)
11. Lins, S., Richter, B., Shank, H.: The Gauss code problem off the plane. Aequationes Mathematicae 33, 81–95 (1987)
12. Lovasz, L., Marx, M.: A forbidden substructure characterization of Gauss codes. Bulletin of the AMS 82, 121–122 (1976)
13. Mohar, B.: A linear time algorithm for embedding graphs in an arbitrary surface. SIAM J. Discrete Math. 12, 6–26 (1999)
14. Mohar, B., Thomassen, C.: Graphs on surfaces. The Johns Hopkins University Press (2001)
15. Naor, D., Gusfield, D., Martel, C.: A fast algorithm for optimally increasing the edge connectivity. SIAM J. Comput. 26, 1139–1165 (1997)
16. Rosenstiehl, P.: Solution algébrique du problème Gauss sur la permutation des points d'intersection d'une ou plusieurs courbes fermées du plan. C. R. Acad. Sc. Paris, Sér. A 283, 551–553 (1976)
17. Tsin, Y.H.: A simple 3-edge-connected component algorithm. Theory Comput. Syst. 40, 125–142 (2007)

Appendix

Index to some definitions.

Isomorphisms

Isomorphism of graphs : notation $G \cong H$, Section 1.1.1.

v-isomorphism of graphs (identity on vertices) : Section 1.1.1.

Automorphisms and v-automorphisms : Section 1.1.1.

v_isomorphisms of t-graphs : Section 1.1.2, Definition 2.

Homeomorphisms

Homeomorphisms and equivalence of maps : Section 1.2, Definition 4.

Equivalences

Conjugacy of sequences : notation $\equiv$, Section 4.1.1.

Conjugacy combined with reversal : notation $\asymp$, Section 4.1.1.

Equivalent maps ($M = N$ or $M = N^{-1}$) : Section 1.2, Definition 4.

t-equivalent maps (same associated t-graph, it is implied by equivalence) : notation $\sim^t$, Section 1.2, Definition 5.

Equivalent multiwords (by reversal of some components) : notation $W \asymp W'$, Section 4.1.1.

Equivalent terms (same value) : Section 2.2, Definition 11.

Chapter 2
Logical Theory of the Additive Monoid of Subsets of Natural Integers

Christian Choffrut and Serge Grigorieff

Abstract. We consider the logical theory of the monoid of subsets of $\mathbb{N}$ endowed solely with addition lifted to sets: no other set theoretical predicate or function, no constant (contrarily to previous work by J̇ez and Okhotin cited below). We prove that the class of true Σ_5 formulas is undecidable and that the whole theory is recursively isomorphic to second-order arithmetic. Also, each ultimately periodic set A (viewed as a predicate $X = A$) is Π_4 definable and their collection is Σ_6. Though these undecidability results are not surprising, they involve technical difficulties witnessed by the following facts: 1) no elementary predicate or operation on sets (inclusion, union, intersection, complementation, adjunction of 0) is definable, 2) The class of subsemigroups is not definable though that of submonoids is easily definable. To get our results, we code integers by a Π_3 definable class of submonoids and arithmetic operations on $\mathbb{N}$ by Δ_5 operations on this class.

2.1 Introduction

The object of this paper could hardly be more elementary since we are concerned with the class of subsets of nonnegative integers equipped with addition as unique operation. Presburger studied in 1929 the first-order logic of integers with addition and showed that this logic admits quantifier elimination on the language enriched with the order relation and all arithmetic congruences. Consequently, the theory is decidable and it was proved in 1974 [3] by Michael J. Fischer and Michael O. Rabin that its time complexity is upper bounded by a double exponential. In the sixties, the class of relations defined by the logic received a simple algebraic characterization by Seymour Ginsburg as the semilinear subsets of integers, [5]. It can thus be reasonably said that this logic is well-understood.

Christian Choffrut · Serge Grigorieff
LIAFA, CNRS and Université Paris 7 Denis Diderot, France
e-mail: `seg@liafa.univ-paris-diderot.fr`

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_2

Our purpose is still a first-order theory but the domain is the power set of $\mathbb{N}$ with set addition and equality, formally $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$. At the beginning of our investigation we came up every day with different properties. Some could be considered as the source of inspiration for exercises in an introductory course in logic or as entertaining mathematical recreation. Others played a more important role and are kept in this paper, but all had a low complexity in the arithmetical hierarchy, i.e., they were expressible with very few quantifier alternations. However, we could not build on them to get a consistent and general view of the problem. E.g., we were not even able to answer the question whether or not the property for a subset to be recognizable by a finite automaton is expressible. The final picture to the contrary is that the expressiveness of the theory is extremely powerful but, at least the way we did it, this was obtained by working out predicates of higher complexity.

It should not be surprising that the submonoids of $\mathbb{N}$ play a crucial role since a nonempty subset X is a submonoid if and only if it satisfies the condition $X + X = X$. Submonoids of $\mathbb{N}$ have a deceiving simplicity. Contrarily to submonoids of nonunary free monoids, they are finitely generated. They are related to an intriguing and well-celebrated problem attributed to Frobenius which asks the following. Say a submonoid is *numerical* if it is generated by a finite subset of integers with greatest common divisor equal to 1. These submonoids are known to be cofinite in $\mathbb{N}$ but what precisely is the largest integer not in the submonoid? There is a rich literature on the topic and many conferences are dedicated to the classification of these monoids [2,4,9,11], nonetheless the problem seems to be far from solved and we could not find any result that would help us in our investigation.

We now turn to a quick presentation of our work. Section 2.2 gathers all basic material of algebraic or logical type used in the sequel and is essentially meant for the reader unfamiliar with the domain. Section 2.3 establishes the main properties of our paper which can be summarized as saying that under certain restrictions which cannot be relaxed, membership of an element to a subset and subset inclusion can be expressed via special (and simple) submonoids. Based on these results, Section 2.4 shows that the theory is highly undecidable by interpreting the second-order theory of arithmetic in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$. Actually, the Σ_5 fragment is already undecidable, see Theorem 7. At this point of the article the only useful subsets all contain the integer 0. Section 2.5 investigates to what extent other classes of subsets are expressible. We show in particular that we cannot express the fact that a subset is obtained from another by just adding 0. The same is true of the simplest set theoretical predicates (inclusion, union, intersection, complementation) and of the class of subsemigroups of $\mathbb{N}$. Nevertheless, this leaves the problem of trying to extend the classes of subsets expressible in the logic which is done in section 2.6. In particular, we show that a singleton class $\{A\}$ is definable with set addition if and only if it is definable in second-order arithmetic. An inventory of typical predicates expressible in the theory to be found in Section 2.7 serves an illustrative purpose among which the class of regular subsets of $\mathbb{N}$ which is Σ_6 definable.

It is worthwhile mentioning the works of Jez and Okhotin since they can be interpreted as studying the Diophantine theory of the current structure enriched with all ultimately periodic subsets of $\mathbb{N}$ (as set constants). In [6] they show that there exists an encoding of the subsets of $\mathbb{N}$ under which each recursive subset of $\mathbb{N}$ is the encoding of the unique solution of some system of equations involving the operation of sum of subsets and the regular subsets as unique constants. Furthermore they prove the satisfiability of this theory to be Π_1^0-complete. Because ultimately periodic constants are Π_4 definable, their undecidable result is in accordance with ours and leaves the open question of finding the minimum undecidable fragment.

2.2 Preliminaries

In this section we recall classical and introduce elementary properties of two types: algebraic and logic.

2.2.1 Submonoids

Given a non negative integer n and two subsets $X, Y \subseteq \mathbb{N}$ we define

$$nX = \{nx \mid x \in X\} \tag{2.1}$$

$$X+Y = \{x+y \mid x \in X, y \in Y\} \tag{2.2}$$

Observe that $2X \neq X+X$.

Definition 1. A subset $X \subseteq \mathbb{N}$ is a *subsemigroup* if it is closed under addition, i.e., $X+X \subseteq X$. A subsemigroup is a *submonoid* if it is nonempty and contains 0. Equivalently a submonoid is a nonempty subset satisfying the condition $X+X=X$.

The *submonoid generated* by Y, denoted Y^*, is the minimum submonoid containing Y, i.e. containing every finite sum of elements of Y

$$Y^* = \{0\} \cup \bigcup_{n \geq 1} \overbrace{Y+\cdots+Y}^{n \text{ times}}$$

The subset Y is a *generating subset* of Y^*.

We are concerned with the first-order theory of $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$. It is not surprising that the submonoids of $\mathbb{N}$ play a central role since a set X containing 0 is a submonoid if and only if $X = X+X$ holds. Most of the following is folklore and does not contain anything new. For the sake of completeness we recall it with some detail. We start with two trivial observations the proofs of which are omitted.

Proposition 1. *A set $X \subseteq \mathbb{N}$ such that $0 \in X$ is a monoid if and only if $X \setminus \{0\}$ is a subsemigroup.*

Proposition 2. *If X and Y are two submonoids, so is $X+Y$.*

A remarkable property of $\mathbb{N}$ is that its submonoids are finitely generated. This can be stated more precisely.

Proposition 3. *1. Every submonoid X of $\mathbb{N}$ is finitely generated and has a minimum generating set $G(X)$ which is equal to*

$$G(X) = S \setminus (S+S) \quad \textit{where } S = X \setminus \{0\} \tag{2.3}$$

The set $G(X)$ is called the minimum generator *or the* minimum generating set *of X and its elements are called the* generators *of X.*

2. A submonoid X is of the form $\{0\}$ or of the form

$$X = b\big(F \cup (a+\mathbb{N})\big) \tag{2.4}$$

where $b \geq 1$ and $0 \in F \subseteq \{0,\dots,a-1\}$.

Observe that the numeric submonoids (i.e. those generated by a finite subset of $\mathbb{N}$ with greatest common divisor equal to 1, cf. [4, 11]) correspond to $b=1$. They are exactly the submonoids which are cofinite.

Proof. 1. Let b be the greatest common divisor of the elements in X. Then $X \subseteq b\mathbb{N}$. Let $a_1,\dots,a_n \in X$ such that g.c.d.$(a_1,\dots,a_n) = b$. By Bézout there exist $x_1,\dots,x_\ell,y_1,\dots,y_{n-\ell} \in \mathbb{N}$ such that

$$x_1a_1 + \cdots + x_\ell a_\ell - y_1 a_{\ell+1} - \cdots - y_{n-\ell}a_n = b$$

(up to a permutation of $a_1,\dots,a_n$). Let $L = x_1a_1 + \cdots + x_\ell a_\ell$ and $R = y_1a_{\ell+1} + \dots + y_{n-\ell}a_n$ and observe that $L,R \subseteq b\mathbb{N}$. For all integers k, set $k = qa_1 + r$ where $0 \leq r < a_1$ and $q \in \mathbb{N}$. Then

$$(a_1-1)R + kb \;=\; qba_1 + (a_1-1-r)R + rL \;\in\; a_1\mathbb{N} + \cdots + a_n\mathbb{N}\,.$$

which shows that all multiples of b greater than $(a_1-1)R$ are generated by $a_1,\dots,a_n$ hence that X is generated by the finite class consisting of all its elements less than or equal to

$$\max\{a_1,\dots,a_n,(a_1-1)R\} \tag{2.5}$$

It is clear that $G(X)$ as in equation (2.3) generates X. Assume by contradiction that there exists a generating set H not containing $G(X)$ and let α be an element in $G(X) \setminus H$. We assume without loss of generality that $0 \notin H$. Since $X = H^*$ we have $\alpha = \beta + \gamma$ where $\beta \in H$ and $\gamma \in H^* \setminus \{0\}$. Then $\alpha \in S+S$ which contradicts the definition of $G(X)$.

2. Observe that the integer in (2.5) is a multiple of b, say ab. Then in order to obtain (2.4) it suffices to consider the subset F satisfying $bF = X \cap b\{0,\dots,a-1\}$

The specific form of nonzero submonoids leads us to the following general notion.

Definition 2. A set $X \subseteq \mathbb{N}$ has *ultimate period b* if there exists an integer K such that for all $n \geq K$ we have

$$n \in X \iff n+b \in X$$

If this holds for some $b \geq 1$, X is *ultimately periodic.*

The following result is classical.

Proposition 4. *Let $X \subseteq \mathbb{N}$.*
1. The following conditions are equivalent:

(i) *X is ultimately periodic,*

(ii) *X is a regular subset of $\mathbb{N}$.*

(iii) *$X = A \cup (B + b\mathbb{N})$ where $a, b \in \mathbb{N}$, $\emptyset \neq A \subseteq [0, a[$ and $B \subseteq [a, a+p[$.*

The set X is finite if and only if $B = \emptyset$ in (iii).

2. If X has ultimate period b then all multiples of b are ultimate periods.

3. When X is a submonoid, the integer b of equation (2.4) is its minimum ultimate period.

Remark 1. Equation (2.3) of Proposition 3 gives a simple algorithm to compute $G(X) = \{g_0, \ldots, g_n\}$ provided its ultimate period b is known. Suppose the submonoid X has minimum nonzero element m. Let $g_0 = m$ and let g_{i+1} be the minimum element of X not in $\sum_{j=0}^{j=i} g_j\mathbb{N}$. Halt when $\{g_0, \ldots, g_n\}$ generate m/b successive elements of the periodic tail of X.

The above representation of submonoids allows us to express the inclusion and intersection of submonoids simply.

Proposition 5. *Any pair of nonzero submonoids X, Y is of the form*

$$\begin{cases} X = bF \cup (ad + b\mathbb{N}) \\ Y = cG \cup (ad + c\mathbb{N}) \end{cases} \quad \textit{with} \quad \begin{cases} b \geq 1,\ c \geq 1,\ d = \text{l.c.m.}(b, c) \\ 0 \in F \cap G, \quad F, G \textit{ finite} \\ bF \cup cG \subseteq \{0, \ldots, ad-1\} \end{cases}$$

The intersection monoid is $X \cap Y = (bF \cap cG) \cup (ad + d\mathbb{N})$.
In particular, $X \subseteq Y$ if and only if $bF \subseteq cG$ and c divides b.

2.2.2 Maximal Submonoids

Notation 1. *We write $X \lhd Y$ whenever X is an inclusion-maximal proper submonoid of the submonoid Y.*

Proposition 6. *Let X be a submonoid of $\mathbb{N}$ with $G(X)$ as minimum generating set.*

1. The proper maximal submonoids of X are the sets $X \setminus \{g\}$ where $g \in G(X)$.

2. Every generator of X distinct from g is a generator of $X \setminus \{g\}$ (but there may be other ones, cf. Example 1). In other words,

$$G(X) \setminus \{g\} \subseteq G(X \setminus \{g\}) \tag{2.6}$$

Proof. Let $g \in G(X)$. All elements in $X \setminus \{g\}$ are of the form

$$\sum_{h \in G} x_h h \quad \text{with} \quad x_g = 0 \quad \text{or} \quad \sum_{h \in G} x_h \geq 2$$

These elements clearly define a proper submonoid of X and this monoid is maximal. Conversely, consider a proper submonoid X' of X. There exists some $g \in G(X) \setminus X'$ hence $X' \subseteq X \setminus \{g\}$ and equality holds in case X' is maximal.

Finally, for $g \in G(X)$, the following inclusion is straightforward:

$$\big(S \setminus (S+S)\big) \setminus \{g\} \subseteq (S \setminus \{g\}) \setminus \big((S \setminus \{g\}) + (S \setminus \{g\})\big) .$$

Thus, every X-generator distinct from g is an $(X \setminus \{g\})$-generator.

Example 1. The subset $X = \{0\} \cup \{3,5,6\} \cup 8 + \mathbb{N}$ is the submonoid with minimum generating set $\{3,5\}$ (use Remark 1). It thus has two proper maximal submonoids $X_1 \lhd X$ and $X_2 \lhd X$. The minimum generating set of $X_1 = X \setminus \{3\}$ is $\{5,6,8,9\}$ and the minimum generating set of $X_2 = X \setminus \{5\}$ is $\{3,8,10\}$. Thus X_1 has 4 proper maximal submonoids and X_2 has 3 proper maximal submonoids.

Consequently, every submonoid which is not reduced to 0 has finitely many proper maximal submonoids but some monoids fail to have minimal proper supermonoids. The following result characterizes them.

Proposition 7. *The submonoids which have no minimal supermonoid are* $\{0\}$ *and the sets* $b\mathbb{N}$*,* $b \geq 0$*.*

Proof. Suppose Y is a minimal nonzero supermonoid of X and let $G(Y)$ be its minimum generating set. Then $X = Y \setminus \{g\}$ for some $g \in G(Y)$ and $G(Y) \setminus \{g\} \subseteq G(X)$. First, using Proposition 3, we show that $\{0\}$ and the sets $b\mathbb{N}$, $b \geq 1$ have no minimal supermonoid. We argue by way of contradiction.

Case $X = \{0\}$. Then $G(X) = \emptyset$ hence $G(Y) = \{g\}$ so that $Y = g\mathbb{N}$. Now, $2g\mathbb{N}$ is a submonoid strictly between $\{0\}$ and Y, contradicting the minimality of Y over X.

Case $G(X) = \{b\}$*, i.e.* $X = b\mathbb{N}$. For $b = 1$ it is trivial so we assume $b > 1$. Then $G(Y)$ has at most two generators. It cannot have only one generator because $G(X)$ would be empty. Thus, $G(Y) = \{b,c\}$ for some $c \notin b\mathbb{N}$. Then by Propostion 2, $b\mathbb{N} + (b+1)c\mathbb{N}$ is a submonoid strictly between $X = b\mathbb{N}$ and $Y = b\mathbb{N} + c\mathbb{N}$, contradicting the minimality of Y over X.

We now show that every submonoid X distinct from $\{0\}$ and the sets $b\mathbb{N}$, $b \geq 1$, has a minimal supermonoid. This condition on X, together with Equation (2.4) supra, insure that $X = b(F \cup (a + \mathbb{N}))$ with $b \geq 1$, $0 \in F$, $a \geq 2$ and $a - 1 \notin F$. The set $Y = \{0\} \cup b((a-1) + \mathbb{N})$ is clearly a submonoid. By Proposition 2, $X + Y$ is a supermonoid of X. A simple computation shows that $(X+Y) \setminus X = \{b(a-1)\}$, i.e., that $X + Y$ is a minimal supermonoid of X.

2.2.3 Basic Predicates

In further sections we try to evaluate the complexity of the predicates which are expressible in the logic. Here we content ourselves with gathering the most elementary predicates.

We recall that a predicate is Σ_n (resp. Π_n) if it is defined by a formula that begins with some existential (resp. universal) quantifiers and alternates $n-1$ times between series of existential and universal quantifiers. It is Δ_n if it is both Σ_n and Π_n. It is $\Sigma_n \wedge \Pi_n$ if it is equivalent to a conjunction of a Σ_n and a Π_n formulas. We assume the reader has some familiarity with computing the logical complexity. As an example of the type of computation consider the $\Sigma_1 \wedge \Pi_1$ formula

$$\theta(x,y,z,t,u) \equiv \exists t\ \phi(x,y,t)\ \wedge\ \forall u\ \psi(x,z,u)$$

Assume that x is the sole common free variable of ϕ and ψ. Then

$$\begin{aligned}
\theta(x,y,z,t,u) &\iff \exists t\ \forall u\ \big(\phi(x,y,t) \wedge \psi(x,z,u)\big)\\
&\iff \forall u\ \exists t\ \big(\phi(x,y,t) \wedge \psi(x,z,u)\big)\\
\exists x\ \theta(x,y,z,t,u) &\iff \exists x\ \exists t\ \forall u\ \big(\phi(x,y,t) \wedge \psi(x,z,u)\big)
\end{aligned}$$

showing both that the predicate associated to θ is Δ_2 and that $\exists x\,\theta$ is Σ_2. Such a type of computation will not be explicitly carried out in the sequel.

2.2.3.1 Removing Definable Constants

Since we deal with definability in a particular structure, the following classical result in logic will be useful.

Proposition 8. *Let $\mathcal{M}$ be any logical structure and a an element of $\mathcal{M}$. Let $n,p \in \mathbb{N}$. Suppose a is Δ_n definable in $\mathcal{M}$, i.e. $x = a$ is equivalent to a Σ_n formula and to a Π_n formula. Then, for every Σ_p (resp. Π_p) formula $\phi(x,\mathbf{y})$ (with free variables x and possibly some other ones), there exists a $\Sigma_{\max(n,p)}$ (resp. $\Pi_{\max(n,p)}$) formula $\psi(\mathbf{y})$ such that $\phi(a,\mathbf{y})$ is equivalent to $\psi(\mathbf{y})$.*

Proof. Suppose $x = a$ is equivalent to formulas

$$\exists \mathbf{v_1}\ \forall \mathbf{v_2} \ldots Q_n \mathbf{v_n}\ F(\mathbf{v_1},\ldots,\mathbf{v_n},x) \quad , \quad \forall \mathbf{v_1}\ \exists \mathbf{v_2} \ldots R_n \mathbf{v_n}\ G(\mathbf{v_1},\ldots,\mathbf{v_n},x)$$

where Q_n (resp. R_n) is $\forall$ if n is even (resp. odd) and is $\exists$ otherwise.
Letting $s = \max(n,p)$, if ϕ is $\exists \mathbf{z_1}\ \forall \mathbf{z_2} \ldots Q_p \mathbf{z_p}\ A(\mathbf{z_1},\ldots,\mathbf{z_p},x,\mathbf{y})$ then

$$\begin{aligned}
\phi(a,\mathbf{y}) \iff \exists \mathbf{z_1}\ \exists \mathbf{v_1}\ \forall \mathbf{z_2}\ \forall \mathbf{v_2} \ldots Q_s \mathbf{z_s}\ Q_s \mathbf{v_s}&\\
\big(F(\mathbf{v_1},\ldots,\mathbf{v_n},x)\ \wedge\ A(\mathbf{z_1},\ldots,\mathbf{z_p},x,\mathbf{y})\big)&
\end{aligned}$$

and if ϕ is $\forall \mathbf{z_1}\ \exists \mathbf{z_2} \ldots R_p \mathbf{z_p}\ B(\mathbf{z_1},\ldots,\mathbf{z_p},x,\mathbf{y})$ then

$$\phi(a,\mathbf{y}) \iff \forall \mathbf{z_1}\, \forall \mathbf{v_1}\, \exists \mathbf{z_2}\, \exists \mathbf{v_2} \ldots R_s \mathbf{z_s}\, R_s \mathbf{v_s} \\ \big(G(\mathbf{v_1},\ldots,\mathbf{v_n},x) \Rightarrow B(\mathbf{z_1},\ldots,\mathbf{z_p},x,\mathbf{y})\big)$$

2.2.3.2 Basic Constants

Proposition 9. *1. The predicate $X = \emptyset$ is Π_1.*
2. The predicate $X = \{0\}$ is Π_1.
3. The predicate $0 \in X$ is Σ_1.
4. The predicate $X = \mathbb{N}$ is $\Sigma_1 \wedge \Pi_1$.

Proof. 1. $X = \emptyset$ if and only if $\forall Y\ X + Y = X$.
2. $\{0\}$ is the neutral element of $\mathcal{P}(\mathbb{N})$ hence is the unique set satisfying $\forall Y\ X + Y = Y$.
3. $0 \in X$ if and only if $\exists Y\ (Y \neq \emptyset \wedge X + Y = Y)$.
4. $X = \mathbb{N}$ if and only if $0 \in X\ \wedge\ \forall Y\ (0 \in Y \Rightarrow X + Y = X)$.

2.2.3.3 Some Classes of Subsets

The following two classes of subsets are easily definable.

Proposition 10. *1. The class of submonoids of $\mathbb{N}$ is Σ_1 definable.*
2. The class of final segments $\{n + \mathbb{N} \mid n \in \mathbb{N}\}$ is $\Sigma_1 \wedge \Pi_1$ definable.

Proof. 1. X is a submonoid if and only if it is nonempty and $X + X = X$.
2. $X \in \{n + \mathbb{N} \mid n \in \mathbb{N}\}$ if and only if $X \neq \emptyset\ \wedge\ \forall Y\ (0 \in Y \Rightarrow X + Y = X)$.

2.2.3.4 Minimum Element in a Set

The minimum element of a nonempty set X is denoted by $\min X$.

Proposition 11. *The following predicates are definable by formulas of the stated complexity:*

1. (a) $\min X \leq k$ *is* Π_2 *for* $k \geq 1$
   (b) $\min X = 0$ *is* Σ_1
   (c) $\min X = 1$ *is* Π_2
   (d) $\min X = k$ *is* $\Sigma_2 \wedge \Pi_2$ *for* $k \geq 2$
2. (a) $\min X \leq \min Y$ *is* Σ_1
   (b) $\min X = \min Y$ *is* Σ_1
   (c) $\min X \leq \min Y + k$ *is* Π_2 *for* $k \geq 1$
   (d) $\min X = \min Y + 1$ *is* Π_2
   (e) $\min X = \min Y + k$ *is* $\Sigma_2 \wedge \Pi_2$ *for* $k \geq 2$
3. (a) $\min X + \min Y \leq \min Z$ *is* Σ_1
   (b) $\min X + \min Y = \min Z$ *is* Σ_1

Proof. 1a. $\min X \leq k$ if and only if $X \neq \emptyset$ and X is not the sum of $k+1$ sets which do not contain 0 :

$$X \neq \emptyset \ \wedge \ \forall X_1, \dots, X_{k+1} \ (X = X_1 + \cdots + X_{k+1} \Rightarrow \bigvee_{i=1}^{i=k+1} 0 \in X_i)$$

Claim 3 of Proposition 9 yields the stated logical complexity.
1b. Again use claim 3 of Proposition 9.
1c. Express that $\min X \leq 1$ and $\min X \neq 0$.
1d. Express that $\min X \leq k$ and $\min X \not\leq k-1$.
2a. $\min X \leq \min Y$ if and only if

$$\exists A, B, R, S \ \ (0 \in R \ \wedge \ 0 \in S \ \wedge \ X = A + R \ \wedge \ Y = A + B + S)$$

Only if. Set $A = \{\min X\}$, $B = \{\min Y - \min X\}$, and $R = X - \min X$ and $S = Y - \min Y$.
If. $\min X = \min A \leq \min A + \min B = \min Y$.
2b. Express that $\min X \leq \min Y$ and $\min Y \leq \min X$.
2c. $\min X \leq \min Y + k$ if and only if

$$\forall A, R, B_1, \dots, B_{k+1}$$
$$((Y = A + R \ \wedge \ 0 \in R \ \wedge \ X = A + B_1 + \cdots + B_{k+1}) \Rightarrow \bigvee_{i=1}^{i=k+1} 0 \in B_i)$$

2d & 2e. Express that $\min X \leq \min Y + k$ and $\min X \not\leq \min Y + k - 1$.
3a. $\min X + \min Y \leq \min Z$ if and only if

$$\exists A, B, R, S, T \ \ (0 \in R \ \wedge \ 0 \in S$$
$$\wedge \ X = A + R \ \wedge \ Y = B + S \ \wedge \ Z = A + B + T)$$

3b. $\min X + \min Y = \min Z$ if and only if

$$\exists A, B, R, S, T \ \ (0 \in R \ \wedge \ 0 \in S \ \wedge \ 0 \in T$$
$$\wedge \ X = A + R \ \wedge \ Y = B + S \ \wedge \ Z = A + B + T)$$

2.2.3.5 Singleton Sets

Proposition 12. *1. The predicate "*X is a singleton*" is* Π_2.
2. The predicate $X = \{0\}$ *is* Π_1.
3. The predicate $X = \{k\}$ *is* $\Sigma_2 \wedge \Pi_2$ *for* $k \geq 1$.
4. The predicate $Y \neq \emptyset \wedge X = \{\min Y\}$ *is* Π_2

Proof. 1. X is a singleton if and only if

$$X \neq \emptyset \text{ and } \forall Y\ ((Y \neq \emptyset \wedge \min Y = \min X) \Rightarrow \exists Z\ Y = X + Z)$$

Only if. Let $X = \{\ell\}$. Since $\min Y = \ell$ we have $(Y - \ell) + \{\ell\} = Y$ hence we can let $Z = Y - \ell$.

If. Let $Y = \{\min X\}$. Equality $\{\min X\} = X + Z$ implies X and Z are singleton sets and $X = \{\min X\}$ and $Z = \{0\}$.

Complexity: to remove the constant $\emptyset$, apply Proposition 8.

2. Already done in claim 2 of Proposition 9.

3. $X = \{k\}$ if and only if X is a singleton and $\min X = k$. We conclude with claim 3 of Proposition 11.

4. We express that X is a singleton and $\min X = \min Y$.

2.3 Using Submonoids to Approximate and Emulate

2.3.1 Special Cases of Inclusion and Membership

The importance of the submonoids of $\mathbb{N}$ relies on the fact that they provide some approximation of two important relations, namely subset inclusion $Y \subseteq X$ and membership $x \in X$.

2.3.1.1 Special Cases of Inclusion

We will prove in Theorem 11 that the inclusion predicate $Y \subseteq X$ is not expressible in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$. However, when X is a submonoid and Y contains 0 the inclusion is expressible.

Proposition 13. *Let* $0 \in Y$ *and let* X *be a submonoid. Then*

$$Y \subseteq X \iff Y + X = X$$

Proof. Indeed, from right to left we have $X = Y + X \supseteq Y + \{0\} = Y$. Conversely, since $0 \in Y$ we have $X \subseteq Y + X$ and since $Y \subseteq X$ and X is a submonoid we have $Y + X \subseteq X + X = X$.

The following result helps us to tightly evaluate syntactical complexity.

Proposition 14. *The predicate* $\lhd$ *is* $\Sigma_1 \wedge \Pi_1$. *Also, it is* Π_1 *on submonoids: there exists a* Π_1 *formula* $F(X,Y)$ *such that*

$$X, Y \textit{ are submonoids} \Rightarrow (F(X,Y) \Leftrightarrow X \lhd Y)$$

Proof. Indeed, $Y \lhd X$ if and only if

$$\begin{aligned} &X, Y \text{ are submonoids} \wedge Y \subseteq X \wedge Y \neq X \\ &\qquad \wedge \forall Z\ \big((Z \text{ is a submonoid} \wedge Y \subseteq Z \subseteq X) \Rightarrow (Z = Y \vee Z = X)\big) \end{aligned}$$

To conclude we use Propositions 10 and 13.

2.3.1.2 Special Cases of Membership

Concerning the approximation of membership we use a special (whence the notation) type of submonoids which is appropriate to evaluate the complexity of the logic.

Definition 3. For each integer $n \geq 1$ we let

$$S_n = \{0\} \cup (n + \mathbb{N})$$

These submonoids are called *special*. The class of special submonoids is denoted by $\mathtt{Special}$.

E.g., $S_1 = \mathbb{N}$ and $S_2 = \{0\} \cup (2 + \mathbb{N}) = \mathbb{N} \setminus \{1\}$ is the largest proper submonoid of $\mathbb{N}$ since the minimum generating subset of $\mathbb{N}$ is $\{1\}$, cf. Proposition 6 Claim 1.

The following shows that the submonoids S_n, $n \geq 1$, can be used to test membership of a fixed n in X provided n is strictly greater than $\min(X)$. In particular, if X contains 0 then the property "$n > 0$ and belongs to X" for a fixed n is expressible since the restriction n greater than 0 is no longer necessary, see Proposition 9 Claim 3. Definability of the subset S_n is done in Theorem 2.

Lemma 1. *Let $m \in \mathbb{N}$ and $n \geq 1$. If $X \neq \emptyset$ and $m = \min X$ then $m + n \in X$ if and only if $X + S_n = X + S_{n+1}$.*

Proof. Observe that

$$\begin{aligned} X + S_n &= X + (\{0\} \cup (n + \mathbb{N})) \qquad = (X + \{0\}) \cup (X + (n + \mathbb{N})) \\ &= X \cup ((m+n) + \mathbb{N}) \\ X + S_{n+1} &= X \cup ((m+n+1) + \mathbb{N}) \,. \end{aligned}$$

Thus, $X + S_{n+1} \subseteq X + S_n$ and $(X + S_n) \setminus (X + S_{n+1}) = \{m+n\} \setminus X$.

2.3.2 Definability Issues of Special Submonoids

We first show that each S_n can be defined. This is done by carefully investigating their proper maximal submonoids. In a second step we show that the fact of being an S_n, i.e., the class $\{S_n \mid n \geq 1\}$ is definable. This also relies on properties of the containment relation between submonoids.

2.3.2.1 Definability of Each Special Submonoid

Recall the notation $G(X)$ for the minimum generating set of X, Proposition 3.

Lemma 2. *Assume $n \geq 1$.*
1. $G(S_n) = \{n,\dots,2n-1\}$. Thus (cf. Proposition 6), S_n is a monoid with exactly n maximal proper submonoids.
2. $G(S_n \setminus \{n\}) = G(S_{n+1}) = \{n+1,\dots,2n+1\}$. Thus, $S_n \setminus \{n\}$ is a monoid with exactly $n+1$ maximal proper submonoids.
3. $G(S_n \setminus \{n+1\}) = \{n\} \cup \{n+2,\dots,2n-1\} \cup \{2n+1\}$. Thus, $S_n \setminus \{n+1\}$ is a monoid with exactly n maximal proper submonoids.
4. If $n+2 \leq i \leq 2n-1$ then $G(S_n \setminus \{i\}) = \{n, n+1,\dots,i-1,i+1,\dots,2n-1\}$. Thus, $S_n \setminus \{i\}$ is a monoid with exactly $n-1$ maximal proper submonoids.

Proof. The right to left inclusions of the form $G(\dots) \supseteq \dots$ of the four claims are straightforward. We check the left to right inclusions.

1. For all integers $k \geq 0$ we have $2n+k = n+(n+k) \in (X \setminus \{0\}) + (X \setminus \{0\})$.
2. We apply Claim 1 with $n+1$ in place of n.
3. For all $k \geq 2$ we have $2n+k = n+(n+k) \in (X \setminus \{0\}) + (X \setminus \{0\})$.
4. For all $0 \leq k \neq i$ we have $2n+k = n+(n+k) \in (X \setminus \{0\}) + (X \setminus \{0\})$ and for $k = i$ we have $2n+i = (n+1)+(n+i-1)$.

We now convert the previous result formally in our logic.

Theorem 2. *1. The predicate $X = S_1$ is $\Sigma_1 \wedge \Pi_1$.*
2. For each $n \geq 2$, the predicate $X = S_n$ is Δ_2.

Proof. 1. Since $S_1 = \mathbb{N}$ this is Proposition 9 Claim 4.
2. In view of applying Lemma 2, we introduce variables $X_1,\dots,X_n$ to represent $S_1,\dots,S_n$ and consider some formulas involving these variables.

(1) Let A be the formula expressing $X_1 = \mathbb{N}$.
(2) Let B be the formula expressing $X_n \lhd X_{n-1} \lhd \cdots \lhd X_2 \lhd X_1$.
(3) Let C be the formula which expresses that, for $m = 1,\dots,n$, there exist $m+1$ pairwise distinct maximal proper submonoids of X_m.

Lemma 2 insures that every maximal submonoid of S_m has at most m maximal submonoids, except $S_m \setminus \{m\} = S_{m+1}$ which has $m+1$ maximal submonoids. Thus, a straightforward induction on $m = 1,\dots,n$ shows that the formula $A \wedge B \wedge C$ implies that $X_m = S_m$. As a consequence, both formulas

$$\begin{aligned} &\exists X_1 \dots X_n\ (A \wedge B \wedge C \wedge\ X_n = X) \\ \text{and } &\forall X_1 \dots X_n\ (A \wedge B \wedge C \Rightarrow\ X_n = X) \end{aligned}$$

express that $X = S_n$. By Proposition 14, formulas A, B are $\Sigma_1 \wedge \Pi_1$ and formula C is Σ_2. This shows that the predicate $X = S_n$ is Σ_2 and is Π_2.

2.3.2.2 Definability of the Class of Special Submonoids

We now look for a formula defining the class of special submonoids S_n, $n \geq 1$.

Definition 4. If M is a submonoid of $\mathbb{N}$ with m as minimum nonzero element, we denote by ∂M the submonoid $M \setminus \{m\}$ of M.

The following technical result based on Proposition 6 is crucial for a definition of the submonoids S_n. It is general and relates the generators of a submonoid with its maximal submonoids.

Proposition 15. *Let M be a submonoid of $\mathbb{N}$, K a maximal submonoid of M and g a generator of M such that $K = M \setminus \{g\}$. For $k \in \mathbb{N}$, the following conditions are equivalent:*

(1) *There are exactly k generators of $M \setminus \{g\}$ which are not in $G(M)$, i.e. $G(M \setminus \{g\})$ is equal to $G(M) \setminus \{g\}$ augmented with k elements.*
(2) *There are exactly k sets in the class $\mathcal{Z}$ of maximal submonoids L of K such that K is the unique submonoid Z satisfying $L \lhd Z \lhd M$.*
(3) *There are exactly k sets in the class $\mathcal{Y}$ of maximal submonoids L of K such that K is the unique submonoid Z satisfying $L \subsetneq Z \subsetneq M$.*

Proof. We use Proposition 3. Any maximal submonoid L of K is of the form $L = K \setminus \{h\} = M \setminus \{g,h\}$ for some $h \in G(K)$. There are obviously only two sets Z such that $M \setminus \{g,h\} \subsetneq Z \subsetneq M$, namely $M \setminus \{g\} = K$ and $M \setminus \{h\}$. Thus, $K \in \mathcal{Y}$ if and only if $M \setminus \{h\}$ is not a submonoid. Observe that the sole possible reason for a failure of $L \lhd M \setminus \{h\} \lhd M$ is that $M \setminus \{h\}$ is not a submonoid. Thus, $K \in \mathcal{Z}$ if and only if $M \setminus \{h\}$ is not a submonoid. To conclude, observe that $M \setminus \{h\}$ is not a submonoid if and only if $h \notin G(M)$.

Definition 5. Let M be a submonoid of $\mathbb{N}$ and $k \in \mathbb{N}$. A generator g of M is *k-creative* if condition (1) of Proposition 15 holds. A maximal submonoid K of M is k-creative if condition (2) of Proposition 15 holds, i.e. if $K = M \setminus \{g\}$ where g is a k-creative generator of M.

We shall write $(\geq \ell)$-creative to mean k-creative for some $k \geq \ell$.

Proposition 3 yields the following result.

Lemma 3. *If M is a submonoid different from $\{0\}$ then $m = \min(M \setminus \{0\})$ is a (≥ 2)-creative generator of M. Thus, ∂M is a (≥ 2)-creative maximal submonoid of M.*

Proof. Since m cannot be a sum of two nonzero elements of M we see that $m \in G(M)$. Also, $G(M \setminus \{m\}) \supseteq \{2m, m+p\}$ hence g is (≥ 2)-creative.

Definition 6. A submonoid M of $\mathbb{N}$ is *good* if ∂M is its unique maximal submonoid which is is (≥ 2)-creative, i.e. $\min M$ is the sole (≥ 2)-creative generator of M.

Lemma 4. *The submonoids S_n, $n \geq 2$, are good.*

Proof. We use Lemma 2. The generators of S_n are $n,\dots,2n-1$.

Case of $S_n\setminus\{n\}$. The generators of $S_n\setminus\{n\}=S_{n+1}$ are $n+1,\dots,2n+1$. Only two of them (namely, $2n,2n+1$) are not in $G(S_n)$. Thus, n is a 2-creative generator of S_n.

Case of $S_n\setminus\{n+1\}$. The generators of $S_n\setminus\{n+1\}$ are the elements in $\{n\}\cup\{n+2,\dots,2n-1\}\cup\{2n+1\}$. Only one of them (namely, $2n+1$) is not in $G(S_n)$. Thus, $n+1$ is a 1-creative generator of S_n.

Case of $S_n\setminus\{i\}$ *with* $n+2\le i\le 2n-1$. The generators of $S_n\setminus\{i\}$ are the elements in $\{n,\dots,2n-1\}\setminus\{i\}$. All of them are in $G(S_n)$. Thus, i is a 0-creative generator of S_n.

This shows that $n=\min S_n$ is the sole (≥ 2)-creative generator of S_n.

The submonoids S_n are not the sole examples of good submonoids.

Example 2. The monoid $M=\{0,6,7,8,9\}\cup(11+\mathbb{N})$ with minimum generating set $\{6,7,8,9,11\}$ is good. Indeed, using Remark 1, we get the facts shown in the following table:

g	$M\setminus\{g\}$	$G(M\setminus\{g\})$	
6	$\{0,7,8,9\}\cup(11+\mathbb{N})$	$\{7,8,9,11,\mathbf{12},\mathbf{13}\}$	6 is 2-creative
7	$\{0,6,8,9\}\cup(11+\mathbb{N})$	$\{6,8,9,11,\mathbf{13}\}$	7 is 1-creative
8	$\{0,6,7,9\}\cup(11+\mathbb{N})$	$\{6,7,9,11\}$	8 is 0-creative
9	$\{0,6,7,8\}\cup(11+\mathbb{N})$	$\{6,7,8,11\}$	9 is 0-creative
11	$\{0,6,7,8,9\}\cup(12+\mathbb{N})$	$\{6,7,8,9\}$	11 is 0-creative

Remark 2. Not every submonoid is good. For instance, the submonoid $X=\{0,3,5,6\}\cup(8+\mathbb{N})$ in Example 1 has two (≥ 2)-creative generators. Indeed, $G(X)=\{3,5\}$ and $G(X\setminus\{3\})=\{5,6,8,9\}$ and $G(X\setminus\{5\})=\{3,8,10\}$ hence 3 is 3-creative and 5 is 2-creative.

Lemma 5. *1. The following predicate is* Σ_2 *:*

$$K \text{ is a } (\ge 2)\text{-creative maximal submonoid of the submonoid } M$$

2. The class of good submonoids is Π_2.

Proof. 1. Using condition (2) of Proposition 15, let $A(M,K)$ be the formula

$$\begin{aligned} K\lhd M \ \wedge\ \exists L_1,L_2\ \big(L_1\neq L_2\ \wedge\ L_1\lhd K\ \wedge\ L_2\lhd K \\ \wedge\ \forall L\ (L \text{ is a submonoid } \Rightarrow \\ \big(L=K\Leftrightarrow(L_1\subsetneq L\subsetneq M)\big)\ \wedge\ \big(L=K\Leftrightarrow L_2\subsetneq L\subsetneq M)\big)\big)\end{aligned}$$

Since the class of submonoids is Σ_1 and the predicate $\lhd$ is $\Sigma_1\wedge\Pi_1$ and inclusion of submonoids is Σ_0 (cf. Propositions 13, 10, 14), the above formula $A(M,K)$ is Σ_2.

2. By definition M is good if and only if M is a submonoid and there exists a unique K such that $A(M,K)$. Using Lemma 3, we see that it suffices to say that there exists at most one K such that $A(M,K)$, i.e.

$$M \text{ is a submonoid } \wedge\ \forall K'\forall K''\ ((A(M,K') \wedge A(M,K'')) \Rightarrow K' = K'')$$

Since formula A is Σ_2, this formula is Π_2.

Lemma 6. *The following predicate is* $\Sigma_2 \wedge \Pi_2$

$$\{(L,M) \mid M \textit{ is a good submonoid and } L = \partial M\}$$

Proof. By Lemma 3, when M is good, ∂M is the unique K such that $A(M,K)$. Thus, the formula M *is good* $\wedge\ A(M,L)$ defines the considered predicate. Since $A(M,L)$ is Σ_2 (cf. the proof of Lemma 5), this formula is $\Sigma_2 \wedge \Pi_2$.

We can now get a useful extension of Lemma 1.

Lemma 7. *Let L,M be submonoids and m be the minimum nonzero element of M. Then $m \in L$ if and only if $L + \partial M = L + M$.*

Proof. Trivially, if $m \in L$ then $L + \partial M = L + M$. Suppose now $L + \partial M = L + M$. Since $m \in L + M$ we have $m \in L + \partial M$ hence $m = x + y$ with $x \in L$ and $y \in \partial M$. Since all nonzero elements of ∂M are strictly greater than m we have $y = 0$ hence $m = x \in L$.

The following proves the definability of the class of special submonoids S_n.

Theorem 3. *The class* $\texttt{Special} = \{S_n \mid n \geq 1\}$ *is* Π_3.

Proof. Consider the following formula $\texttt{Special}(X)$ which (by Lemma 6) is Π_4 and (by Lemma 7 and Proposition 13) expresses that X is a submonoid and, for any good M with m as minimum nonzero element, if $m \in X$ then $M \subseteq X$:

$$\begin{aligned} &X \text{ is a submonoid and } \forall M\ \forall L \\ &\qquad \big(M \text{ is a good submonoid and } L = \partial M \text{ and } X + L = X + M \\ &\qquad\qquad\qquad\qquad\qquad\qquad \Rightarrow X + M = X\big) \end{aligned}$$

The submonoids S_n clearly satisfy this property. Conversely, if X satisfies this property and n is the minimum nonzero element of X then, applying the property with the good submonoid $M = S_n$, we get $S_n \subseteq X$ hence $S_n = X$.

Proposition 16. *The following predicates are* Π_3 *:*

$$\begin{aligned} \texttt{Succ}_1(X,Y) &\equiv (X,Y) \in \{(S_n, S_{n+1}) \mid n \geq 1\} \\ \texttt{Succ}_k(X,Y) &\equiv (X,Y) \in \{(S_n, S_{n+k}) \mid n \geq 1\} \\ \texttt{Succ}_*(X,Y) &\equiv (X,Y) \in \{(S_n, S_{n+k}) \mid n,k \geq 1\} \end{aligned}$$

For $k = 1$ we simply write $\texttt{Succ}$ *in place of* $\texttt{Succ}_1$.

Proof. Observe that

$$\begin{aligned}
\mathtt{Succ}_1(X,Y) &\iff X \text{ is special and } Y = \partial X \\
\mathtt{Succ}_k(X_0,Y) &\iff \mathtt{Special}(X_0) \wedge \forall X_1 \ldots \forall X_k \\
&\qquad \left(\left(\bigwedge_0^{k-1} X_i \text{ is good and } X_{i+1} = \partial X_i \right) \Rightarrow Y = X_k \right) \\
\mathtt{Succ}_*(X,Y) &\iff X,Y \text{ are special and } Y \subseteq X
\end{aligned}$$

then apply Theorem 3, Lemma 6 and Proposition 13.

2.3.3 *Addition and Multiplication on Special Submonoids*

Here we show that the set of submonoids of the form S_n, with $n \geq 1$, can be equipped with two definable operations $\oplus$ and $\otimes$ which make it isomorphic to $\langle \mathbb{N} \setminus \{0\}; +, \times, = \rangle$.

2.3.3.1 Insight into the Proof

This paragraph is meant to give some intuition behind the formal proofs of the next two ones. The idea is to define two operations on the family of special submonoids, namely an addition $(S_n, S_p) \to S_{n+p}$ and a multiplication $(S_n, S_p) \to S_{n \times p}$.

The addition is defined via the finite initial segments by observing that $\{0,\ldots,n+p\} = \{0,\ldots,n\} + \{0,\ldots,p\}$ holds and by using the correspondence $S_n \mapsto \{0,\ldots,n\}$ (which is definable, cf. Proposition 20).

Based on a number theoretic result which we recall below, multiplication can be expressed by using addition and divisibility. Divisibility is defined via the sets of the form $n\mathbb{N}$ by observing that n divides p if and only if $p\mathbb{N} \subseteq n\mathbb{N}$ and by using the correspondence $S_n \mapsto n\mathbb{N}$ (which is definable, cf. Proposition 22).

Lemma 8. *Multiplication on $\mathbb{N}$ is definable from addition and divisibility. More precisely, it is Δ_1 relative to addition and some predicates which are themselves Π_1 relative to divisibility.*

Proof. Schnirelman's famous result (1931) insures the existence of a constant K such that every integer ≥ 2 is the sum of at most K primes. Olivier Ramaré, [8], showed that $K \leq 7$. If $x = \sum_{i=1}^{a} p_i$ and $y = \sum_{j=1}^{b} q_j$ with $a,b \leq 7$ and the numbers p_i, q_j are primes then $x \times y = \sum_{i=1}^{a} \sum_{j=1}^{b} p_i \times q_j$. Now, the product of two primes p,q (distinct or not) is the unique number with p,q as sole proper divisors. Let $s \,|\, t$ mean that s is a divisor of t, let $P(x)$ mean that x is prime and let $A(x,p,q)$ mean that p,q are prime and $x = p \times q$. Then

$$\begin{aligned}
P(p) &\equiv p \neq 1 \wedge \forall s\,(s\,|\,p \iff s = 1 \vee s = p)\\
A(x,p,q) &\equiv P(p) \wedge P(q) \wedge x \neq p,q\\
&\qquad \wedge\, \forall s\,(s\,|\,x \iff s = 1 \vee s = x \vee s = p \vee s = q)
\end{aligned}$$

are Π_1 relative to divisibility. Also, the predicate $z = x \times y$ is expressed as the conjunction of the formulas $(x = 0 \vee y = 0) \Rightarrow z = 0$, $x = 1 \Rightarrow z = y$ and $y = 1 \Rightarrow z = x$ and any one of the following formulas:

$$E(x,y,z) \equiv x,y \geq 2 \Rightarrow \bigvee_{a,b\in\{1,\ldots,7\}} \exists(x_i,y_j,z_{i,j})^{1\leq j\leq b}_{1\leq i\leq a}\ \left(\varphi\ \wedge\ z = \textstyle\sum_{i,j} z_{i,j}\right)$$

$$F(x,y,z) \equiv x,y \geq 2 \Rightarrow \bigvee_{a,b\in\{1,\ldots,7\}} \forall(x_i,y_j,z_{i,j})^{1\leq j\leq b}_{1\leq i\leq a}\ \left(\varphi \Rightarrow z = \textstyle\sum_{i,j} z_{i,j}\right)$$

where φ is $x = \sum_i x_i\ \wedge\ y = \sum_j y_i\ \wedge \bigwedge_{i,j} P(x_i) \wedge P(y_j) \wedge A(z_{i,j}, x_i, y_j)$.

2.3.3.2 Addition on Special Submonoids

Definition 7. We denote by `Initial` the class of all initial segments $\{0,\ldots,n\}$ for some $n \in \mathbb{N}$.

Proposition 17. *The class* `Initial` *is* Π_4

Proof. Observe that $X \in$ `Initial` if and only if $0 \in X$ and $X \neq \mathbb{N}$ and, for all $x \geq 2$, if $x \in X$ then $x - 1 \in X$. This is expressible as follows:

$$\begin{aligned}
&0 \in X\ \wedge\ X \neq \mathbb{N}\ \wedge \forall Z,T,U\\
&\qquad\big((\texttt{Succ}(Z,T) \wedge \texttt{Succ}(T,U) \wedge X + T = X + U) \Rightarrow X + Z = X + T\big)
\end{aligned}$$

using the predicate `Succ` defined in Proposition 16 and Lemma 1.

As explained in paragraph 2.3.3.1 we view an integer as the maximal element of an initial segment which allows us to indirectly express that it belongs to some subset containing 0.

Proposition 18. *The following two predicates are* Π_4

$$X \in \texttt{Initial}\ \wedge\ 0 \in Y\ \wedge\ \max X \in Y\quad,\quad X \in \texttt{Initial}\ \wedge\ 0 \in Y\ \wedge\ \max X \notin Y$$

Proof. Observe that the maximum element of a finite initial segment X non reduced to $\{0\}$ is the integer n such that $n \in X$ and $n + 1 \notin X$. Thus, $\max X \in Y$ is expressed by the formula

$$\begin{aligned}
&\texttt{Initial}(X)\ \wedge\ 0 \in Y\ \wedge \big(X \neq \{0\} \Rightarrow \forall Z\, \forall T\, \forall U\\
&\qquad \big(\texttt{Succ}(Z,T) \wedge \texttt{Succ}(T,U) \wedge (X + Z = X + T) \wedge (X + T \neq X + U)\\
&\qquad\qquad \Rightarrow Y + Z = Y + T\big)
\end{aligned}$$

Idem for the second predicate with $\max X \notin Y$: just replace the last equality $Y+Z = Y+T$ by an inequality. The stated complexity comes from Propositions 17, 16 and 9.

Proposition 19. *The following predicate is* Π_4

$$X,Y,Z \in \texttt{Initial} \ \wedge\ \max X + \max Y = \max Z$$

Proof. Observe that equality $\max X + \max Y = \max Z$ is equivalent to $X+Y = Z$ when X,Y,Z are finite initial segments.

Proposition 20. *The following predicate is* Π_4 *:*

$$X \in \texttt{Initial} \ \wedge\ X \neq \{0\} \ \wedge\ Y = S_{\max X}$$

Proof. Observe that $\max X$ is the largest n such that $\max X \in S_n$. Thus, the predicate can be expressed as follows:

$$\begin{aligned} X \in \texttt{Initial} \ \wedge\ X \neq \{0\} \ \wedge\ \max X \in Y \wedge \texttt{Special}(Y) & \\ \wedge\ \forall Z\ (\texttt{Succ}(Y,Z) \Rightarrow \max X \notin Z) & \end{aligned}$$

using Proposition 18 and Lemma 1.

Theorem 4. *The relation* $\{(S_n, S_p, S_{n+p}) \mid n,p \geq 1\}$ *is* Δ_5.
We write $T \oplus U = V$ *if* (T,U,V) *is in this relation.*

Proof. Using Propositions 20 and 19, $T \oplus U = V$ holds if and only if it any one of the following formulas holds

$$\varphi\ \wedge\ \exists I\ \exists J\ \exists K\ (\psi\ \wedge I+J=K) \quad , \quad \varphi\ \wedge\ \forall I\ \forall J\ \forall K\ (\psi\ \Rightarrow I+J=K)$$

$$\text{where} \begin{cases} \varphi \equiv \texttt{Special}(T)\ \wedge\ \texttt{Special}(U)\ \wedge\ \texttt{Special}(V) \\ \psi \equiv I,J,K \in \texttt{Initial}\ \wedge\ I,J,K \neq \{0\} \\ \qquad \wedge\ T = S_{\max I}\ \wedge\ U = S_{\max J}\ \wedge\ V = S_{\max K} \end{cases} .$$

2.3.3.3 Multiplication on Special Submonoids

We introduce two useful predicates.

Proposition 21. *The predicate* $\texttt{Periodic} = \{n\mathbb{N} \mid n \geq 1\}$ *is* Π_2.

Proof. By Proposition 7, the sets $n\mathbb{N}$, $n \geq 1$, are the nonzero submonoids with no minimal supermonoid:

$$\texttt{Periodic}(X) \iff (X \neq \{0\} \text{ is a submonoid and } \forall Y\ \neg(X \lhd Y))$$

Conclude with Proposition 14.

Proposition 22. *1. The predicate $\mathcal{B} = \{(S_n, n\mathbb{N}) \mid n \geq 1\}$ is Π_3.*
2. The relation $\{(S_n, S_p) \mid n \geq 1$ and n divides $p\}$ is Δ_4.

Proof. 1. Recall that $\mathtt{Periodic}(X)$ is the Π_2 predicate of Proposition 21. Observing that $n\mathbb{N}$ is the unique submonoid which is included in S_n and not in $\partial(S_n) = S_{n+1}$, the predicate $(X,Y) \in \mathcal{B}$ is expressible as

$$\mathtt{Special}(X) \wedge \mathtt{Periodic}(Y) \wedge Y \subseteq X \wedge \forall Z\, (Z = \partial(X) \Rightarrow Y \not\subseteq Z)$$

Theorem 3, Proposition 13 and Lemma 6 give the complexity.
2. Recall that n divides p if and only if $p\mathbb{N} \subseteq n\mathbb{N}$. Thus, the formulas

$$\begin{cases} \exists Y\, \exists Y'\, (\mathcal{B}(X,Y) \wedge \mathcal{B}(X',Y') \wedge Y' \subseteq Y) \\ \forall Y\, \forall Y'\, ((\mathcal{B}(X,Y) \wedge \mathcal{B}(X',Y')) \Rightarrow Y' \subseteq Y) \end{cases}$$

(which are Σ_4 and Π_4) define the divisibility predicate on the submonoids S_n.

Theorem 5. *The relation $\{(S_n, S_p, S_{n\times p}) \mid n,p \geq 1\}$ is Δ_5. We write $T \otimes U = V$ if (T,U,V) is in this relation.*

Proof. Do the following in the formulas $E(x,y,z)$ and $F(x,y,z)$ given in the proof of Lemma 8 (and involving the predicates P and A):
- replace the variables $x, y, z, x_i, y_j, z_{i,j}$ by $X, Y, Z, X_i, Y_j, Z_{i,j}$,
- replace addition by the Δ_5 predicate $\oplus$ (cf. Theorem 4),
- using claim 2 of Proposition 22, replace the predicates P and A which are Π_1 relative to divisibility, by Π_4 predicates in $X, Y, Z, X_i, Y_j, Z_{i,j}$.

We now extend the two elementary operations of addition and multiplication to an arbitrary polynomial.

Corollary 1. *Let $T(x_1, \ldots, x_k)$ be a polynomial with non zero coefficients in $\mathbb{N}$ and variables in $\{x_1, \ldots, x_n\}$ with $n \geq 1$. The following relation is Δ_5 :*

$$\{(S_{n_1}, \ldots, S_{n_k}, S_p) \mid n_1, \ldots, n_k \geq 1 \text{ and } p = T(n_1, \ldots, n_k)\}$$

Proof. There are polynomials $T_0, \ldots, T_s$ such that $T = T_s$ and, for $0 \leq i \leq s$, T_i is the constant 1 or a variable or $T_j + T_\ell$ or $T_j \times T_\ell$ with $j, \ell < i$. Let I be the set of numbers i such that $T_i = 1$, let J be the set of pairs (i,m) such that $T_i = x_m$, A (resp. M) be the set of triples (i,j,ℓ) such that $T_i = T_j + T_\ell$ (resp. $T_i = T_j \times T_\ell$). The relation is expressed by either of the following formulas with free variables $X_1, \ldots, X_k, X$:

$$\exists Z_1 \ldots \exists Z_s\, (\varphi \wedge X = Z_s) \quad , \quad \forall Z_1 \ldots \forall Z_s\, (\varphi \Rightarrow X = Z_s)$$

where φ is $\bigwedge_{i\in I} Z_i = S_1 \wedge \bigwedge_{(i,m)\in J} Z_i = X_m$
$\wedge \bigwedge_{(i,j,\ell)\in A} Z_i = Z_j \oplus Z_\ell \wedge \bigwedge_{(i,j,\ell)\in M} Z_i = Z_j \otimes Z_\ell$

By Theorems 2, 4, 5, these formulas are respectively Σ_5 and Π_5.

2.4 Complexity of the Theory

2.4.1 Emulating Second-Order Arithmetic

Here, we show that we can interpret the second-order theory of arithmetic in the theory of $\langle \mathcal{P}(\mathbb{N}); =, + \rangle$.

Theorem 6. *1. To each second-order arithmetical formula φ one can computably associate a formula* $\mathtt{Trad}(\varphi)$ *so that if φ has m free first-order variables and n free second-order variables then* $\mathtt{Trad}(\varphi)$ *has $m+n$ free variables and, for all $a_1, \ldots, a_m \in \mathbb{N}$ and $A_1, \ldots, A_n \subseteq \mathbb{N}$,*

$$\begin{gathered}\langle \mathbb{N}, \mathcal{P}(\mathbb{N}); =, \in, 1, +, \times \rangle \models \varphi(a_1, \ldots, a_m, A_1, \ldots, A_n) \iff \\ \langle \mathcal{P}(\mathbb{N}); =, + \rangle \models \mathtt{Trad}(\varphi)(S_{1+a_1}, \ldots, S_{1+a_m}, \{0\} \cup (1+A_1), \ldots, \{0\} \cup (1+A_1))\end{gathered}$$

2. If φ is quantifier-free then $\mathtt{Trad}(\varphi)$ *can be taken either Σ_5 or Π_5. If φ is in prenex form with a nonempty quantifier prefix of the form $Q_1\xi_1 \ldots Q_k\xi_k$ where the variables ξ_i are first or second order variables and there are ℓ alternating blocks of quantifiers $\exists, \forall$ in $Q_1 \ldots Q_k$ then* $\mathtt{Trad}(\varphi)$ *can be taken $\Sigma_{\ell+4}$ if $Q_1 = \exists$ and $\Pi_{\ell+4}$ if $Q_1 = \forall$.*

Proof. 1. The transformation $\mathtt{Trad}$ is defined as the composition $\Omega \circ \Theta$ of two reductions. The first reduction Θ allows to go from the second-order arithmetical structure of $\mathbb{N}$ to that of $\mathbb{N} \setminus \{0\}$. The second reduction Ω allows to go to the structure $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$.

The reduction Θ maps a second-order arithmetical formula φ to another such formula $\Theta(\varphi)$ with the same free variables so that, for all $a_1, \ldots, a_m \in \mathbb{N}$ and $A_1, \ldots, A_n \subseteq \mathbb{N}$,

$$\begin{gathered}\langle \mathbb{N}, \mathcal{P}(\mathbb{N}); =, \in, 1, +, \times \rangle \models \varphi(a_1, \ldots, a_m, A_1, \ldots, A_n) \iff \\ \langle \mathbb{N} \setminus \{0\}, \mathcal{P}(\mathbb{N} \setminus \{0\}); =, \in, 1, +, \times \rangle \models \Theta(\varphi)(1+a_1, \ldots, 1+a_m, 1+A_1, \ldots, 1+A_n)\end{gathered}$$

Thus, in $\Theta(\varphi)$, integer quantifications are over $\mathbb{N} \setminus \{0\}$ and set quantifications are over $\mathcal{P}(\mathbb{N} \setminus \{0\})$. This is simply done by replacing in φ any equation over integers $Q(x_1, \ldots, x_k) = R(x_1, \ldots, x_k)$ by the equation obtained from $Q(x_1 - 1, \ldots, x_k - 1) = R(x_1 - 1, \ldots, x_k - 1)$ by developing and moving any monomial with negative coefficient from one side to the other side of the equation. For instance, $\Theta(x + y = z)$ is obtained by the above process from $(x-1) + (y-1) = z - 1$ hence $\Theta(x + y = z)$ is $x + y = z + 1$; similarly $\Theta(x \times y = z)$ is obtained from $(x-1) \times (y-1) = z - 1$ and is therefore $x \times y + 2 = x + y + z$. As for subformulas $x \in X$, they are left unchanged.

We now define Ω which maps a second-order arithmetical formula ψ to a formula $\Omega(\psi)$ with the same number of free variables so that, for all $b_1, \ldots, b_m \in \mathbb{N} \setminus \{0\}$ and $B_1, \ldots, B_n \subseteq \mathbb{N} \setminus \{0\}$,

$$\begin{aligned}&\langle \mathbb{N}\setminus\{0\},\mathcal{P}(\mathbb{N}\setminus\{0\});=,\in,1,+,\times\rangle \models \psi(b_1,\dots,b_m,B_1,\dots,B_n)\\ &\qquad\iff \langle\mathcal{P}(\mathbb{N});=,+,\rangle \models \Omega(\psi)(S_{b_1},\dots,S_{b_m},\{0\}\cup B_1,\dots,\{0\}\cup B_n)\end{aligned}$$

First, we distinguish two disjoint infinite families of set variables $(U_i)_{i\in\mathbb{N}}$ and $(V_i)_{i\in\mathbb{N}}$. The variables U_i in $\Omega(\psi)$ are to vary over the class of special submonoids (i.e. the sets S_n), they correspond to first-order variables in ψ: if x_i takes value $n\in\mathbb{N}\setminus\{0\}$ then U_i is to take value $S_n\in\mathcal{P}(\mathbb{N})$. The variables V_i in $\Omega(\psi)$ correspond to the second-order variables in ψ : if X_i takes value $B\in\mathcal{P}(\mathbb{N}\setminus\{0\})$ then V_i is to take value $\{0\}\cup B\in\mathcal{P}(\mathbb{N})$.

In view of Claim 2, we inductively define two variants of Ω, namely $\Omega^{\exists}$ and $\Omega^{\forall}$.

(1) If ψ is an atomic formula $Q(x_1,\dots,x_k)=R(x_1,\dots,x_k)$ then $\Omega^{\exists}(\psi)$ and $\Omega^{\forall}(\psi)$ are the Σ_5 and Π_5 formulas

$$\begin{aligned}\Omega^{\exists}(\psi)&\equiv \exists U\ \big(A(U_1,\dots,U_k,U)\ \wedge\ B(U_1,\dots,U_k,U)\big)\\ \Omega^{\forall}(\psi)&\equiv \mathtt{Special}(U_k)\ \wedge\dots\wedge\ \mathtt{Special}(U_k)\ \wedge\ \forall U\ \forall U'\\ &\qquad((A(U_1,\dots,U_k,U)\ \wedge\ B(U_1,\dots,U_k,U'))\Rightarrow U=U')\end{aligned}$$

where A,B are Σ_6 formulas associated to Q and R by Corollary 1 (thus, the i-th first-order variable x_i is replaced by the variable U_i). Note: the subformulas $\mathtt{Special}(U_i)$ are omittted in $\Omega^{\exists}(\psi)$ since they are implied by $A(U_1,\dots,U_k,U)$.

(2) If ψ is $x_i\in X_m$ then, relying on Lemma 1, $\Omega^{\exists}(\psi)$ and $\Omega^{\forall}(\psi)$ are the Σ_4 and Π_4 formulas (cf. Proposition 16)

$$\begin{aligned}\Omega^{\exists}(\psi)&\equiv 0\in V_m\ \wedge\ \exists U\ (\mathtt{Succ}(U_i,U)\ \wedge\ V_m+U_i=V_m+U)\\ \Omega^{\forall}(\psi)&\equiv \mathtt{Special}(U_i)\ \wedge\ 0\in V_m\\ &\qquad\wedge\forall U\ (\mathtt{Succ}(U_i,U)\ \Rightarrow\ V_m+U_i=V_m+U)\end{aligned}$$

(3) $\Omega^{\exists}$ and $\Omega^{\forall}$ commute with conjunction and disjunction.

(4) $\Omega^{\exists}(\neg\varphi)=\neg\Omega^{\forall}(\varphi)$ and $\Omega^{\forall}(\neg\varphi)=\neg\Omega^{\exists}(\varphi)$.

(5) $\Omega^{\exists}(\exists x_i\ \psi)=\Omega^{\forall}(\exists x_i\ \psi)=\exists U_i\ (\mathtt{Special}(U_i)\wedge\Omega^{\exists}(\psi))$
$\Omega^{\exists}(\forall x_i\ \psi)=\Omega^{\forall}(\forall x_i\ \psi)=\forall U_i\ (\mathtt{Special}(U_i)\Rightarrow\Omega^{\forall}(\psi))$

(6) $\Omega^{\exists}(\exists X_m\ \psi)=\Omega^{\forall}(\exists X_m\ \psi)=\exists V_m\ (0\in V_m\wedge\Omega^{\exists}(\psi))$
$\Omega^{\exists}(\forall X_m\ \psi)=\Omega^{\forall}(\forall X_m\ \psi)=\forall V_m\ (0\in V_m\Rightarrow\Omega^{\forall}(\psi))$

Letting Ω be either $\Omega^{\exists}$ or $\Omega^{\forall}$, Corollary 1 and Lemma 1, show that the above clauses insure the wanted property of Ω hence also those of $\mathtt{Trad}:\varphi\mapsto\Omega(\Theta(\varphi))$.
2. The assertion about quantifier-free formulas φ is clear from clauses (1) and (2). An easy induction on the complexity of φ shows that if φ has ℓ alternating blocks of quantifiers and the last one is a Q-block then $\mathrm{Trad}(\Omega^{Q}(\Theta(\varphi)))$ is $\Sigma_{\ell+4}$ if the first block is a $\exists$-block and is $\Pi_{\ell+4}$ if the first block is a $\forall$-block.

2.4.2 The Theory of Addition on Sets Is Undecidable

The complexity results concerning the theory $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ are direct consequences of the results in the previous sections.

Theorem 7. *The class of Σ_5 sentences true in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ is undecidable.*

Proof. Use the undecidability of the Diophantine theory of $\langle \mathbb{N}; =, 1, +, \times \rangle$ (Matiyasevich's celebrated result) and Theorem 6.

The theory of addition on sets is, in fact, *highly* undecidable.

Theorem 8. *The class $\mathcal{T}$ of sentences true in $\langle \mathcal{P}(\mathbb{N}); =, + \rangle$ is recursively isomorphic to the second order theory $\mathcal{A}$ of $\langle \mathbb{N}; =, +, \times \rangle$, i.e. there exists a computable bijection θ between the set of first-order formulas in the language $\{=, +\}$ and the set of second-order formulas in the language $\{=, +, \times\}$ such that $\mathcal{T} = \theta^{-1}(\mathcal{A})$.*

Remark 3. The class of sentences with quantifications over $\mathbb{N}$ only which are true in $\langle \mathbb{N}; =, 1, +, \times \rangle$ (i.e. the first order theory of arithmetic) is Δ^1_1 and not Σ^0_n for any $n \in \mathbb{N}$. As for the class of sentences with quantifications over $\mathbb{N}$ and over $\mathcal{P}(\mathbb{N})$ which are true in $\langle \mathbb{N}, \mathcal{P}(\mathbb{N}); =, \in, 1, +, \times \rangle$ (i.e. the second order theory of arithmetic), it is Δ^2_1 and not Σ^1_n for any $n \in \mathbb{N}$. Thus, the Turing degree of the second order theory of arithmetic is an order of magnitude higher than that of the first order theory.

Proof (Proof of Theorem 8). Recall Myhill's isomorphism theorem which is the computable analog of Cantor-Bernstein's theorem in set theory (cf. [10] Theorem VI page 85, or [7] Theorem III.7.13 page 325): if $X, Y \subseteq \mathbb{N}$ and there exists computable injective maps $\varphi : \mathbb{N} \to \mathbb{N}$ and $\psi : \mathbb{N} \to \mathbb{N}$ such that $X = \varphi^{-1}(Y)$ and $Y = \psi^{-1}(X)$ then there exists a computable bijective map $\theta : \mathbb{N} \to \mathbb{N}$ such that $X = \theta^{-1}(Y)$. Thus, to prove the theorem it suffices to get injective computable reductions of $\mathcal{T}$ to $\mathcal{A}$ and of $\mathcal{A}$ to $\mathcal{T}$.

Recursive reduction of $\mathcal{T}$ to $\mathcal{A}$. To each sentence about $\langle \mathcal{P}(\mathbb{N}); =, + \rangle$ associate the second order arithmetical formula obtained by replacing any subformula $X + Y = Z$ by

$$\forall r \, (r \in Z \iff \exists p, q \, (n = p + q \wedge p \in X \wedge q \in Y) \; .$$

Recursive reduction of $\mathcal{A}$ to $\mathcal{T}$. Use Theorem 6.

2.5 Non Definable Predicates

2.5.1 What Is So Special about the Predicate $0 \in X$?

As implicitly used in numerous instances in the previous sections, the existence of 0 in a subset seems to be the crux for proving remarkable properties such as those in paragraph 2.3.1. If the logic could allow us to add 0 to an arbitrary subset then we could extend these properties to all subsets. However this is not possible. We show that the following predicate is not definable:

$$Y = X \cup \{0\} \tag{2.7}$$

The proof uses two ingredients. The first one, cf. Lemma 9 below, insures that equations and inequations between set variables can be split into conditions on elements of $\mathbb{N}$ and conditions on subsets which are either empty or contain 0. This transformation is lifted to formulas in Lemma 10 below. The second ingredient, cf. Lemma 11 below, is a general result about formulas consisting of combinations of claims on disjoint sets of variables.

Notation 9. *Let $\mathcal{P}_0(\mathbb{N})$ be the class of sets which contain 0. Let $\mathcal{P}_{0,\emptyset}(\mathbb{N}) = \mathcal{P}_0(\mathbb{N}) \cup \{\emptyset\}$ be the class of sets which contain 0 or are empty. Let*

(1) *$\mathcal{E}$ be the subset $\{\emptyset\}$ of $\mathcal{P}_{0,\emptyset}$*

(2) *$+_{\mathbb{N}}$ and $=_{\mathbb{N}}$ be addition and equality on $\mathbb{N}$,*

(3) *$+_{\mathcal{P}_{0,\emptyset}(\mathbb{N})}$ and $=_{\mathcal{P}_{0,\emptyset}(\mathbb{N})}$ be addition and equality of sets in $\mathcal{P}_{0,\emptyset}(\mathbb{N})$.*

We consider the following two sort structure:

$$\mathcal{M} = \langle \mathbb{N}, \mathcal{P}_{0,\emptyset}(\mathbb{N}); +_{\mathbb{N}}, +_{\mathcal{P}_{0,\emptyset}}, \mathcal{E}, =_{\mathbb{N}}, =_{\mathcal{P}_{0,\emptyset}} \rangle$$

Lemma 9. *Let I, J be disjoint subsets such that $I \cup J = \{1, \ldots, n\}$. Then, for all integers $a_1, \ldots, a_n \in \mathbb{N}$ and sets $A_1, \ldots, A_n \in \mathcal{P}_{0,\emptyset}$,*

$$\langle \mathcal{P}(\mathbb{N}); +, = \rangle \models \sum_{i \in I} a_i + A_i = \sum_{j \in J} a_j + A_j \iff \mathcal{M} \models \psi(a_1, \ldots, a_n, A_1, \ldots, A_n)$$

where $\psi(x_1, \ldots, x_n, X_1, \ldots, X_n)$ is a Boolean combination of formulas $\mathcal{E}(X_\ell)$, for $\ell = 1, \ldots, n$, and equalities $\sum_{i \in I} x_i = \sum_{j \in J} x_j$ and $\sum_{i \in I} X_i = \sum_{j \in J} X_j$.

Proof. Consider the class $\mathcal{S}$ of solutions of equation $\sum_{i \in I} X_i = \sum_{j \in J} X_j$ in $\mathcal{P}(\mathbb{N})$. Then $\mathcal{S} = \mathcal{Z} \cup (\mathcal{S} \setminus \mathcal{Z})$ where

(i) $\mathcal{Z}$ is the class of n-tuples of sets satisfying $X_i = X_j = \emptyset$ for some $i \in I$ and $j \in J$.

(ii) $\mathcal{S} \setminus \mathcal{Z}$ is the class of n-tuples in $\mathcal{S}$ consisting of nonempty sets.

Since $a + \emptyset = \emptyset$ for all $a \in \mathbb{N}$, we have

$$\begin{aligned} \mathcal{Z} = \{(a_1 + A_1, \ldots, a_n + A_n) \mid a_1, \ldots, a_n \in \mathbb{N}, A_1, \ldots, A_n \in \mathcal{P}_{0,\emptyset}(\mathbb{N}), \\ A_i = A_j = \emptyset \text{ for some } i \in I \text{ and } j \in J\}\ . \end{aligned} \tag{2.8}$$

Let $\mathcal{S}_0 = \mathcal{S} \cap (\mathcal{P}_0(\mathbb{N}))^n$ and $\mathcal{R} = \{(a_1, \ldots, a_n) \in \mathbb{N}^n \mid \sum_{i \in I} a_i = \sum_{j \in J} a_j\}$. Observe that if $B_1, \ldots, B_n \in \mathcal{P}(\mathbb{N})$ are all nonempty and $a_i = \min B_i$, $A_i = B_i - a_i$ (i.e. $B_i = a_i + A_i$ with $A_i \in \mathcal{P}_0(\mathbb{N})$) for $i = 1, \ldots, n$, then equality $\sum_{i \in I} B_i = \sum_{j \in J} B_j$ holds if and only if both equalities $\sum_{i \in I} b_i = \sum_{j \in J} b_j$ and $\sum_{i \in I} A_i = \sum_{j \in J} A_j$ hold. Thus,

$$\mathcal{S} \setminus \mathcal{Z} = \{(a_1 + A_1, \ldots, a_n + A_n) \mid \overrightarrow{a} \in \mathcal{R}, \overrightarrow{A} \in \mathcal{S}_0\}\ . \tag{2.9}$$

Consider the following formulas (recall $\mathcal{E}(X)$ expresses $X=\emptyset$):

$$\begin{aligned}\psi_{\mathcal{Z}}(x_1,\ldots,x_n,X_1,\ldots,X_n) &\equiv \bigvee_{i\in I, j\in J} \mathcal{E}(X_i)\wedge\mathcal{E}(X_j)\\ \psi_{\mathcal{S}\setminus\mathcal{Z}}(x_1,\ldots,x_n,X_1,\ldots,X_n) &\equiv \neg\mathcal{E}(X_1)\ \wedge\ldots\wedge\ \neg\mathcal{E}(X_n)\\ &\quad\ \wedge \sum_{i\in I} x_i = \sum_{j\in J} x_j\ \wedge\ \sum_{i\in I} X_i = \sum_{j\in J} X_j\end{aligned}$$

Equalities (2.8) and (2.9) show that, for $a_1,\ldots,a_n\in\mathbb{N}$ and $A_1,\ldots,A_n\in\mathcal{P}_{0,\emptyset}(\mathbb{N})$,

$$\begin{aligned}&\langle\mathcal{P}(\mathbb{N});+,=\rangle \models \sum_{i\in I} a_i + A_i = \sum_{j\in J} a_j + A_j\\ &\qquad\Longleftrightarrow\ \mathcal{M}\models \psi_{\mathcal{Z}}(a_1,\ldots,a_n,A_1,\ldots,A_n)\ \vee\ \psi_{\mathcal{S}\setminus\mathcal{Z}}(a_1,\ldots,a_n,A_1,\ldots,A_n)\end{aligned}$$

We now lift Lemma 9 to formulas with quantifications over $\mathcal{P}(\mathbb{N})$.

Lemma 10. *For any Boolean combination $F(\overrightarrow{Y},\overrightarrow{X})$ of equalities between sums of the set variables $X_1,\ldots,X_n$ and $Y_1,\ldots,Y_k$, there exists a Boolean combination $\mathcal{T}(F)$ of*

(1) *equalities between sums of the set variables $U_1,\ldots,U_n,V_1,\ldots,V_k$,*
(2) *equalities between sums of the integer variables*
(3) *formulas $\mathcal{E}(X_1),\ldots,\mathcal{E}(X_n),\mathcal{E}(Y_1),\ldots,\mathcal{E}(Y_k)$,*

such that, for all integers $a_1,\ldots,a_n\in\mathbb{N}$, all sets $A_1,\ldots,A_n$ in $\mathcal{P}_{0,\emptyset}(\mathbb{N})$ (i.e. each A_i is either empty or contains 0), any sequence $Q_1,\ldots,Q_k$ of quantifiers $\exists$ or $\forall$,

$$\begin{aligned}&\langle\mathcal{P}(\mathbb{N});+,=\rangle \models Q_1Y_1\cdots Q_kY_k\ F(\overrightarrow{Y},a_1+A_1,\ldots,a_n+A_n)\\ &\qquad\Longleftrightarrow\ \mathcal{M}\models Q_1v_1\ Q_1V_1\cdots Q_kv_k\ Q_kV_k\ \mathcal{T}(F)(\overrightarrow{v},\overrightarrow{a},\overrightarrow{V},\overrightarrow{A})\end{aligned}\tag{2.10}$$

Proof. If E is an equation then Lemma 9 gives a formula ψ_E which is a convenient $\mathcal{T}(E)$. For a Boolean combination F of equations $E_1,\ldots,E_p$, let $T(F)$ be the same Boolean combination with $\psi_{E_1},\ldots,\psi_{E_p}$.

Having defined $\mathcal{T}(F)$, we now prove the Lemma by induction on k. The case $k=0$ (i.e. no prefix of quantifications) is clear from Lemma 9. Suppose (2.10) holds for the quantification prefix $Q_1\ldots Q_k$ and any Boolean combination F. Then, for $a_1,\ldots,a_n\in\mathbb{N}$, $A_1,\ldots,A_n$ in $\mathcal{P}_{0,\emptyset}(\mathbb{N})$ and $Q\in\{\exists,\forall\}$,

$$\begin{aligned}&\langle\mathcal{P}(\mathbb{N});+,=\rangle \models QZ\ Q_1Y_1\cdots Q_kY_k\ F(\overrightarrow{Y},Z,a_1+A_1,\ldots,a_n+A_n)\\ &\Longleftrightarrow\ Qb\in\mathbb{N}\ QB\in\mathcal{P}_{0,\emptyset}(\mathbb{N})\\ &\qquad\qquad \langle\mathcal{P}(\mathbb{N});+,=\rangle\models \overrightarrow{QY}\ F(\overrightarrow{Y},b+B,a_1+A_1,\ldots,a_n+A_n)\\ (\dagger)\quad&\Longleftrightarrow\ Qb\in\mathbb{N}\ QB\in\mathcal{P}_{0,\emptyset}(\mathbb{N})\\ &\qquad \mathcal{M}\models Q_1v_1\ Q_1V_1\cdots Q_kv_k\ Q_kV_k\ \mathcal{T}(F)(\overrightarrow{v},b,\overrightarrow{a},\overrightarrow{V},B,\overrightarrow{A})\\ &\Longleftrightarrow\ \mathcal{M}\models Qw\ QW\ Q_1v_1\ Q_1V_1\cdots Q_kv_k\ Q_kV_k\\ &\qquad\qquad\qquad\qquad \mathcal{T}(F)(\overrightarrow{v},w,\overrightarrow{a},\overrightarrow{V},W,\overrightarrow{A})\end{aligned}$$

where line $(\dagger)$ is obtained using the induction hypothesis.

Lemma 11 (Splitting lemma). *Consider two disjoint sets of variables* $z_1,\ldots,z_k,Z_1,\ldots,Z_k$ *and* $t_1,\ldots,t_n,T_1,\ldots,T_n$ *and let* $\Phi(\overrightarrow{t},\overrightarrow{T})$ *be a formula with* $2n$ *free variables of the form*

$$\Phi(\overrightarrow{t},\overrightarrow{T}) \equiv Q_k z_k Q_k Z_k \cdots\ Q_1 z_1 Q_1 Z_1\ \mathcal{B}(\overrightarrow{z},\overrightarrow{Z},\overrightarrow{t},\overrightarrow{T})$$

where the Q_i*'s are quantifiers in* $\{\exists,\forall\}$ *and where* $\mathcal{B}(\overrightarrow{z},\overrightarrow{Z},\overrightarrow{t},\overrightarrow{T})$ *is a Boolean combination of atomic formulas depending on the variables* $\overrightarrow{z},\overrightarrow{t}$ *only and atomic formulas depending on the variables* $\overrightarrow{Z},\overrightarrow{T}$ *only. Then there exists* $r \geq 1$ *and finitely many formulas* ϕ_ℓ, ψ_ℓ*, for* $\ell = 1,\ldots,r$ *each having n variables, such that* $\Phi(\overrightarrow{t},\overrightarrow{T})$ *is logically equivalent to*

$$\bigvee_{\ell=1,\ldots,r} \phi_\ell(\overrightarrow{t}) \wedge \psi_\ell(\overrightarrow{T})$$

Proof. We argue by induction on $k \in \mathbb{N}$. The initial case $k = 0$ is an instance of the classical disjunctive normal form. We now show the induction step. Suppose the Lemma is true for $k-1$ with $k \geq 1$. Then there exists $r \geq 1$ and $\phi_{k-1,\ell}(z_k,\overrightarrow{t})$, $\psi_{k-1,\ell}(Z_k,\overrightarrow{T})$, for $\ell = 1,\ldots,r$, such that

$$\begin{aligned}
&Q_{k-1}z_{k-1}Q_{k-1}Z_{k-1}\cdots Q_1 z_1 Q_1 Z_1\ \mathcal{B}(z_1,\ldots,z_{k-1},z_k Z_1,\ldots,Z_{k-1},Z_k,\overrightarrow{t},\overrightarrow{T})\\
&\qquad\qquad\iff \bigvee_{\ell=1,\ldots,r} \phi_{k-1,\ell}(z_k,\overrightarrow{t}) \wedge \psi_{k-1,\ell}(Z_k,\overrightarrow{T})
\end{aligned}$$

Then, in case Q_k is $\exists$,

$$\begin{aligned}
&\exists z_k \exists Z_k\ Q_{k-1}z_{k-1}Q_{k-1}Z_{k-1}\cdots Q_1 z_1 Q_1 Z_1\ \mathcal{B}(\overrightarrow{z},\overrightarrow{Z},\overrightarrow{t},\overrightarrow{T})\\
&\iff \exists z_k \exists Z_k\ \bigvee_{\ell=1,\ldots,r} \phi_{k-1,\ell}(z_k,\overrightarrow{t}) \wedge \psi_{k-1,\ell}(Z_k,\overrightarrow{T})\\
&\iff \bigvee_{\ell=1,\ldots,r} \left(\exists z_k \phi_{k-1,\ell}(z_k,\overrightarrow{t})\right) \wedge \left(\exists Z_k \psi_{k-1,\ell}(Z_k,\overrightarrow{T})\right)\\
&\iff \bigvee_{\ell=1,\ldots,r} \phi_{k,\ell}(\overrightarrow{t}) \wedge \psi_{k,\ell}(\overrightarrow{T})
\end{aligned}$$

where $\phi_{k,\ell}(\overrightarrow{t})$ is $\exists z_k \phi_{k-1,\ell}(\overrightarrow{t})$ and the same with $\psi_{k,\ell}(\overrightarrow{T})$. Finally, the case Q_k is $\forall$ is treated similarly by first converting from disjunctive to conjunctive form and, after distributing the $\forall$ quantifiers, converting back from conjunctive to disjunctive form.

We finally come to the wanted nondefinability result.

Theorem 10. *The predicate* $Y = \{0\} \cup X$ *is not definable in* $\langle \mathcal{P}(\mathbb{N});+,=\rangle$.

Proof. By way of contradiction, assume that the predicate $Y = \{0\} \cup X$ can be defined in $\langle \mathcal{P}(\mathbb{N});+,=\rangle$. There exists a Boolean combination $F(\overrightarrow{Z},X,Y)$ of equalities of sums of the sets Z_i and X,Y such that

$$\langle \mathcal{P}(\mathbb{N});+,=\rangle \models\ Y = \{0\} \cup X \iff Q_1 Z_1 \cdots Q_k Z_k\ F(\overrightarrow{Z},X,Y) \tag{2.11}$$

By Lemma 10, for all $a,b \in \mathbb{N}$ and sets $A,B \in \mathcal{P}_{0,\emptyset}(\mathbb{N})$,

$$\begin{aligned}\langle \mathcal{P}(\mathbb{N});+,=\rangle &\models A=\mathbb{N} \,\wedge\, b+B=\{0\}\cup(a+A)\\ &\iff \mathcal{M} \models Q_1v_1\, Q_1V_1 \cdots Q_kv_k\, Q_kV_k\; \mathcal{T}(F)(\overrightarrow{v},\overrightarrow{V},a,b,A,B) \qquad (2.12)\end{aligned}$$

where $\mathcal{T}(F)(\overrightarrow{v},\overrightarrow{V},u,v,U,V)$ is a Boolean combination of formulas with free variables among u,v and formulas with free variables among U,V. By Lemma 11, we get formulas $\varphi_\ell(u,v)$, $\psi_\ell(U,V)$, $\ell=1,\ldots,L$, such that

$$\begin{aligned}\langle \mathcal{P}(\mathbb{N});+,=\rangle &\models b+B=\{0\}\cup(a+A)\\ &\iff \mathcal{M} \models \bigwedge_{\ell=1,\ldots,L} \varphi_\ell(a,b)\wedge\psi_\ell(A,B) \qquad (2.13)\end{aligned}$$

Now, for each $n \geq 1$, we have $0+S_n=\{0\}\cup(n+\mathbb{N})$ hence there exists ℓ such that $\varphi_\ell(n,0)$ and $\psi_\ell(\mathbb{N},S_n)$ are true. Since there are finitely many such indices ℓ, there exists two values of n, say $p,q \geq 1$, $p \neq q$, with the same associated ℓ. In particular, $\varphi_\ell(p,0)\wedge\psi_\ell(\mathbb{N},S_q)$ is true yielding equality $p+\mathbb{N}=0+S_q$, contradicting the condition $p \neq q$.

2.5.2 Other Nondefinable Predicates

Theorem 10 implies the nondefinability of many other predicates. We select some of them in this section. In particular, we compare four ways to code integers by sets in definable classes:

`Final`	$=\{n+\mathbb{N} \mid n\geq 1\}$	(see Proposition 9)
`Single`	$=\{\{n\} \mid n\geq 1\}$	(see Proposition 12)
`Special`	$=\{\{0\}\cup n+\mathbb{N} \mid n\geq 1\}$	(see Theorem 3)
`Periodic`	$=\{n\mathbb{N} \mid n\geq 1\}$	(see Proposition 21)

Definition 8. 1. Given predicates $A_1,\ldots,A_k$ and B over $\mathcal{P}(\mathbb{N})$, we say that B is *definable from* $A_1,\ldots,A_k$ in $\langle \mathcal{P}(\mathbb{N});+,=\rangle$ if B is first-order definable in the structure $\langle \mathcal{P}(\mathbb{N});+,=,A_1,\ldots,A_k\rangle$. We then write $(A_1,\ldots,A_k) \rightsquigarrow B$.
2. A and B are *definable from each other* in $\langle \mathcal{P}(\mathbb{N});+,=\rangle$ if $A \rightsquigarrow B$ and $B \rightsquigarrow A$.

Theorem 11. *No predicate in Table 1 is definable in $\langle \mathcal{P}(\mathbb{N});+,=\rangle$. Moreover, any two of them are definable from each other in $\langle \mathcal{P}(\mathbb{N});+,=\rangle$.*

Open problem 12. *Is there a non definable predicate which is not definable from each other with the predicates in Table 1?*

Proof (Proof of Theorem 11). Together with the predicates `Final`, `Single`, `Special` and `Periodic` recalled supra, we also freely use some definable predicates such as $\min X \leq \min Y$ (cf. § 2.2.3.4) and

Table 2.1 Some predicates not definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$

$E(X,Y)$	Inclusion	$X \subseteq Y$
$F(X,Y)$	Adjoin 0	$Y = X \cup \{0\}$
$F_1(X,Y,Z)$	Union	$X \cup Y = Z$
$F_2(X,Y,Z)$	Intersection	$X \cap Y = Z$
$F_3(X,Y)$	Complement	$X = \mathbb{N} \setminus Y$
$F_4(X,Y)$	Star	$Y = X^*$
$G(X,Y)$	Coding interchange	$(X,Y) \in \{(n+\mathbb{N}, S_n) \mid n \geq 1\}$
$G_1(X,Y)$		$(X,Y) \in \{(\{n\}, S_n) \mid n \geq 1\}$
$G_2(X,Y)$		$(X,Y) \in \{(n+\mathbb{N}, n\mathbb{N}) \mid n \geq 1\}$
$G_3(X,Y)$		$(X,Y) \in \{(\{n\}, n\mathbb{N}) \mid n \geq 1\}$
$H(X,Y)$	Membership	$(X,Y) \in \{(n+\mathbb{N}, B) \mid n \in B\}$
$H_1(X,Y)$		$(X,Y) \in \{(\{n\}, B) \mid n \in B\}$
$H_2(X,Y)$		$(X,Y) \in \{(n\mathbb{N}, B) \mid n \in B\}$
$H_3(X,Y)$		$(X,Y) \in \{(S_n, B) \mid n \in B\}$
$H_4(X,Y)$		$(X,Y) \in \{(A,B) \mid \min A \in B\}$
$H_5(X,Y)$		$(X,Y) \in \{(A,B) \mid$ the 2^{d} elem. of A is in $B\}$
$J(X)$	Semigroup	X is a semigroup (i.e. $X + X \subseteq X$)
$J_1(X)$		$X \in \{an + n\mathbb{N} \mid n \geq 1, a \in \mathbb{N}\}$
$J_2(X)$		$X \in \{n + n\mathbb{N} \mid n \geq 1\}$

$$\begin{array}{ll} \theta(X,Y) = \{(A, S_n) \mid (\min A) + n \in A\} & \text{(cf. Lemma 1)} \\ \sigma(X,Y) \equiv X \text{ is a submonoid containing } Y \ \wedge\ 0 \in Y & \\ & \text{(cf. Propositions 9, 10, 13)} \end{array}$$

We also freely use definable constants and functions (cf. §2.2.3.1): $\emptyset$, $\{0\}$, $\mathbb{N}$, $X \mapsto \{\min X\}$ (defined for $X \neq \emptyset$) and

$$\begin{array}{llll} \mathtt{Succ} & = S_n \mapsto S_{n+1} & \text{for } n \geq 1 & \text{(cf. Proposition 16)} \\ S & = n\mathbb{N} \mapsto S_n & \text{for } n \geq 1 & \text{(cf. Proposition 22)} \\ \pi & = S_n \mapsto n\mathbb{N} & \text{for } n \geq 1 & \text{(cf. Proposition 22)} \\ S_{\max} & = A \mapsto S_{\max A} & \text{for initial segments } A \neq \{0\} & \text{(cf. Proposition 20)} \end{array}$$

The structure of the proof is as follows:

$$E \rightsquigarrow F \rightsquigarrow \begin{cases} F_4 \rightsquigarrow G \rightsquigarrow G_2 \rightsquigarrow G_3 \rightsquigarrow G_1 \rightsquigarrow G & \\ F_3 \rightsquigarrow G & G_1 \rightsquigarrow E \rightsquigarrow H \rightsquigarrow H_1 \rightsquigarrow G_1 \\ F_2 \rightsquigarrow E & (E, G_3) \rightsquigarrow H_2 \rightsquigarrow H_3 \rightsquigarrow G_1 \\ F_1 \rightsquigarrow E \rightsquigarrow J \rightsquigarrow J_1 \rightsquigarrow J_2 \rightsquigarrow G_3 & H_1 \rightsquigarrow H_4 \rightsquigarrow H_5 \rightsquigarrow H_3 \end{cases}$$

$E \rightsquigarrow F$. Use E to express that Y is the smallest set containing X and $\{0\}$.

$F \rightsquigarrow F_i$ for $i = 1,2,3$. For $i = 3$, we have to relate, for any n, condition $n \in X$ with condition $n \in Y$. The case $n = 0$ is treated apart whereas the case $n \geq 1$ can be treated with θ relatively to $X \cup \{0\}$ (which is obtained using F). The cases $i = 1,2$ are similar. Thus,

$$\begin{aligned}
F_3(X,Y) \quad &\Leftrightarrow (0\in X \Leftrightarrow 0 \notin Y) \wedge \forall X_0\, \forall Y_0\, \big(F(X,X_0)\wedge F(Y,Y_0) \\
&\qquad\qquad\qquad \Rightarrow \forall T\ (\mathtt{Special}(T) \Rightarrow (\theta(X_0,T) \Leftrightarrow \neg\theta(Y_0,T)))\big) \\
F_1(X,Y,Z) &\Leftrightarrow (0\in Z \Leftrightarrow (0\in X \vee 0\in Y)) \\
&\quad \wedge \forall X_0\, \forall Y_0\, \forall Z_0\, \big(F(X,X_0)\wedge F(Y,Y_0) \\
&\quad \Rightarrow \forall T\ (\mathtt{Special}(T) \Rightarrow (\theta(Z_0,T) \Leftrightarrow (\theta(X_0,T)\vee\theta(Y_0,T)))\big) \\
F_2(X,Y,Z) &\Leftrightarrow (0\in Z \Leftrightarrow (0\in X \wedge 0\in Y)) \\
&\quad \wedge \forall X_0\, \forall Y_0\, \forall Z_0\, \big(F(X,X_0)\wedge F(Y,Y_0) \\
&\quad \Rightarrow \forall T\ (\mathtt{Special}(T) \Rightarrow (\theta(Z_0,T) \Leftrightarrow (\theta(X_0,T)\wedge\theta(Y_0,T)))\big)
\end{aligned}$$

$F \rightsquigarrow F_4$. Express that X^* is the smallest submonoid containing $X\cup\{0\}$.

$$F_4(X,Y) \Leftrightarrow \exists X_0\ (F(X,X_0)\wedge \sigma(Y,X_0) \wedge \forall T\ (\sigma(T,X_0) \Leftrightarrow \sigma(T,Y)))$$

$F_1 \rightsquigarrow E$, $F_2 \rightsquigarrow E$. $E(X,Y) \Leftrightarrow F_1(X,Y,Y) \Leftrightarrow F_2(X,Y,X)$.

$F_3 \rightsquigarrow G$. Observe that if X is a final segment then its complement is an initial segment and apply Proposition 20:

$$\begin{aligned}
G(X,Y) \Leftrightarrow\ & \mathtt{Final}(X) \wedge (X = 1+\mathbb{N} \Rightarrow Y = \mathbb{N}) \\
&\qquad\qquad \wedge\ (X \neq 1+\mathbb{N} \Rightarrow \exists Z\ (F_3(X,Z)\wedge Y = \mathtt{Succ}(S_{\max}(Z))))
\end{aligned}$$

$F_4 \rightsquigarrow G$. $G(X,Y) \Leftrightarrow \mathtt{Final}(X) \wedge F_4(X,Y)$.

$G \rightsquigarrow G_2$. $G_2(X,Y) \Leftrightarrow \mathtt{Periodic}(Y) \wedge G(X,S(Y))$.

$G_2 \rightsquigarrow G_3$. $G_3(X,Y) \Leftrightarrow \mathtt{Single}(X) \wedge G_2(X+\mathbb{N},Y)$.

$G_3 \rightsquigarrow G_1$. $G_1(X,Y) \Leftrightarrow \mathtt{Periodic}(Y) \wedge \mathtt{Single}(X) \wedge G_3(X,S(Y))$

$G_1 \rightsquigarrow G$. $G(X,Y) \Leftrightarrow \mathtt{Final}(X) \wedge G_1(\min(X),Y)$.

$G_1 \rightsquigarrow E$. Observe that $X \subseteq Y$ if and only if three conditions hold: 1) $\min X \geq \min Y$ 2) $\min X \in Y$ and 3) for all $p,q \geq 1$, if $(\min X)+p = (\min Y)+q$ then $(\min X)+p \in X$ implies $(\min Y)+q \in Y$. Now, $\min X \in Y$ if and only if $\min X = \min Y$ or $(\min Y)+a \in Y$ with $a = \min X - \min Y \geq 1$. In the next formula $\widetilde{X}$, $\widetilde{Y}$ denote $S_{\min X}$, $S_{\min Y}$ and P denotes S_p ($p>0$) and Q denotes S_q (with $q = p+a$ and $p \geq 0$). Also line 3 expresses $\min X \in Y$ (in the sole necessary case where $\min X > \min Y$) and line 4 expresses that if $p,q \geq 1$ and $\min X + p = \min Y + q$ holds then $\min X + p \in X$ implies $\min Y + q \in Y$.

$$\begin{aligned}
E(X,Y) &\Leftrightarrow \min X \geq \min Y \wedge\ \exists\widetilde{X}\ \exists\widetilde{Y}\ \forall P\ \forall Q \\
&\quad \big(G_1(\{\min X\},\widetilde{X}) \wedge G_1(\{\min Y\},\widetilde{Y}) \wedge\ \mathtt{Special}(P) \wedge\ \mathtt{Special}(Q)\big) \\
&\quad \Rightarrow \big(((\min X > \min Y \wedge \widetilde{X} = \widetilde{Y} \oplus Q) \Rightarrow \theta(Y,Q))) \\
&\qquad \wedge \widetilde{X} \oplus P = \widetilde{Y} \oplus Q \Rightarrow (\theta(X,P) \Rightarrow \theta(Y,Q)\big)
\end{aligned}$$

$E \rightsquigarrow H$. $H(X,Y) \Leftrightarrow \texttt{Final}(X) \wedge E(\{\min X\},Y)$.

$H \rightsquigarrow H_1$. $H_1(X,Y) \Leftrightarrow \texttt{Single}(X) \wedge H(X+\mathbb{N},Y)$.

$H_1 \rightsquigarrow G_1$. $G_1(X,Y) \Leftrightarrow \texttt{Special}(Y) \wedge H_1(X,Y) \wedge \neg H_1(X,\texttt{Succ}(Y))$.

$(E,G_3) \rightsquigarrow H_2$. $H_2(X,Y) \Leftrightarrow \texttt{Periodic}(X) \wedge \exists Z\ (G_3(Z,X) \wedge E(Z,Y))$.

$H_2 \rightsquigarrow H_3$. $H_3(X,Y) \Leftrightarrow \texttt{Special}(X) \wedge H_2(\pi(X),Y)$.

$H_3 \rightsquigarrow G_1$. $G_1(X,Y) \Leftrightarrow \texttt{Single}(X) \wedge \texttt{Special}(Y) \wedge H_3(Y,X) \wedge \neg H_3(\texttt{Succ}(Y),X)$.

$H_1 \rightsquigarrow H_4$. $H_4(X,Y) \Leftrightarrow H_1(X+\mathbb{N},Y)$.

$H_4 \rightsquigarrow H_5$. $$\begin{aligned} H_5(X,Y) \Leftrightarrow\ & X \neq \emptyset \wedge \neg\texttt{Single}(X) \\ & \wedge \exists Z\ \big(\texttt{Single}(Z) \wedge (\min Z > \min X) \wedge H_4(Z,X) \wedge H_4(Z,Y) \\ & \wedge \forall T\ ((\texttt{Single}(T) \wedge \min X < \min T < \min Z) \Rightarrow \neg H_4(T,X))\big). \end{aligned}$$

$H_5 \rightsquigarrow H_3$. $H_3(X,Y) \Leftrightarrow \texttt{Special}(X) \wedge H_5(X,Y)$.

$E \rightsquigarrow J$. Observe that X is a semigroup if and only if $X+X \subseteq X$.

$J \rightsquigarrow J_1$. Let $\mathcal{T} = \{a+n\mathbb{N} \mid a \in \mathbb{N}, n \geq 1\}$. Then $\mathcal{T}(X)$ if and only if $X = Y+Z$ for some Y,Z satisfying $\texttt{Single}(Y)$ and $\texttt{Periodic}(Z)$. Now, $J_1(X)$ if and only if $\mathcal{T}(X)$ and X is a semigroup: implication $\Rightarrow$ is obvious, conversely, if $X = a+n\mathbb{N}$ is a semigroup then its ultimate period is n. It divides all elements of X and in particular a and $J_1(X)$ holds.

$J_1 \rightsquigarrow J_2$. Observe that $J_2(X)$ if and only if $J_1(X)$ and $\min X \neq 0$ and $\{\min X\}$ is the unique singleton set such that $X = T+Y$ for some T,Y such that $T \neq \{0\}$, $\texttt{Single}(T)$ and $J_1(Y)$.

$J_2 \rightsquigarrow G_3$. $G_3(X,Y) \Leftrightarrow \exists Z\ (J_2(Z) \wedge X = \{\min Z\} \wedge Z = X+Y)$.

Out of the six pairs of the four codings of $\mathbb{N} \setminus \{0\}$ by $\texttt{Final}$, $\texttt{Single}$, $\texttt{Special}$ and $\texttt{Periodic}$, Table 2.1 tells that four conversions are incomparable with respect to $\rightsquigarrow$ (cf. predicates G and G_i, $i = 1,2,3$). In contrast the two remaining ones are definable as shown in the next result.

Proposition 23. *The following predicates are respectively Π_2 and Π_3.*

$$(X,Y) \in \{(n+\mathbb{N},\{n\}) \mid n \geq 1\} \quad , \quad (X,Y) \in \{(n\mathbb{N},S_n) \mid n \geq 1\}$$

Proof. Observe that $(X,Y) \in \{(n+\mathbb{N},\{n\}) \mid n \geq 1\}$ if and only if $0 \notin Y$and Y is a singleton set and $Y+\mathbb{N} = X$. Expressing the last equality as $\forall Z\ (Z = \mathbb{N} \Rightarrow Y+Z = X)$ Propositions 9 and 12 give the Π_2 complexity.

For the second predicate see Proposition 22.

2.6 Logical Definability in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$

2.6.1 Families of Sets All Containing 0

A simple application of Theorem 6 proves the following result.

Theorem 13. *Suppose $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ is a class of sets all containing* 0*. Then $\mathcal{F}$ is definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ if and only if it is definable in second-order arithmetic.*

Proof. Observe that $\mathcal{F}$ is definable in second-order arithmetic if and only if so is $\{A \mid \{0\} \cup (1+A) \in \mathcal{F}\}$ and apply Theorem 6.

Corollary 2. *Suppose $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ is a class of sets all containing an integer in $\{0, \ldots, n\}$. Then $\mathcal{F}$ is definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ if and only if it is definable in second-order arithmetic.*

Proof. The $\Rightarrow$ implication is trivial. Conversely, suppose $\mathcal{F}$ is definable in second-order arithmetic. For $i = 1, \ldots, n$, let $\mathcal{F}_i = \mathcal{F} \cap \{X \mid \min X = i\}$ and $\mathcal{G}_i = \{X - \min X \mid X \in \mathcal{F}_i\}$. Then the formulas $\mathcal{G}_i$ are also definable in second-order arithmetic. Since all sets in the formulas $\mathcal{G}_i$ contain 0, these formulas $\mathcal{G}_i$ are definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ (use Theorem 13). Then so are the sets $\{i + X \mid X \in \mathcal{G}_i\} = \mathcal{F}_i$ hence also their union which is $\mathcal{F}$.

2.6.2 Families of Sets Invariant by Translation

There is yet another application of Theorem 6 which extends the class of subsets definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$.

Theorem 14. *Suppose $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ is a class of subsets such that for all subsets $A \subseteq \mathbb{N}$ and all integers $a \in \mathbb{N}$ it holds*

$$A \in \mathcal{F} \Leftrightarrow A + a \in \mathcal{F}$$

Then $\mathcal{F}$ is definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ if and only if it is definable in second-order arithmetic.

Proof. Let $\mathcal{F}_0$ be the subclass of subsets in $\mathcal{F}$ containing 0. For all $B \in \mathcal{F}$ the subset $B - \min B$ is in $\mathcal{F}_0$. Clearly $\mathcal{F}_0$ is definable in second-order arithmetic, thus in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ by a formula $\phi(X)$. Then $\mathcal{F}$ is definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ by the formula

$$\exists X\ \exists Y\ (\phi(X) \wedge \mathrm{Sing}(Y) \wedge Z = Y + X)$$

Corollary 3. *The following classes of subsets are definable*

(i) $\{A \subseteq \mathbb{N} \mid A \textit{ is finite}\}$*,*

(ii) $\{A \subseteq \mathbb{N} \mid A \textit{ is cofinite}\}$*,*

(iii) $\{A \subseteq \mathbb{N} \mid A \textit{ is regular by a finite automaton}\}$*).*

Proof. Assertions (i) and (ii) are clear. Concerning assertion (iii) recall (cf. Proposition 4) that a subset $A \subseteq \mathbb{N}$ is recognizable by a finite automaton if and only if it is a finite union of subsets of the form $a + b\mathbb{N}$ with $a, b \in \mathbb{N}$. Thus, this class satisfies the conditions of Theorem 14.

2.6.3 Definable Sets of Integers

Theorem 15. *Let $A \in \mathcal{P}(\mathbb{N})$. The following conditions are equivalent:*

(1) *As a set of integers, A is definable in second-order arithmetic,*

(2) *As a class of sets, $\{A\}$ is definable in second-order arithmetic,*

(3) *$\{A\}$ is definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$.*

Proof. (1) $\Leftrightarrow$ (2) is straightforward. (3) $\Rightarrow$ (2) is obvious.
(2) $\Rightarrow$ (3). Let $a = \min A$. If $\{A\}$ is definable in second-order arithmetic then so is $\{A - a\}$. By Theorem 13 $\{A - a\}$ is definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$ (since $0 \in A - a$) hence so is $\{A\} = \{X + a \mid X \in \{A - a\}\}$.

2.6.4 Definability with an Extra Predicate

Theorem 16. *Let $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ and A be any predicate in Table 1. Then $\mathcal{F}$ is definable in second-order arithmetic if and only if $\mathcal{F}$ is definable in the structure $\langle \mathcal{P}(\mathbb{N}); +, A, = \rangle$.*

Proof. Implication $\Leftarrow$ is obvious since all predicates in Table 1 are definable in second-order arithmetic. Conversely, suppose $\mathcal{F}$ is definable in second-order arithmetic. Then so are the classes

$$\mathcal{F}^+ = \mathcal{F} \cap \{A \mid 0 \in A\}\ ,\ \mathcal{F}^- = \mathcal{F} \cap \{A \mid 0 \notin A\}\ ,\ \mathcal{H} = \{\{0\} \cup A \mid A \in \mathcal{F}^-\}$$

By Theorem 13 $\mathcal{F}^+$ and $\mathcal{H}$ are definable in $\langle \mathcal{P}(\mathbb{N}); +, = \rangle$. Using the predicate $F(X,Y)$ (which insures $Y = \{0\} \cup X$) one can then define $\mathcal{F}^-$ from $\mathcal{H}$. Finally, $\mathcal{F} = \mathcal{F}^+ \cup \mathcal{F}^-$ is definable in $\langle \mathcal{P}(\mathbb{N}); +, F, = \rangle$. Since all predicates in Table 1 are definable from each other, $\mathcal{F} = \mathcal{F}^+ \cup \mathcal{F}^-$ is also definable in $\langle \mathcal{P}(\mathbb{N}); +, A, = \rangle$ for any A in Table 1.

2.7 Remarkable Definable Sets and Classes

2.7.1 Operations on Sets with Close Minimum Elements

General set-theoretical operations are not definable. Here we show sufficient conditions for some of these operations to be definable.

Proposition 24. *The predicate $\min X = \min Y \ \wedge\ \phi(X,Y,Z)$ is Π_4 when $\phi(X,Y,Z)$ is either $X \cup Y = Z$ or $X \cap Y = Z$ or $X \subseteq Y$.*

Proof. We argue for union. The $\min X = \min Y$ condition is Σ_1 (cf. Proposition 11) and, by Lemma 1, $X \cup Y = Z$ if and only if, for all $n \geq 1$,

$$Z + S_n = Z + S_{n+1} \iff ((X + S_n = X + S_{n+1}) \vee (Y + S_n = Y + S_{n+1}))$$

Theorem 3 and Proposition 16 yield the stated logical complexity.

Proposition 25. *1. The predicate* $|\min X - \min Y| \leq k \ \wedge\ \phi(X,Y,Z)$ *is* Π_4 *when the integer* $k \geq 1$ *is fixed and* $\phi(X,Y,Z)$ *is either* $X \cup Y = Z$ *or* $X = \mathbb{N} \setminus Y$ *or* $X \subseteq Y$.
2. The following predicate is Π_4 *when the integer* $k \in \mathbb{N}$ *is fixed:*

$$\max(|\min(X) - \min(Y)|, |\min X - \min Z|) \leq k \ \wedge\ X \cap Y = Z$$

Proof. Point 1. Since the condition $|\min X - \min Y| \leq k$ is a disjunction of conditions $\min X = \min Y + \ell$ with $|\ell| \leq k$, it suffices to prove that the predicate $\min X = \min Y + k \ \wedge\ \phi(X,Y,Z)$ is Π_5. We argue for union. The $\min(X) = \min(Y) + k$ condition is $\Sigma_2 \wedge \Pi_2$ (cf. Proposition 11). Assuming $\min(X) = \min(Y) + k$, Lemma 1 insures that $X \cup Y = Z$ if and only if

1) $\min Z = \min Y$,
2) $Z + S_n = Z + S_{n+1} \iff Y + S_n = Y + S_{n+1}$ for all $n \in \{1, \ldots, k-1\}$,
3) $Z + S_k = Z + S_{k+1}$,
4) for all $n \geq 1$,
$Z + S_{k+n} = Z + S_{k+n+1}$
$\iff (X + S_n = Z + S_{n+1} \ \vee\ Y + S_{k+n} = Y + S_{k+n+1})$.

Theorems 2, 3 and Proposition 16 yield the stated logical complexity. The proof of Point 2 is similar.

Remark 4. Observe that the intersection is not definable even when the minima of the two subsets differ by 1 : we have to also bound the difference between $\min(X \cap Y)$ and $\min X$. For instance, $\{(n + \mathbb{N}, S_n) \mid n \geq 2\}$ which is $G(X,Y)$ of Theorem 11 is expressed as follows

$$\exists Z\ (\texttt{Special}(Z) \wedge Y = \partial Z \wedge (X = Y \cap (Z + 1)))$$

Indeed, if $Z = S_n$ then Y and $1 + Z$ have close minimum elements since $\min(1 + Z) = 1$ and $\min Y = 0$ but $\min(Y \cap (1 + Z)) = n + 1$ is not close to $\min Y$.

2.7.2 Fixed Submonoids

Proposition 11 can be generalized as follows.

Lemma 12. *Let* X *be a subset with minimum element equal to* m *and let* $a_1 < \ldots < a_n$ *be elements of* $\mathbb{N}$. *The following predicates* $\phi_m(X)$ *and* $\psi_m(X)$ *are* Δ_2 *:*

$\phi_m(X)$: $\min X = m$ *and* $a_1, \ldots, a_n > m$ *belong to* X
$\psi_m(X)$: $\min X = m$ *and* $a_1, \ldots, a_n > m$ *do not belong to* X

Proof. Using Lemma 1, we have:

$$\phi_m(X) \equiv \min X = m \wedge \bigwedge_{1\le i\le n} X + S_{a_i-m} = X + S_{a_i-m+1}$$
$$\psi_m(X) \equiv \min X = m \wedge \bigwedge_{1\le i\le n} X + S_{a_i-m} \neq X + S_{a_i-m+1}$$

The Δ_2 complexity is a corollary of Theorem 2 and Proposition 8.

Theorem 17. *Let M be a submonoid. The predicate $X = M$ is definable and its complexity is as follows*
1. Case $M = \{0\}$. The predicate $X = M$ is Π_1.
2. Case $M = \mathbb{N}$. The predicate $X = M$ is $\Sigma_1 \wedge \Pi_1$.
3. Case $M = \{0\} \cup (a + \mathbb{N}) = S_a$ with $a \ge 2$. The predicate $X = M$ is Δ_2.
4. For the general case the predicate $X = M$ is Π_2.

Proof. Claims 1, 2: cf. Proposition 9. Claim 3: cf. Theorem 2.
4. Let $G = \{g_1,\dots,g_n\}$ be the minimum generating set for M which we assume ordered. Then M is the smallest submonoid containing G. If $\phi_0(X)$ is the formula of Lemma 12 with $g_1,\dots,g_n$ in place of $a_1,\dots,a_n$ the predicate $X = M$ is equivalent to

$$\phi_0(X) \wedge X + X = X \wedge \forall Y \ \big((\phi_0(Y) \wedge (Y + Y = Y)) \Rightarrow X + Y = Y\big)$$

expressing that X is the smallest submonoid containing G. The Π_2 complexity comes from Lemma 12.

2.7.3 *Regular Subsets of* $\mathbb{N}$

Here we give a precise estimate of the structural complexity of some subsets and classes of subsets of particular importance, the definability of which is a consequence of Theorem 14.

2.7.3.1 Fixed Regular Subsets of $\mathbb{N}$

Theorem 18. *If $R \subseteq \mathbb{N}$ is regular then the predicate $X = R$ is Π_4. In case $R = \emptyset$ or $R = \{0\}$ it is Π_1. In case R is a singleton different from 0 it is $\Sigma_2 \wedge \Pi_2$.*

Proof. By Proposition 4, $R = A \cup (B + p\mathbb{N})$ with $a, p \in \mathbb{N}$ and $\emptyset \neq A \subseteq [0,a[$ and $B \subseteq [a, a+p[$. First, we introduce some formulas. We set $m = \min A$ and $b = a + p$ with the convention $b = a$ when B is empty. The following Σ_4 and Π_4 predicates tell which elements in the initial interval $[0,b[$ belong to the set and which do not.

$$\begin{aligned}F^{\exists}(X) \equiv\ & m = \min X \ \wedge\ \exists Y_1,\dots,Y_{b-m-1}Z_1,\dots,Z_{b-m-1} \\ & \bigwedge_{i\in A\cup B\setminus\{m\}} (Y_i = S_{i-m} \ \wedge\ \mathtt{Succ}(Y_i,Z_i) \ \wedge\ X+Y_i = X+Z_i) \\ & \wedge \bigwedge_{i\in\{m+1,\dots,b-1\}\setminus(A\cup B)} (Y_i = S_{i-m} \ \wedge\ \mathtt{Succ}(Y_i,Z_i) \ \wedge\ X+Y_i \neq X+Z_i)\end{aligned}$$

$$\begin{aligned}F^{\forall}(X) \equiv\ & m = \min X \ \wedge\ \forall Y,Z \\ & \bigwedge_{i\in A\cup B\setminus\{m\}} ((Y = S_{i-m} \ \wedge\ \mathtt{Succ}(Y,Z)) \Rightarrow X+Y = X+Z) \\ & \wedge \bigwedge_{i\in\{m+1,\dots,b-1\}\setminus(A\cup B)} ((Y = S_{i-m} \ \wedge\ \mathtt{Succ}(Y,Z)) \Rightarrow X+Y \neq X+Z)\end{aligned}$$

If the subset is finite, it suffices to express the fact that it does not contain any integer greater than or equal to a. This leads to the Π_4 formula

$$\begin{aligned}G(X) \equiv\ & \forall Y,Z,T\ \big((T = S_{a-m} \wedge \mathtt{Succ}(Y,Z) \ \wedge\ Y+T = T) \\ & \Rightarrow X+Y \neq X+Z\big)\end{aligned}$$

Thus when R is finite it is expressed by the predicate $F^{\forall}(X) \wedge G(X)$.

When the subset is infinite, i.e., when $B \neq \emptyset$ we must say that the subset of R consisting of all elements greater than or equal to a is periodic of period p, equivalently for all $x \geq a$ we have $x \in R \Leftrightarrow x+p \in R$ which is expressed by the Π_4 formula

$$\begin{aligned}H(X) \equiv\ & \forall T,Y,Y',Z,Z' : \\ & T = S_{a-m} \wedge \mathtt{Succ}(Y,Y') \wedge \mathtt{Succ}(Z,Z') \wedge \mathtt{Succ}_p(Y,Z) \\ & \Rightarrow (X+Y = X+Y' \Leftrightarrow X+Z = X+Z')\end{aligned}$$

Thus, when R is infinite it is expressed by the formula $F^{\forall}(X) \wedge H(X)$.
The remaining cases are a consequence of Proposition 12.

2.7.3.2 Finite and Cofinite Subsets of $\mathbb{N}$

Proposition 26. *The predicates "X is finite" and X is cofinite" are Σ_5.*

Proof. For the finiteness predicate, use Lemma 1 to express that $\min(X)+n \notin X$ for all large enough n :

$$\exists Y\ \big(\mathtt{Special}(Y) \ \wedge\ \forall Z,W\ ((Y+Z=Y \ \wedge\ \mathtt{Succ}(Z,W)) \Rightarrow X+Z \neq X+W\big)$$

For the cofiniteness predicate, express that $\min(X)+n \in X$ for all large enough n. The logical complexity is given by Theorem 3 and Proposition 16.

2.7.3.3 The Class of Regular Subsets of $\mathbb{N}$

Theorem 19. *The predicate "X is regular" is Σ_6.*

Proof. Observe that X is regular if and only if X is finite or is periodic with period $p \geq 1$. This latter assertion means that there exist two integers n and p such that all for all integers $x \geq n$ we have $x \in X$ if and only if $x + p \in X$. Using Theorems 4 and Proposition 26 this can be expressed as follows (where the variables N, P encode the above integers n and p and where the pairs of variables (V, V') and (W, W') respectively encode x and $x + p$):

$$\begin{aligned} X \text{ is finite } \vee\ & \exists N\ \exists P\ \forall V, W, V', W' \\ & \big(\mathtt{Special}(N) \wedge \mathtt{Special}(P) \wedge N + V = N \wedge V \oplus P = W \\ & \qquad \wedge\ \mathtt{Succ}(V, V') \wedge \mathtt{Succ}(W, W') \\ & \qquad\qquad \Longrightarrow (X + V = X + V' \Leftrightarrow X + W = X + W')\big) \end{aligned}$$

2.8 Conclusion

This paper proves the undecidability of the Σ_5 theory of additive monoid of subsets of $\mathbb{N}$. The decision problem for positive Σ_1 formulas (i.e. no negation) is trivially decidable since every equation is satisfied when all the variables are equal to the emptyset. What about the full Σ_1 theory? Care: we are looking at formulas in which the atomic subformulas are equations between variables (such as $XYXZ = ZXX$): no parameter is allowed. When regular sets are allowed as parameters then the decision problem for systems of equations becomes undecidable, cf. [6].

The question "What is definable and what is not definable in the additive monoid of subsets of $\mathbb{N}$?" is largely answered in this paper. Some definability results involve logically complex definitions: up to Σ_6 for the class of regular sets. Are such complex definitions optimal?

The additive monoid of subsets of $\mathbb{N}$ can be seen as the monoid of tally languages. What about the monoid of languages over an alphabet with at least two letters. This question is investigated in a forthcoming paper [**?**].

References

1. Choffrut, C., Grigorieff, S.: Logical theory of the monoid of languages over a non tally alphabet (in preparation)
2. Cohn, L.: On the submonoids of the additive group of integers, `http://www.macs.citadel.edu/cohnl/submonoids2002.pdf`
3. Fischer, M.J., Rabin, M.O.: Super-exponential complexity of presburger arithmetic. In: SIAM-AMS Symposium in Applied Mathematics, vol. 7, pp. 27–41 (1974)
4. García-Sánchez, P.A.: Numerical semigroups minicourse, `http://www.ugr.es/~pedro/minicurso-porto.pdf`
5. Ginsburg, S., Spanier, E.H.: Semigroups, Presburger formulas, and languages. Pacific J. Math. 16, 285–296 (1966)
6. İez, A., Okhotin, A.: Equations over sets of natural numbers with addition only. In: STACS, pp. 577–588 (2009)

7. Odifreddi, P.: Classical recursion theory. The theory of functions and sets of natural integers, vol. 1. North Holland (1989)
8. Ramaré, O.: On Shnirelman's constant 22(4), 645–706 (1995)
9. Ramírez-Alfonsín, J.L.: The Diophantine Frobenius Problem. Oxford University Press (2005)
10. Rogers, H.: Theory of recursive functions and effective computability. McGraw Hill (1967)
11. Rosales, J.C., García-Sánchez, P.A.: Numerical semigroups. Developments in Mathematics, vol. 20. Springer (2009)

Chapter 3
Some Reflections on Mathematics and Its Relation to Computer Science

Liesbeth De Mol

This paper resulted from a talk I gave at Machines, Computations and Universality 2013 in Zürich and I am very much indebted to the organizers and the participants of this conference for a very fruitful discussion. I am particularly grateful to Maurice Margenstern who, since he was a reader of my PhD, has given me several useful advices related to my work and has often motivated me for inquiring further into problems of decidability and undecidability in the context of tag systems, and more generally, for developing my thoughts on experimental mathematics and computer science

“*No paradigm should ever be allowed to dominate education*”

Benoît Mandelbrot, 2002

In [24, p. 325] Knuth recounts the following story:

> [At some time,] I wondered how to calculate the greatest common right divisor of two given matrices. A few days later I happened to be attending a conference where I met the mathematician H.B. Mann, and I felt that he would know how to solve this problem. I asked him, and he did indeed know the correct answer; but it was a mathematician’s answer, not a computer scientist’s answer! He said, “Let $\mathscr{R}$ be the ring of $n \times n$ integer matrices; in this ring, the sum of two principal left ideals is principal, so let D be such that
>
> $$\mathscr{R}A + \mathscr{R}B = \mathscr{R}D$$
>
> Then D is the greatest common right divisor of A and B.” This formula is certainly the simplest possible one, we need only eight symbols to write it down; and it relies on rigorously-proved theorems of mathematical algebra. But from the standpoint of a computer scientist, it is worthless.

Liesbeth De Mol
Centre Nationale de la Recherche Scientifique, UMR Savoirs,
Textes Langage, Université de Lille 3
e-mail: liesbeth.demol@univ-lille3.fr

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_3

Knuth used this story to explain how he considered mathematics and computer science to be distinct from each other: it illustrates very clearly the different ways by which mathematicians and computer scientists approach a given problem. Despite these differences in thinking, it is not uncommon for computer scientists to stress also the similarities between their discipline and mathematics. For instance, Dijkstra has argued that the methods of programming are mathematical in nature [14] and Wegner has emphasized on several occasions that computer science is "*in part a mathematical discipline*" [38].

Such reflections on the relation between mathematics and computer science usually take place in a computer science context since, today, most mathematicians do not really feel the need to define their discipline with respect to another. However, since the significance of the computer is increasing within mathematics, some mathematicians have also started to reflect on this issue. Most well-known at this time is probably the work by mathematicians like Borwein who has extensive experience with so-called computer-assisted experimental mathematics and who has argued on multiple occasions that "*[t]he computer is changing the way we are doing mathematics*" [3].

The aim of this paper is to revisit the historically-developed question on the nature of the relation between mathematics and computer science by (mainly) focusing on mathematical practices that involve the use of the computer. This will allow me to highlight *some* aspects of mathematics that are being affected by what I like to call a computer-science way of thinking. By doing so I not only want to place this "way of thinking" into a broader historical context of mathematics but also offer some reflections on the computer science discipline itself.

By considering this relation between mathematics and computer science, it is at some points almost unavoidable to represent things as if they were black and white whereas in real practice this is only rarely ever true. It is for this reason that I would like to recall here, as a kind of warning to the reader, the words following by Hamming who stated during his Turing award lecture [21]:

> We live in a world of shades of grey, but in order to argue, indeed even to think it is often necessary to dichotomize and say "black" or "white". Of course in doing so we do violence to the truth, but there seems to be no other way to proceed. I trust, therefore, that you will take many of my small distinctions in this light – in a sense, I do not believe them myself, but there seems to be no other simple way of discussing the matter.

3.1 Mathematical Logic, the Computer and Mathematics

Several decades before computer science was recognized as a discipline, mathematics was already inexorably tied up with what were to become some of the foundational sources for computer science. The papers by people like Church, Curry, Gödel, Hilbert, Kleene, Post and Turing are nowadays considered as fundamental sources of (theoretical) computer science but resulted in a context of reflections on the foundations of mathematics. Hilbert's formalistic and finitist program was one important school of thought within this foundational context.[1] He understood formalism and the method of finite axiomatization as a way to tackle several foundational problems of mathematics, viz., problems about *all* possible assertions in mathematics, most notably, consistency. He believed that by addressing such problems through finitist and formalist means, mathematics could be provided with a firm foundation (See for instance [31]).

One important conviction of Hilbert was that within mathematics, there is no Ignorabimus. Indeed, as Hilbert famously stated during a lecture in 1930 in Königsberg[2] (translated from [23, p. 963]):[3]

> The true reason why Comte could not find an unsolvable problem, lies in my opinion in the assertion that there exists no unsolvable problem. Instead of the stupid Ignorabimus, our solution should be: We must know. We shall know.

Others however were less positive about the prospect of having a mechanical solution to decide any mathematical proposition. As Von Neumann stated in 1927 (quoted from [16]):

> [T]he contemporary practice of mathematics, using as it does heuristic methods, only makes sense because of this undecidability. When the undecidability fails then mathematics, as we now understand it, will cease to exist; in its place there will be a mechanical prescription for deciding whether a given sentence is provable or not.

As we all know now, Hilbert was in fact too optimistic: in 1930-31 Gödel proved the limitations of the method of finite axiomatization and formalization through his incompleteness theorems and in 1936 Church and Turing independently proved the undecidability of the decision problem for first-order logic.[4] These results are often interpreted as the death-knell to Hilbert's finitist and formalist program. They certainly were for his dream of a mathematics without Ignorabimus!

Ironically, it were exactly the different formalist devices and techniques used and/or developed as tools to obtain such impossibility results, that would turn out to be very useful instruments for developing (some of) the theoretical foundations of and tools for the machine that *can* be seen as thé formalist device per se, viz. the computer. The computer is not only finite per definition but its actions are those of the cliché formalist practice, viz. the blind manipulation of meaningless symbols according to some rules. It was exactly for this reason that people like Dijkstra made a plea on multiple occasions for the significance of formalism in the context of computer science [15]:

[1] One other such school is Brouwer's intuitionism which today surely also has an important impact on computer science by means of constructive type theory.

[2] Ironically, Gödel would announce his incompleteness results at the same meeting!

[3] Hilbert repeated his belief in the non-existence of Ignorabimus in mathematics on several occasions.

[4] It is less well-known that Emil Post had already obtained incompleteness and undecidability results in the early 20s in the context of so-called normal systems. These results were later published as [33].

> The manipulation of uninterpreted formulae is [...] a most familiar operation for the computing scientist: it is the one and only operation computers are very good at. [...] The manipulation of uninterpreted formulae requires unambiguous formalisms [...] Our disappointing experiences with formality should not be interpreted as something being wrong with formality; the experiences were disappointing because we were incompetent amateurs. But this has been changed by our exposure to computing.

Today, the idea to use formalist techniques as a tool has become part of the standard practice within computer science. A leading thought is that of controlling problems of unreliability, unpredictability and complexity which frequently occur in the context of computing and is rooted in a belief that, to quote Dijkstra again, "*[m]astery of the reaction of the computer must not only be a theoretical possibility but a real, practical one*" [13]. As such, formal verification, denotational semantics, Chomsky grammars etc are today used within (the development of) programming languages and compilers in order to deal with problems of error, non-termination, security, etc. Also within mathematics, this formal approach is applied: proof assistants like Coq, which help to formally specify and verify mathematical proofs, have been used or are being used to formalize contested mathematical results, like the four-color theorem or the sphere packing problem. Such formalized proofs allow to control and verify that the steps taken by the machine are correctly executed and indeed result in a given theorem.

However, such practical realizations of formalism(s) are just one aspect of the computer and its surrounding practices. There is also a "non-rigorous" side which relies on experimentation, statistics, etc. to deal with problems of unpredictability, unreliability and complexity on the level of hardware, software, the humans that rely on it and the problems that are studied with it. For instance, in compiler design, experiments have been executed to determine the optimal size of a hash table; within hardware design statistical methods are used to regulate the cache memory; during programming, the debugging and testing of code is often preferred over formal verification, etc. The interplay between the different levels involved in computing is often too complex to permit for a feasible (complex of) formal method(s) and is thus often bound to escape its formal control. This more "ugly" side of computing, is, in a certain sense, more in line with von Neumann's preference for a mathematical practice which relies on heuristics.

It is exactly within this historically developed mathematical practice that we find another fundamental connection between mathematics and computer science, viz. computations and algorithms as the means to execute them. Indeed, algorithms and computations have always been a part of mathematics. However, this significance of computation and algorithms within mathematics, evident though it may seem from our contemporary perspective, has not always been properly acknowledged. Indeed, many of us have been educated with a mathematics that is more about abstract and general structures and theories than about concretely computed objects. The reason for this is that for a long time, a majority of mathematicians simply preferred general and abstract results over particular computation-based results.[5] As is suggested in [9], the separation between "pure" mathematics, on the one hand, and computation-intensive mathematics on the other, became sharper over the course of the 19th century and the beginning of the 20th century. Interestingly enough, Hilbert played an important role in this trend of treating "computation" rather pejoratively: in an influential report on algebraic number theory, known as *Zahlbericht* and published in 1897, Hilbert favors a more conceptual approach over a more algorithmic approach and certainly preferred "pure" ideas over computation (Quoted from [9]):

> I have tried to avoid Kummer's elaborate computational machinery, so that here too Riemann's principle may be realized and the proof completed not by computations but purely by ideas.

This idea to obtain results purely by ideas rather than "computational machinery" is still quite popular within mathematics. However, with the rise of the computer one also sees a slow renaissance of computations and algorithms: so-called "experimental" or "explorative" mathematics is becoming more popular, curricula no longer shy away from discrete mathematics and a growing number of mathematicians is focusing on mathematics of computation. It is exactly this "style" of mathematics that von Neumann, who himself was for a long time a strong supporter of the formalist school of thought (See e.g. [35]), promoted and practiced in the last 10 to 15 years of his life, viz. a mathematics that is rooted in computational results. In fact, von Neumann understood this possibility, which the new computing machines were offering to the mathematician, as a way to escape from a mathematics "*in danger of degeneration [after] much 'abstract inbreeding'*" [36].

As becomes clear through this account, the computer and with it, computer science, incarnates (at least) two different "styles" or "approaches": on one side of the spectrum one finds a more Hilbertian style which we can associate with abstract, elegant, rigorous and formal approaches on computation, on the other side, we find a late-von Neumann style which is usually considered more ugly, computation-intensive and more experimental. Evidently, in the context of computer science, both approaches are strongly connected through the computer and its computations. As such, they cannot be strictly separated from one another. Both can also be traced within the history of mathematics, even though the former seems to express the current more dominant view on mathematics.

Today, a growing community of mathematicians is embracing the computer to advance their work and, with it, the so-conceived less elegant style of doing mathematics. Indeed, despite the fact that the computer *is*, from a certain point of view, a formalist device, its effect on mathematics proper[6] apparently lies exactly in the opposite of being formal. So what exactly is it with the computer which encourages this style of mathematics and how does it relate to more formalist approaches within computer science? More generally, how is the computer affecting mathematics?

[5] This does not necessarily mean that for some period in the history of mathematics, computations and algorithms were no longer used or developed. Rather it means that they were not explicitly a part of the general mathematical discourse.

[6] Viz., not a mathematically-oriented computer science.

3.2 A Number-Theorist's Point of View

What is the impact of the computer on mathematics? An important source of inspiration for my own work on this question is Derrick H. Lehmer: he was one of the first mathematicians, a number theorist, to use a computer for doing mathematics and he has written on several occasions on the potential of extensive computation for mathematics. He identified two schools of thought within mathematics [29, p. 745]:

> The most popular school now-a-days favors the extension of existing methods of proof to more general situations. This procedure tends to weaken hypothesis rather than to strengthen conclusions. It favors the proliferation of existence theorems and is psychologically comforting in that one is less likely to run across theorems one cannot prove. Under this regime mathematics would become an expanding universe of generality and abstraction, spreading out over a multi-dimensional featureless landscape in which every stone becomes a nugget by definition. Fortunately, there is a second school of thought. This school favors *exploration* [m.i.] as a means of discovery. [B]y more or less elaborate expeditions into the dark mathematical world one sometimes glimpses outlines of what appear to be mountains and one tries to beat a new path in their direction. [N]ew methods, not old ones are needed, but are wanting. Besides the frequent lack of success, the exploration procedure has other difficulties. One of these is distraction. One can find a small world of its own under every overturned stone.

Lehmer clearly had a preference for the more "experimental" school of thought: having been raised by Derrick N. Lehmer, also a number theorist and well-known table-maker of prime numbers and factors, he was convinced of the experimental nature of mathematics and especially number theory. As he explains himself: "*My father did many things to make me realize at an early age that mathematics, and especially number theory, is an experimental science.*" Such experimental approach usually requires exploration and hence also something to explore. It is thus not surprising that Lehmer – and with him several other mathematicians – regard(ed) the computer as the perfect partner to support the "exploratory-minded". Indeed, the increase of "several orders of magnitude" in speed and memory combined with the capability of the machine to deal with combinatorial complexities on the executional level of a program, makes it possible "*to explore the terrain that has been staked out so freely and that something worth proving will be discovered in the rapidly expanding universe of mathematics*" [28, p. 146].

But what kind of mathematics is such "explorative" computer-assisted mathematics? Let us first look at a simple example and then relate it to Lehmer's views. The example comes from the early history of computer-assisted mathematics and concerns the study of the decimal expansion of π and e on ENIAC, one of the first electronic and (externally) programmable machines.[7] It is rather well-known that one of the applications of ENIAC was the computation of multiplication rates of

[7] ENIAC in its initial form was not a stored-program computer. Instead one had to manually wire the machine for each computation. Nonetheless, it was Turing complete in the same sense as modern machines are, viz., making abstraction from its memory limitations, it could simulate any Turing machine (amongst others, it had a conditional). For a detailed example of a program that was actually ran on ENIAC see [6].

neutrons in fission devices to estimate the size of the device for triggering fusion in an H-bomb. One of the people involved with this project was John von Neumann. He was convinced that a more experimental approach was the most suitable for studying this problem and it was decided to let ENIAC "numerically simulate", amongst others, the multiplication rates. It was in this context that the now so widely used but rarely questioned Monte Carlo method was introduced by Ulam. Of course, in order to compute with the Monte Carlo method one needs a source of random numbers. The most obvious method at the time was to use numbers generated by some other device (e.g. a roulette wheel) but, since electronic memory was limited at the time, these numbers had to be fed mechanically to the machine and significantly slowed down the computational process. Von Neumann and others started to reflect on the possibility of letting the machine generate its own random numbers. This resulted in the construction of so-called pseudo-random generators. However, how should one construct an algorithm for generating random numbers? Isn't it paradoxical to generate randomness through deterministic procedures? It is against this background that one should understand the computation of 2,000 digits of π and e by ENIAC: the aim was to investigate the statistical distribution of these numbers.[8] ENIAC was used to compute these digits and then a team of human computers was used to perform the statistical analysis. Also other methods were studied, including von Neumann's middle square method[9] – which was eventually used – and the quadratic iterator ($x_i = ax(1 - x_{i-1})$) which is now known as one of the paradigmatic cases of chaotic non-linear equations.

This example is a very simple but clear example of explorative mathematics: not only because it explicitly concerns the exploration of the digits of π and e but also because of its wider background: the search for and study of pseudo-random generators by exploring several possible instances. This act of exploration assumes several other activities. Amongst others, it necessitates the development of several special techniques: the development or selection of feasible approximating iterative algorithms that are fitted to the digital ENIAC; techniques to check for possible errors in the computation; etc.

This study of the decimal expansion of π and e was not new but in fact fits into a wider computational and explorative tradition: long before ENIAC, expansions of π and e were computed for several different reasons – as an exploration or test of good approximation algorithms, as a study of whether π is rational are not, etc.[10] In other words, even though the local context changes significantly, there is an almost natural bond between this type of explorative mathematics and (extensive) computation. Since the electronic computer is the machine which opens up the path of extensive computation, its usage in such explorative contexts is certainly not surprising.

[8] The exact nature of the distribution of these numbers is still unknown today even though it has been surmised more than once that they are normal.

[9] This method is known to be a very bad random generator in general. It functions as follows: pick a number with n digits, square it, resulting in a number of $2n$ digits, take its n middle digits, square it, etc.

[10] It was only in the 18th century that the irrationality of π was proven.

However, by using the computer, the mathematician also engages with a particular type of thinking summarized by Lehmer as follows [29]:

> I should like to speculate briefly on the overall impact of mechanization upon mathematics of the not too distant future. Mechanization tends to emphasize practice rather than theory, deeds rather than words, explicit answers rather than existence statements, definitions that are formalized rather than behavioristic, local rather than global phenomena, the limited rather than the infinite, the concrete rather than the abstract, and one could almost say, the scientific rather than the artistic [...] The computer is the instrument of our observatory, our window to the hard facts of the world of mathematics.

If one were to differentiate computer science from mathematics, the finite vs. the infinite, the concrete vs. the more abstract, etc are indeed the kind of typical differences one immediately comes up with. Even though abstraction from the machine, the development of formal and uniform methods and infinite models are an important aspect of the usual computer science practice, the object remains limited, discrete, local, concrete and practical. It is perhaps for this reason that, as Knuth explains, computer scientists are more familiar with non-uniformity and case-by-case analyses [27]:

> If I had to put my finger on the greatest difference between mathematicians and computer scientists, I would say that mathematicians have a strong preference for non-uniform rules, coupled with a strong dislike for case-by-case analysis; computer scientists, by contrast, are comfortable and fluent with highly non-uniform structures (like the different operations performed by real computers, or like the various steps in long and complex algorithms).

It is also this type of thinking that fits well with exploration: one needs to define local methods and a complex of subroutines that work in one concrete context, the methods need to be finite (one cannot "explore" the infinite as such, only its finite approximations) and one often needs to proceed on a case-by-case basis. In other words, what I claim here and what Lehmer claims is that some of the typical features of what one could tentatively call a computer science way of thinking has a common basis with that fraction of the mathematical discourse which is usually associated with exploration and computation. It is for this reason that, in view of a supposed increasing significance of the computer, Lehmer makes the following two possible predictions about the future of mathematics [29]:

> It would be a pleasure to predict that, as time goes on, the use of the instrument will become widespread and the nature of mathematics will slowly change from the dangerously unstable fluid art that it is apparently approaching today to a more and more structured and explicit science. There is an alternate prediction. Already we see, instead, a splitting from mathematics of a new branch commonly called computer science, which includes enough technology to frighten away your topologist or functional analyst. Soon disciplinary fences will be erected. It has been said that the invention of photography relieved the graphic artist of his obligation to depict nature and drove him into impressionism and finally to abstraction. This, it seems to me, is apt to be also the future of mathematics.

It is impossible at this point in history to verify which of these two predictions is the most correct one even though disciplinary fences *have* been built. However, what we can do is to partially verify the current tendencies within mathematics with respect to computation and exploration.

3.3 The Impact of the Computer on Mathematics: Some Quantitative Results

In order to find at least a heuristic guide to trace the impact the computer is having on mathematics, I studied the quantitative impact of the computer on mathematics. The method consists in measuring the frequency by which particular terms are being used within the mathematics discipline as represented in databases such as MathSciNet and Zentralblatt. This study is not completed yet, but the initial results at least give some indications for further research. This far, I focused on MathSciNet and three groups/clusters of terms: Group I consists of the words: comput*, calcula*, machine*; group II consists of: experiment*, heurist*, conject*, explor*, inspect* and, finally, group III consists of: Algorith*, program*, automat*, digital*, simulation*. Figs. 3.1–3.3 show a plot of the evolution of the frequency of these three groups of terms as used within title or review text of items listed in MathSciNet between 1940–2010.

These plots indicate that the significance of these three groups within mathematics increases as a function of time: by 2010, almost 10% of all items listed in MathSciNet explicitly refer to terms that are related to computers and computing in title or review text. Also the usage of terms that are often associated with exploration, terms such as experiment, conjecture, etc, show a steady increase with about 7% in

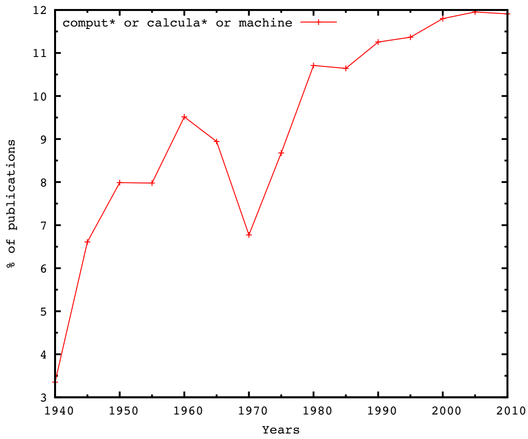


Fig. 3.1 Evolution of the frequency of the terms comput*, calcula*, machine* in MathSciNet 1940–2010

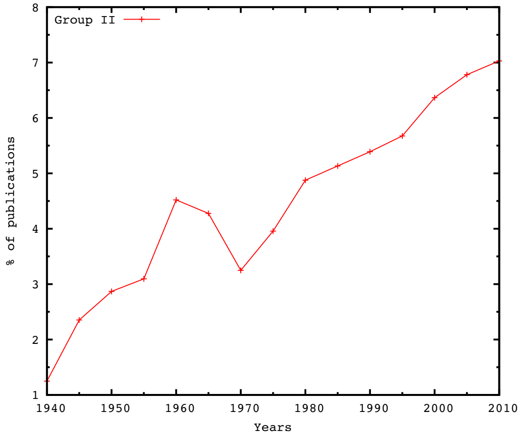


Fig. 3.2 Evolution of the frequency of the terms experiment*, heurist*, conject*, explor*, inspect* in MathSciNet 1940–2010

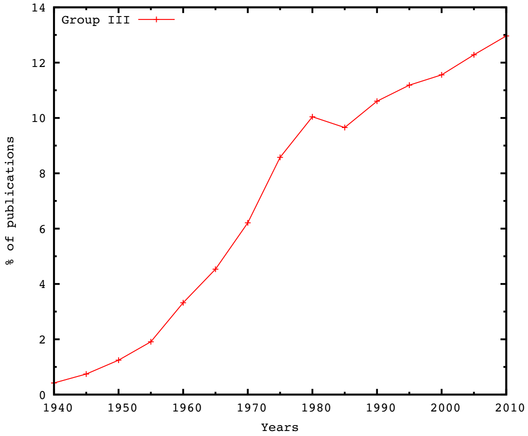


Fig. 3.3 Evolution of the frequency of the terms Algorith*, program*, automat*, digital*, simulation* in MathSciNet 1940–2010

2010. But most interestingly is the evolution of group III with no less than +/- 13% of the publications included in MathSciNet containing one of the terms of group III in its title or abstract by 2010. A more detailed analysis of the results also shows that within each of the three groups there are always one or two terms that take up most of the percentages. In group I, the terms starting with comput* and calcula* are the most dominant, with comput* steadily taking over from calcula*. For group II, the dominant terms are conjecture* and experiment*. Finally, for group III, it are algorith* and program* that are very dominant (by 2010, they take up about 9%) though one can observe a steady increase also in the usage of simulation* starting around 1980.

So what do these results indicate? Even though further analysis on the content or semantic level is required, combined with an extension to other databases, the results at the very least indicate that the use of terms that are intricately tied to computers (Groups I and III) show a significant increase since the early years of digital general-purpose computing. This increase cannot be reduced to the development of the computer science discipline alone: a more detailed study of the distribution of group I over the disciplines, using the MSC classification of disciplines, shows that for the computer science related disciplines like MSC 65 (Numerical Analysis) or MSC 68 (computer science) there is in fact a decrease in the significance of Group I which starts already around 1950 and then stabilizes around 1970. This decrease can be explained by a complementary increase in the usage of the terms algorith* and program* as is shown in Fig. 3.4.

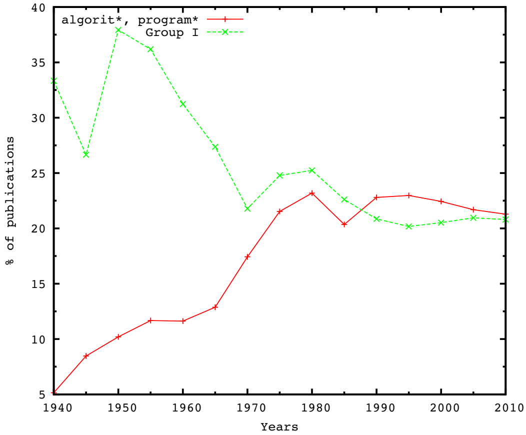


Fig. 3.4 Evolution of the frequency of the terms of group I versus that of algorith* and program* in the disciplines 65* (numerical analysis), 68* (computer science), 90* (operations research, mathematical programming), 93* (systems theory; control) and 94* (information and communication; circuits)

Whereas Groups I and III are quite naturally related to the computer, the results for Group II are not. Nonetheless it cannot be neglected that the usage of terms associated with "experimental" mathematics shows a steady increase which parallels the one for Group I.

These results are what they are: mere quantitative results which should certainly not be overinterpreted. However, for now they do serve the purpose of testing Lehmer's two alternative predictions in our time: even though it is indeed true that, in the meantime, disciplinary fences have been built there is a clear indication that computation, calculation, algorithms, programs, conjectures and experimentation have become more important over the years. However, these quantitative results do not allow us to *directly* tackle the question of the qualitative impact of the computer on mathematics but can at best serve as a heuristic guide to detect more carefully the impact of the computer on mathematics.

3.4 Computer-Assisted Explorative Mathematics: Characteristics and Problems

How is the computer changing mathematics, if at all? This far I have looked at this question mostly from the perspective of mathematics itself rather than from the computer, focusing first on the particular views of Derrick H. Lehmer and then at some more global developments we can detect within mathematics. But what is it exactly that the computer brings to mathematics besides extensive computational results to be explored? What particular characteristics are there to the computer that help to understand or clarify its (potential) effect on mathematics? In [12] I discuss three features which I consider as characteristic to computer-assisted explorative mathematics: human-computer interaction; the significance of time and processes and the internalization of mathematics into the machine.

3.4.1 Human-Computer Interaction

If there is one characteristic which is typical of computer-assisted explorative mathematics, it is the interaction with the computer. Of course, the history of mathematics is full of interactions between mathematicians and non-human instruments[11]. The most frequently used is the pencil-and-paper method: one writes and develops a notation in order to develop some technique, in order to communicate a result, in order to make a computation etc. This writing act always involves a "coding" practice – the use of symbols, of drawings, of abbreviations etc. Also within computer-assisted explorative mathematics coding practices are involved. However, whereas in the former case, the interpreter is always a human, the interpreter in the latter case is a non-human, viz., the computer. This affects the coding practice and hence also the interaction. This practice requires new specialized types of knowledge and skills which take into account the fact that one is communicating with a non-human. First of all, one needs (a) suitable "language(s)" for the two-way communication. It is exactly for this reason that we have seen the development of specialized software packages like Mathematica or Maple. It is also for this reason that extensive use is being made from visualization techniques that allow the mathematician to "digest" the vast amount of data generated by the computer. Secondly, and perhaps more importantly, one needs a good "format" to talk with the machine, viz. one needs good algorithms that are adapted to what the machine is good at *and* what it is bad at. I already provided the example of the development of pseudo-random generators, a type of algorithms that was never really considered before in the history of mathematics. Lehmer talks in this context about the need for an "idiot" approach: one needs algorithms that are executable by an "idiot" who cannot rely, for instance, on educated guesses and hence needs to be given instructions for every possible "behavior". One can, for instance, think of the SRT algorithm for division: it replaces

[11] Before the first computers, machines were already used within (applied) mathematics (e.g. Hartree's differential analyzer) and the practice of computer-assisted mathematics certainly has some of its roots in these practices. I will not consider such machines here.

the educated guessing we rely on for long division by a numerical table. Hartree, another computer pioneer, understood this new way of developing algorithms for the idiot as a real challenge for the future [22]:

> [I]n programming a problem for the machine, it is necessary to take a "machine's-eye view" of the operating instructions, that is to look at them from the point of view of the machine which can only follow them literally, without introducing anything not expressed explicitly by them, and try to foresee all the unexpected things that might occur in the course of the calculation, and to provide the machine with the means of identifying each one and with appropriate operating instructions in each case. And this is not so easy as it sounds; it is quite difficult to put oneself in the position of doing without *any* of the hints which intelligence and experience would suggest to a human computer in such situations.

Today, part of these problems are already dealt with through developments in programming. For instance, the compiler allows to detect a whole range of syntactical and semantical errors by means of parsers, the symbol table and the semantic analyzer. Viz., part of these problems have already found a formal solution. However, some of them haven't and require the development of special more heuristic and statistical techniques. For instance, on the hardware level, it are the laws of statistics that govern part of how the memory is cached.

This requirement to look at a problem also from the machine's eye has resulted in important progress both within computer science and mathematics. In computer science, for instance, the development of compiler techniques allow to help the programmer to specify his algorithm in the machine's language. Within mathematics, an obvious example are the advances being made within numerical analysis since the rise of electronic computing: it fills the need for good and efficient iterative approximation techniques that "work" for the machine. For instance, in the 50s, when electronic memory was still very limited, new iterative algorithms were being used that allow to reduce the size of the memory needed during computation (see e.g. [8]).

3.4.2 Internalization

In the early years of digital general-purpose computing there were two major bottlenecks. The first was the programming bottleneck: even though computation speed was increased with several orders of magnitude, the lack of programming languages and compilers meant that the setting up of a problem could take several days if not weeks. The second bottleneck was the memory bottleneck: electronic memory was very limited so one had to rely mostly on external memory devices like the punched cards. If additional memory was required during computation, this reliance on mechanical memory seriously slowed down the computation. Viz., the speed of memory storage and retrieval wasn't adapted at all to that of the computation.[12] As these two bottlenecks got steadily and partially resolved, it became possible to store internally into the machine more and more subroutines and the process of the so-called "algorithmization" of knowledge could be started. Also within mathematics one can

[12] Today the speed of read and write operations is still a major topic in hardware design.

observe a steady "algorithmization" of mathematical knowledge and skills. Some famous examples are the Zeilberger-Gosper algorithm and the PSQL algorithm. The result of this is that today we have huge libraries of mathematical subroutines at our availability where algorithms like the Zeilberger-Gosper algorithm, can be called by their name.

This steady process of internalization evidently goes hand-in-hand with human-computer interaction: since more and more processes are assigned to the machine rather than to the human, the boundaries between what is done by the "idiot" and what is done by the "intellect" are more and more blurred. For instance, in the early years of digital general-purpose computing the machine was concerned only with brute-force mathematical calculations and the human did most of the exploration. Nowadays, part of this exploration is internalized into the machine. Indeed, going back to our simple example of the ENIAC computations of the digits of π and e, it is clear that today, no human would bother to do the statistics on the digits. The machine will compute the digits and inspect them before something is returned to the human.

3.4.3 Time and Processes

One final characteristic that I would like to consider briefly here, is the significance of time and processes within computer-assisted mathematics.[13] In a study which attempts at differentiating between so-called "computer science thinking" and "classical" mathematical thinking, Knuth sampled nine mathematical text books in order to get a firmer grip on the possible differences. His conclusions are [26]:

> Computer scientists will notice [...]that two types of thinking are absent from the examples we have studied [...] In the first place, there is almost no notion of "complexity" or economy of operation in what we have discussed. Bishop's mathematics is constructive, but it does not have all the ingredients of an algorithm because it ignores the "cost" of the constructions. [...] The other missing concept is related to the "assignment operation" :=, which changes values of quantities. More precisely, I would say the missing concept is the dynamic notion of the *state* of a process: "How did I get here? What is true now? What should happen next if I'm going to get to the end?" Changing states of affairs, or snapshots of a computation, seem to be intimately related to algorithms and algorithmic thinking. Many of the concepts of data structures [...] depend very heavily on an ability to reason about the notion of process states, and we rely on this notion also when studying the interaction of processes that are acting simultaneously.

Knuth's conclusion is in a certain sense not surprising: "classical" mathematics has a very different sense of time as compared to computer science. A computation is executed as a dynamical process that develops and changes over time. One of the reasons why time is really an issue within computing is the combination of very fast computation (relative to human computation) with the fact that all computations on

[13] I am in fact very much indebted to Maurice for having drawn my attention to this fundamental difference between mathematics and computer science in a report he wrote for my dissertation.

a standard computer are finite both with respect to time and space. This stands in sharp contrast with mathematics where the infinite rules and computational speed is only rarely a concern. In a report describing the state of computer science and engineering research this difference is described as follows [1, p. 9]:

> Mathematics deals with theorems, infinite processes, and static relationships, while computer science emphasizes algorithms, finitary constructions, and dynamic relationships. If accepted, the frequently quoted mathematical aphorism, the system is finite, therefore trivial, dismisses much of computer science.

This time dimension of computations becomes more explicit in the light of the fact that most computational processes are often not reversible and this in contrast to mathematics. As Margenstern explains [32, p. 645]:

> Mathematical theories make use of reversible time; how can they exhaust nature, which is not at all reversible? Let us note that in our discrete time of computations, time is irreversible: it is very often extremely difficult to run an algorithm backward. At the highest level of generality it is impossible.

The introduction of the arrow of time into computations combined with the finiteness of the machine also affects the mathematics that is done with such computations. For instance, if one is working with iterations over the reals, the fact that one has to work with finite approximations of these reals, combined with the fact that one is squeezing thousands or even millions of computations in a short time span, necessitates a study of the error propagations that may occur along the process. In fact, it is this problem that lies at the very foundation of the study of non-linear equations like the quadratic iterator that I mentioned before (See e.g. [37, pp.3–4]).

The processual and dynamical character of computation is reflected both on the hardware as well as on the programming level. We already saw the example of the assignment statement on the programming level in Knuth's quote (See also Margenstern:2012). That this processual character of computations needs to be reflected also on the programming level, was already understood by von Neumann. Indeed, in a series of reports which introduces the well-known flowchart notation, he explains [17]:

> [C]oding is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of a meaning.

In fact, as he explains elsewhere, it is exactly this dynamical nature of computation combined with the speed by which it is executed that results in the need for logical control [37]:

> [C]ontemplate the prospect of locking twenty people for two years during which they would be steadily performing computations. And you must give them such explicit instructions at the time of incarceration that at the end of two years you could return and obtain the correct result for your lengthy problem! This dramatizes the necessity for high planning, foresight, and consideration of the logical nature of computation. This integration of logic in the problem is a consequence of the high speed.

On the hardware level, standard CPU design relies on a central clock signal that is emitted to all the hardware units. Without that central pulse which is used to control time, our current architectures would not be able to sequence and synchronize their operations let alone be able to "remember something".

3.5 Some (New and Open) Problems

Faced with a new practice created by the introduction of the computer, it becomes necessary for the mathematician who wants to use a computer in his research to reflect on a wide range of (new) problems or challenges that, at least, partially integrate a way of thinking that is more akin to computer science. I already indicated some important developments within mathematics that are directly rooted in this new practice, like for instance within the field of numerical analysis or the study of randomness. In what follows I will sketch *some* (new and open) problems that (can) arise within a context of computer-assisted mathematics in more detail.

3.5.1 Mathematical Understanding

It is well-known that results coming from computer-assisted mathematics, especially so-called computer-assisted proofs, are not very much appreciated by a part of the mathematical community. One important reason for this is that it is claimed that such results do not provide any understanding of the problem resolved with them. One famous example in this context is the computer-assisted proof of the four color theorem. In this context, Bonsall, a mathematician at the university of Edinburgh stated [2, p. 14]:

> It is no better to accept without verification the word of a computer than the word of another mathematician [...] We cannot possibly achieve what I regard as the essential element of proof – our own personal understanding – if part of the argument is hidden away in a box. [...] Perhaps we are seeing the birth of a new kind of computer-assisted quasi-mathematics, but is has no place in the science of mathematics and if it is to survive must develop its own scientific ethos – perhaps more akin to the experimental sciences.

Another mathematician, Ian Stewart, complains not only about the fact that part of such proofs are hidden but also about their lack of structure, viz. (Quoted from [30, p. 41]):

> [Such proofs do] not give a satisfactory explanation *why* the theorem is true. This is [...] mostly because it is so apparently structureless. The answer appears as a kind of monstrous coincidence. Why is there an unavoidable set of reducible configurations [within the four-color problem]? The best answer at the present time is: there just is. The proof: here it is, see for yourself. The mathematician's search for hidden structure, his pattern-binding [sic] urge, is frustrated.

The fact that with computer-assisted proofs, part of the proof is generated by a machine and mediated by a very lengthy code implies that, first of all, such computer-assisted proofs are far from being “elegant” and very lengthy and, secondly, that part of the proof is in fact not “surveyable” by a human. Moreover, such proofs are mostly based on a case-by-case analysis and, as such, are in need of a more non-uniform approach which is usual business in computer science but not in mathematics. For instance, for the four-color theorem over 1500 cases were derived! Add to this that such proofs are considered to be more error-prone because of their length and the programming and hardware involved and the mathematician is left with a feeling that this cannot be a “good” insightful proof. It is certainly true that the understanding one gains from, say, one of the classical proofs of the Pythagorean theorem cannot be identical to the understanding one gains from a proof that contains hundreds of pages of programs and millions of computations. And it is indeed not very insightful to “have a look” at the over 1500 cases of the original proof of the four color theorem just as there is no insight into why there are x cases rather that $x-1$ cases!

Does this mean that there is no understanding whatsoever to be found in proofs such as that of the four color theorem? I do not think so. Rather it is my view that they provide a different kind of understanding. Indeed, from the perspective of a computer-assisted proof, it makes no sense to ask why a given problem reduces to x cases and not to y but it does make sense to follow the overall structure of such proofs as sketched in their published versions and to see how and why this semi-algorithmic structure works. To put it in the words of Borwein, a well-known advocate of experimental mathematics [3]: “*[T]he act of programming – if well performed – always leads to more insight about the structure of the problem.*”

This problem of what kind of understanding computer-assisted proofs such as the four-color theorem convey is a non-trivial problem which will have to be addressed properly by the mathematical community especially if, in the future, more and more mathematical results would be proven in this way.

3.5.2 Hidden Algorithms and Explorations

One important characteristic of (explorative) computer-assisted mathematics is that part of the mathematics required in tackling a given problem is internalized into the machine. I briefly discuss two problems here that are partially rooted in this characteristic.[14] First of all, with current programming environments, much of the algorithms being used are no longer explicitly programmed by the mathematician who uses them but are instead called by their name. This means that part of the knowledge that is used within computer-assisted mathematics is unknown to the mathematician unless he/she has bothered him/herself with looking at the details of the subroutines called by the main program. This becomes more problematical in the

[14] Note that the above problem of mathematical understanding is also partially rooted in the internalization of knowledge inside the machine.

light of software that is not open source. As Joris van der Hoeven, mathematician and developer of GNU TeXmacs and Mathemagix explains:[15]

> As a mathematician, I am deeply convinced that only free programs are acceptable from a scientific point of view. I see two main reasons for this:
>
> - A result computed by a "mathematical" system, whose source code is not public, can not be accepted as part of a mathematical proof.
> - Just as a mathematician should be able to build theorems on top of other theorems, it should be possible to freely modify and release algorithms of mathematical software.
>
> However, it is strange, and a shame, that the main mathematical programs which are currently being used are proprietary. The main reason for this is that mathematicians often do not consider programming as a full scientific activity. Consequently, the development of useful software is delegated to "engineers" and the resulting programs are used as black boxes.

In a certain sense, the reliance on algorithms without having gone through all of their details, is comparable to referring to common or accepted mathematical knowledge without having gone through every of its details. However, if this knowledge is simply not free to access then the challenges posed by computer-assisted proofs are perhaps rather small when compared to those posed by non open-source mathematical software! If, in the future, computer-assisted mathematics would become more important, this will become a major problem by itself touching upon such issues as scientific reproducibility and patentability.

A second problem that I would like to discuss briefly here concerns the idea of letting the computer do part of the exploration. It is a well-known practice that, when one uses a computer today, one will probably instruct it not to provide the mathematician with all data computed but with a certain selection of them, be it by way of visualizations, plots, statistics, etc. However, by delegating more and more of the inspection to the machine itself one unavoidably also makes more assumptions about the data to be inspected by the machine and one runs the risk of missing out on some essential information. One example in this context comes from Wolfram who was studying so-called mobile automata, a class of computational devices which differ from cellular automata in that they do not update all cells in parallel but one at a time. Instead of looking explicitly at the "raw" behavior of these automata, Wolfram automated his search for a particular type of behavior [39]:

> [B]eing convinced that more complicated behavior must be possible, [...] I wrote a program that would automatically search through large numbers of mobile automata. I set up various criteria of the search, based on how I expected mobile automata could behave. And quite soon, I had made the program search a million mobile automata, then ten million. But still I found nothing. So then I went back and started looking by eye at mobile automata with large numbers of randomly chosen rules. And after some

[15] Extracted from `http://www.texmacs.org/tmweb/manual/webman-about.en.html` on February 24, 2014.

> time what I relayed was that with the compression scheme I was using there could be mobile automata that would be discarded according to my search criteria, but which nevertheless had interesting behavior.

As becomes clear from this example, by internalizing part of the exploration one is always making certain assumptions which can result in errors. Another example is the use of Monte Carlo methods: since the 40s this has become a standard method in computer-assisted science in general. However, contrary to the early years of the Monte Carlo method, one only rarely questions the statistical assumptions underpinning the method.

The internalization of knowledge inside of the machine implies a number of problems that are far from trivial. The bottom line with these type of problems seems to be this: the more knowledge and assumptions that are hidden inside the machine, the more important it becomes for the mathematician to be critical about or at least be aware of the knowledge he/she is presupposing.

3.5.3 Finiteness, Time and Unpredictability

Perhaps one of the most important differences between mathematics and computer science is the fact that computer science deals with computational processes that develop over and are limited by time and space. Hence, it should not be surprising that this feature results in several challenges for computer-assisted mathematics.

One important side-effect of the time-sensitive character of computations is their unpredictability. As Hamming explains [20]:

> One often hears the remark that *computers can only do what they are told to do*. True, but that is like saying that, insofar as mathematics is deductive, once the postulates are given all the rest is trivial. [T]he truth is that in moderately complex situations, such as the postulates of geometry or a complicated program for a computer, it is not possible on a practical level to foresee all of the consequences. Indeed, there is a known theorem that there can be no program which will analyze a general program to tell how long it will run on a machine without actually running the program.

This unpredictability is not just some theoretical problem or property. It is in fact this unpredictability that usually brings mathematicians to the computer: it is because one cannot predict the outcome of a certain computational problem that one needs to rely on and trust the machine's abilities. However, many such problems, when programmed, may contain infinite loops or need an unreasonable if not infinite amount of time and/or space and such that this cannot be predicted by the mathematician. This necessitates the need for developing local programming strategies that are often "experimental" in nature, viz., they do not necessarily result in a (correct) solution and often require adjustments in the light of new computational results. For instance, Hales had to experimentally determine several constants in the process of developing the proof of the sphere packing problem [5]:

> Hales remarks for instance that "The constant 2.51 was determined experimentally to have a number of desirable properties", and similar experimental determinations recur

> repeatedly throughout the paper. Several of the initial decisions had to be modified in the light of later calculations.

Another problem that comes up in this context can be illustrated with the following example coming from my own computer-assisted research on a class of computational devices known as tag systems. Also here it was the unpredictabilty of the computational processes of these devices that made it necessary for me to include certain limitations into the programs used to study these devices and which instructed the computer when it should give up on a certain tag system. For instance, in studying the behavior of tag systems with arbitrary initial words of length 300, I included the limitation that the program should stop if (1) after 10,000,000 computational steps the tag system had not halted or become periodic and (2) one of the produced words became longer than 15,000,000. Initial words that resulted in more than 10,000,000 computational steps were tentatively classified as possible immortals. But what kind of derivations can one make on the basis of information that is based only on a finite piece of information of a (possibly) infinite dataset? In how far are, for instance, the results from a statistical analysis based only on the first part of a computation representative for the whole computation?

These kind of problems are not only characteristic for computer-based research but for explorative mathematics in general: experimental research on the Riemann-zeta function, the twin prime conjectures, the Goldbach conjecture or the Collatz problem all suffer from the problem that one has information only about a finite subdomain of the (possibly) infinite domain. Consider for instance the Collatz function C. Given an integer n_0, if n_0 is odd compute $n_1 = 3n_0 + 1$ if not, compute $n_1 = n_0/2$. If n_1 is odd, compute $n_2 = 3n_1 + 1$, else, compute $n_2 = n_1/2$, etc. The Collatz conjecture states that for any n_i, there is an m_j such that after m_j iterates of C on n_i it will end in the loop 1, 2, 4. Now, assume we want to compute C for some very large integer. We start the computation on our computer but after weeks of iterations we have an overflow. What would this tell us? Conversely, what can we derive from the fact that the conjecture has been verified for 5×2^{60} integers? It is in such experimental context that one is in need of a "conduct of good behavior", a certain standard of when one has reasons to make a conjecture and when not and one should always be extremely careful in making generalizations based on a finite number of instances.

Another problem, which is directly related to the finiteness of the computer, comes up in the context of computations over the reals and is related to the Mandelbrotset. Given the function:

$$c \to c^2 + c,\ c \in \mathbb{C}$$

the Mandelbrot set is defined as:

$$M = \{c \in \mathbb{C} | c \to c^2 \to c^2 + c \to \dots \text{remains bounded}\}$$

The set is most well-known for its intricate boundary which gives rise to very beautiful visualizations. One important question, that was formulated by Roger Penrose, is whether it is decidable for any c whether or not it belongs to the Mandelbrotset.

There are now two major competing models of computation over the reals to address this and other related problem:

> [The bit model] reflects the fact that computers can store only finite approximations to real numbers. Roughly speaking, a real function f is computable in the bit model if there is an algorithm which, given a good rational approximation to x, finds a good rational approximation to $f(x)$. The second approach is the algebraic approach, which abstracts away the messiness of finite approximations and assumes that real numbers can be represented exactly and each arithmetic operation can be performed exactly in one step. The complexity of a computation is usually taken to be the number of arithmetic operations (for example, additions and multiplications) performed. The algebraic approach applies naturally to arbitrary rings and fields, although for modeling scientific computation the underlying structure is usually $\mathbb{R}$ or $\mathbb{C}$.

The second approach is most known through the work of Blum, Shub and Smale: within their model, the Mandelbrotset turns out to be undecidable. However, the drawback of the model is that functions which one would consider intuitively as being decidable, like, for instance, a simple transcendental function such as e^x, also turn out to be undecidable under this model. This is not the case with the bit model. In this latter model, the undecidability of the Mandelbrot set is still an open problem. One interesting consequence of this model is that *if* a function f is uncomputable than there is no good rational approximation of f. As such, the undecidability of the Mandelbrot would imply that we cannot rely on our computer-generated visualizations of the set. Indeed, since any such visualization is but a finite approximation of the "real" set, we cannot rely on it and hence also not on the observations and conjectures that are rooted in those visualizations.

3.5.4 Unreliability

From the previous sections it becomes clear that some of the characteristics of computing on a machine result in a number of problems within computer-assisted mathematics that need to be taken into account in some or the other way by the mathematician who uses the computer in his/her research. All of these different problems contribute to a feeling that the results from computer-based research are quite unreliable – in fact, it is often for this reason that the label "experimental" is used. The fact that usually hundreds or thousands of lines of code are involved, the fact that the human him/herself can often only wait and see, the fact that one does not have access to all data generated during the process, the fact that one may encounter truncation errors, etc indeed do not give an impression of high reliability. Add to this that many results are "explorative" results and it becomes understandable that many a mathematician turns his/her back to the results that are machine-aided. On the other hand, however, one cannot deny that the computer in fact gives us the possibility, to state it again in Lehmer's words, "*to explore the terrain that has been staked out so freely and that something worth proving will be discovered in the rapidly expanding universe of mathematics*". The choice is there to make for the mathematician but if the choice is in favor of the machine it is clear that one needs to deal in some or the other way with these problems of unreliability that becomes so explicit in this context. Two extreme positions have been proposed in this context and relate back to the tensions I sketched in Sec3.1 between formalism and "experimentalism"; rigorous and non-rigorous math, etc as highlighted in the contrast between Hilbert's work and that of the late von Neumann.

One extreme position is represented by Doron Zeilberger, a well-known mathematician and strong supporter of computer-assisted mathematics. He has claimed that, even though today [40]:

> the mathematical faith is *thou shalt prove everything rigorously* [...] a new testament is going to be written. Although there will always be a small group of "rigorous" old-style mathematicians [...] who will insist that the true religion is theirs, and that the computer is a false Messiah, they may be viewed by future mainstream mathematicians as a fringe sect of harmless eccentrics [...] In the future, not all mathematicians will care about absolute certainty, since there will be so many exciting new facts to discover. We will have (both human and machine) professional *theoretical* mathematicians, who will develop conceptual paradigms to make sense out of the empirical data, and who will reap Fields medals along with (human and machine) *experimental* mathematicians. [...] As absolute truth becomes more and more expensive, we would sooner or later come to grips with the fact that few non-trivial results could be known with old-fashioned certainty. Most likely we will wind up abandoning the task of keeping track of price altogether, and complete the metamorphosis to non-rigorous mathematics.

This is quite a strong claim: Zeilberger is convinced that human "proofs" as certificates of truth will be abolished and replaced by a non-rigorous mathematics because few non-trivial truths that can be proven by short and human proofs will be left. Again, it is impossible at this point to evaluate this prediction, but the idea that mathematics would *completely* abandon the very idea of proof as a guarantee of certainty, is precarious. This, however, does not mean that there will be no situations where mathematical results, which are true only to a very high degree of certainty, will be more easily accepted. It is already a common practice to use for instance probabilistic algorithms and, in the context of computer-assisted proofs, the idea of corroboration, where different groups of researchers prove the same result independently from each other, has also already been suggested. For instance, Brady remarked about his proof of the determination of the Busy Beaver winner for four-state Turing machines [4]:[16]

> While not all of the exploratory activities are reproducible, the runs [...] can be reproduced, so that by utilizing the techniques described in this paper the proof can be corroborated.

An opposite response to the challenges posed by the computer for mathematics, is to take the side of full rigorous mathematics by means of formalized proofs with the help of interactive proof assistants such us HOL or Isabel. The four-color theorem, for instance, has been formalized in this way by Gonthier [18] and also Hales and

[16] The Busy Beaver problem for a class of binary Turing machines with m states is the problem to determine which of these Turing machines will print out the largest number of 1s when halting.

others are now working for some years on the so-called FlySpeck project which aims at formalizing the computer-assisted proof of Kepler's conjecture [19]. The reason why people like Gonthier and Hales are working on such proofs is rooted in the uncertainty associated with lengthy computer-assisted proofs. By means of formalization, it is believed that possible errors are excluded since every single logical step is (supposedly) verified by the proof assistant [19]:

> A formal proof is a proof in which every logical inference has been checked, all the way back to the foundational axioms of mathematics. No step is skipped no matter how obvious it may be to a mathematician. A formal proof may be less intuitive, and yet is less susceptible to logical errors. Because of the large number of inferences involved, a computer is used to check the steps of a formal proof.

This approach of formalized proofs however is not only applied to lengthy computer-assisted proofs. The developments of automated verification and proving has resulted in initiatives like the Mizar project which aim at formalizing the whole of mathematics. The motivations are the same: a believe that, partially due to the introduction of the computer into mathematics, mathematical knowledge is becoming too complex and, as a consequence, that there might be a tendency towards non-rigouresness. In the QED manifesto, which proposes the formalization of all mathematics, this is stated as follows [34]:

> [T]he increase of mathematical knowledge during the last two hundred years has made the knowledge, let alone understanding, of all of even the most important mathematical results something beyond the capacity of any human. For example, few mathematicians, if any, will ever understand the entirety of the recently settled structure of simple finite groups or the proof of the four color theorem. Remarkably, however, the creation of mathematical logic and the advance of computing technology have also provided the means for building a computing system that represents all important mathematical knowledge in an entirely rigorous and mechanically usable fashion. The QED system we imagine will provide a means by which mathematicians and scientists can scan the entirety of mathematical knowledge for relevant results and, using tools of the QED system, build upon such results with reliability and confidence but without the need for minute comprehension of the details or even the ultimate foundations of the parts of the system upon which they build.

Also here it seems precarious to believe that, in the future, mathematical knowledge will become completely formalized in some computer-based system. Besides the fact that there is a problem here of infinite regress, indeed, "*who will control the controller*",[17] "*[i]t is a large labor-intensive undertaking to transform a traditional proof into a formal proof*" [19]. Hence, just as one can wonder whether the average mathematician will take the side of fully non-rigorous mathematics à la Zeilberger, one can wonder whether he/she will take the side of fully formalized mathematics.

These two opposite alternatives, formalized and non-rigorous mathematics, are two sides of the same medal. They are both rooted in a believe that mathematics is becoming less reliable and rigorous, partially due to the introduction of the computer. At this point, it is hard to imagine a future where either of these two extremes

[17] Remark from Maurice Margenstern during MCU 2013.

in one or the other form will really become a dominant practice. But is it really desirable that we evolve to any of these two alternatives? Faced with a new situation in mathematics, I believe that mathematics would only gain if both sides, rather than one dominating the other, would try and complement/advance each others findings.

3.6 Some Afterthoughts

Computer science has many different historical roots and mathematics is certainly one of them: formalization and algorithms were and are a part of the mathematical discourse and this long before the computer inspired the formation of a new discipline known as computer science. Today we see that a converse influence is manifesting itself: the computer is also starting to affect mathematics. Computer-assisted exploration is resulting in a revival of so-called experimental mathematics, a practice which is certainly not new to the history of mathematics but has been pushed to the background for the last two centuries. It is within such practice that algorithms and non-uniform methods rather than general theories are at play.

It is thus not surprising that explorative computer-assisted mathematics has more in common with a computer science way of thinking than "classical" mathematics. The reason for this is not the mere fact that both are (usually) concerned with computation, but is also rooted in the use of a physical, finite machine. From a contemporary perspective, the explicit focus in this paper on how the machine itself affects mathematics may seem a bit old-fashioned. Indeed, today, there is a tendency towards abstraction up to the development of interfaces which are made as "user-friendly" as possible, hiding at the same time that one is using a technological device. In fact, it has become rather common to see computation as something that transcends the computer itself and can be found anywhere (See e.g. [11]). Evidently, we are very much in need of such layers of abstraction. Amongst others, they allow us to overcome the programming bottleneck that characterized the early machines. However, if the addition of such layers goes hand-in-hand with forgetfulness about the constraints imposed by the machine or, more generally, the physicality of computation, on our (mathematical) thinking, one also gives away control, not only to the machine, but also, more importantly, to the developers of these different layers of abstraction.

The reliance on the computer implies that one needs to develop algorithms that *really* work, also, from the machine's eye and for which, as a consequence, mathematical elegance is constrained by the need for procedures that have to be scientific before becoming an art.[18] It also means that a new range of problems need to be faced by the mathematical community, problems that originate in, amongst others,

[18] I am referring here to Knuth's Turing award lecture titled "Computer programming as an art" in which he reflects on the multiple ways in which "science" and "art" can be used in the context of programming, and beyond. During that lecture he stated: "*Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it. Since the notion of an algorithm or a computer program provides us with an extremely useful test for the depth of our knowledge*

the time-based character of computations which has become more explicit through their execution on a very fast but limited machine; the internalization of procedures inside of the machine and the fact that part of the work is, literally, out of control of the human. These problems result in two extreme positions with respect to computer-assisted explorative mathematics and the more general observation that mathematical knowledge is becoming too complex to guarantee rigor: on one side, there are those who argue for the complete formalization of mathematical knowledge and thus "automated rigor", on the other side, there are those who want the exact opposite, viz. "automated non-rigor". Interestingly, as I argued in Sec. 3.1, one can trace similar oppositions within the short history of computer science itself, which, very roughly speaking, boils down to the fact that computer science is as much about engineering as it is about mathematics.[19] The formal verification debates, with its high point in the 80s, is perhaps one of the more explicit and harsh exemplifications of this opposition. However, just as Lehmer's two styles need not be exclusive in mathematics, the same holds true for computer science. In fact, within computer science, both engineering and formalization are so intricately tied together through the physicality of computation that one can only feel regret when these two "styles" are regarded as each others opposite. One can only hope that as computer science matures, that the disciplinary fences that have already been built, will not result in a complete separation of both styles. This, I believe, would only result in an impoverishment of the field.

References

1. Arden, B.W. (ed.): What can be automated? The computer science and engineering study. MIT Press (1980)
2. Bonsall, F.F.: A down-to-earth view of mathematics. The American Mathematical Monthly 89(1), 8–15 (1982)
3. Borwein, J.: Implications of experimental mathematics for the philosophy of mathematics. In: Gold, B., Simons, R.A. (eds.) Proof and Other Dilemmas: Mathematics and Philosophy, pp. 33–60. Mathematical Association of America (2008)
4. Brady, A.H.: The determination of the value of Radó's noncomputable function σ for four-state Turing machines. Mathematics of Computation 40(162), 647–665 (1983)
5. Braverman, M., Cook, S.: Computing over the reals: Foundations for scientific computing. Notices of the AMS 53(3), 318–329 (2006)
6. Bullynck, M., De Mol, L.: Setting-up early computer programs: D. H. Lehmer's ENIAC computation. Archive for Mathematical Logic 49, 123–146 (2010)
7. Conway, J.H., Goodman-Strauss, C., Sloane, N.J.A.: Recent progress in Sphere Packing. Contemporary Mathematics (to appear), `http://comp.uark.edu/~strauss/papers/sphere.pdf`

about any given subject, the process of going from an art to a science means that we learn how to automate something." [25]

[19] In fact, it has been argued that there are three styles within computer science: a mathematical style characterized by theoretical work, an engineering style characterized by design and a scientific style characterized by abstraction and modeling [10]. A discussion of this third style lies beyond the scope of this paper.

8. Clenshaw, C.: Polynomial approximations to elementary functions. Mathematical Tabels and Other Aids to Computation 8(47), 143–147 (1954)
9. Corry, L.: Number crunching vs. number theory: computers and FLT, from Kummer to SWAC (1850–1960) and beyond. Archive for the History of Exact Sciences 62, 393–455 (2008)
10. Denning, P.J., Comer, D.E., et al.: Computing as a discipline. Communications of the ACM 32(1), 9–23 (1989)
11. Denning, P.J.: Computing as a natural science. Communications of the ACM 50(7), 13–18 (2007)
12. De Mol, L.: The proof is in the process. A preamble for a philosophy of computer-assisted mathematics. In: Galavotti, M.-C., et al. (eds.) New Directions in the Philosophy of Science. Series The Philosophy of Science in a European Perspective, vol. 5, pp. 15–34. Springer
13. Dijkstra, E.W.: On the design of Machine Independent Programming Languages, Report MR34. Stichting Mathematisch Centrum, Amsterdam (1961)
14. Dijkstra, E.W.: Programming as a discipline of mathematical nature. The American Mathematical Monthly 81(6), 608–612 (1974)
15. Dijkstra, E.W.: How computing science created a new mathematical style, EWD 1073 (1990)
16. Gandy, R.: The confluence of ideas in 1936. In: Herken, R. (ed.) The Universal Turing Machine, pp. 55–111. Oxford University Press, Oxford (1988)
17. Goldstine, H.H., von Neumann, J.: Planning and coding of problems for an electronic computing instrument, vol. 2, part I, II and III (1947-1948), Report prepared for U. S. Army Ord. Dept. under Contract W-36-034-ORD-7481
18. Gonthier, G.: Formal proof – The four color theorem. Notices of the AMS 55(11), 1382–1393 (2008)
19. Hales, T., Harrison, J., et al.: A revision of the proof of the Kepler conjecture. Discrete and Computational Geometry 44, 1–34 (2010)
20. Hamming, R.W.: Impact of Computers. The American Mathematical Monthly 72, 1–7 (1965)
21. Hamming, R.W.: One man's view of computer science. Journal of the ACM 16, 3–12 (1969)
22. Hartree, D.R.: Calculating instruments and machines. University of Illinois Press (1949)
23. Hilbert, D.: Naturerkennen und Logik. Naturwissen 18, 959–963 (1930)
24. Knuth, D.: Computer science and its relation to mathematics. The American Mathematical Monthly 81(4), 323–343 (1974)
25. Knuth, D.: Computer programming as an art. Communications of the ACM 17(12), 667–673 (1974)
26. Knuth, D.: Algorithmic thinking and mathematical thinking. The American Mathematical Monthly 92(3), 170–181 (1985)
27. Knuth, D.: Algorithmic themes. In: Duren, P., et al. (eds.) A Century of Mathematics in America, Part I, pp. 439–445. American Mathematical Society (1988)
28. Lehmer, D.H.: Mathematical methods in large scale computing units. In: Proceedings of Second Symposium on Large-Scale Digital Calculating Machinery, pp. 141–146. Harvard University Press, Cambridge (1949, 1951)
29. Lehmer, D.H.: Mechanized mathematics. Bulletin of the American Mathematical Society 72, 739–750 (1966)
30. MacKenzie, D.: Slaying the Kraken: The sociohistory of a mathematical proof. Social Studies of Science 29(1), 7–60 (1999)

31. Mancosu, P., Zach, R., Badesa, C.: The Development of Mathematical Logic from Russell to Tarski, 1900 - 1935. In: Haaparanta, L. (ed.) The Development of Modern Logic, pp. 318–470. Oxford University Press, New York (2009)
32. Margenstern, M.: Comment. In: Zenil, H. (ed.) A Computable Universe, pp. 645–646. World Scientific, Singapore (2012)
33. Post, E.: Absolutely unsolvable problems and relatively undecidable propositions — Account of an anticipation. In: Davis, M. (ed.) The Undecidable. Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions, Raven Press, New York (1965); Corrected Republication, pp. 340–433. Dover Publications, New York (2004)
34. The QED Manifesto, `http://www.cs.ru.nl/~freek/qed/qed.html`
35. Ulam, S., von Neumann, J.: Bulletin of the American Mathematical Society 64, 1–49 (1958)
36. von Neumann, J.: The Mathematician. In: Heywood, R.B. (ed.) The Works of the Mind, pp. 180–196. University of Chicago Press (1947)
37. von Neumann, J.: Electronic Methods of Computation. Bulletin of the American Academy of Arts and Sciences 1, 2–4 (1948)
38. Wegner, P.: Research paradigms in computer science. In: Proceedings of the 2nd international conference on Software Engineering, ICSE 1976, pp. 322–330 (1976)
39. Wolfram, S.: A new kind of science. Wolfram Media, Champaign (2002)
40. Zeilberger, D.: Theorem for a price. Tomorrow's semi-rigorous mathematical culture. The Mathematical Intelligencer 16(4), 11–18 (1994)

Chapter 4
Sampling a Two-Way Finite Automaton

Zhe Dang, Oscar H. Ibarra, and Qin Lin

Abstract. We study position sampling in a 2-way nondeterministic finite automaton (2NFA) to measure the information dependency and information flow between state variables, based on the information-theoretic sampling technique proposed in [16]. We prove that for a 2NFA, the language generated by position sampling is regular. We also show that for a 2NFA, we can effectively find a vector of sampling positions that maximizes dependency and information flow in a run of the 2NFA. Finally, we give some language properties of sampled runs of 2NFAs augmented with restricted unbounded storage.

4.1 Introduction and Motivation

One way to understand the behavior of a software system is through observation. That is, when the system runs, we record a sequence of values of all or part of its state variables like PC (program counter), variable values, pointer locations, stack frames, I/O, etc. Such a sequence, called a trace, can later be used for either offline or online analysis. A trace contains valuable information on dependency and information flow among the state variables, where "information flow", a jargon in

Zhe Dang
School of Computer Science and Technology,
Anhui University of Technology, Ma'anshan, China and School of Electrical Engineering and Computer Science Washington State University, Pullman, WA 99164, USA
e-mail: `zdang@eecs.wsu.edu`

Oscar H. Ibarra
Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
e-mail: `ibarra@cs.ucsb.edu`

Qin Lin
School of Computer Science and Technology,
Anhui University of Technology, Ma'anshan, China
e-mail: `qinli@ahut.edu.cn`

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_4

software security, between state variables means the information, totally [11] or partially [12,17], flows from secret variables to public ones. When confidentiality is the concern, this kind of information flow is illegal, and must be constrained.

There have been a number of white-box techniques using static analysis, such as program slicing [28,30], type analysis [2,23,25], probabilistic analysis [18,20,24], and information-theoretic analysis [4,12,19], to find dependency among variables. In our recent work [16], we introduce a new information-theoretic technique that does not depend on static analysis and can be adapted to black-boxes easily as shown in [16].

There has been a fundamental notion shown below, proposed by Shannon [22] and later by Chomsky and Miller [3], that computes the information quantity in a string (word). Let L be a set of words over a given finite and non-empty alphabet Σ, and $S_n(L)$ be the number of words of length n in L. The *information rate* λ_L of L is defined as

$$\lambda_L = \lim \frac{\log S_n(L)}{n}.$$

Where the limit does not exist, we take the upper limit, which always exists for a finite alphabet. Throughout this paper, the logarithm is in base 2. Intuitively, λ_L is the average amount of information per symbol contained in a word in L. We emphasize that the information rate does not require any probabilistic nor statistical arguments. Instead, it is a characteristic of the language L itself.

Based on the Shannon information rate, we introduced information dependency and information flow in [16]. We now briefly state the definitions. Consider a device M where we observe two of its state variables, x_1 and x_2, and record a trace, α. M, in general, is nondeterministic. We use λ_{x_1,x_2} to denote the information rate of the language of all such traces α. When we replace every value of x_2 with a special symbol $-$ in every α in the language, we obtain a new language with its information rate denoted by $\lambda_{x_1,-}$. Similarly, one can define λ_{-,x_2}. Now, information dependency between x_1 and x_2 is defined as

$$D(x_1;x_2) = \frac{\lambda_{x_1,-} + \lambda_{-,x_2} - \lambda_{x_1,x_2}}{\lambda_{x_1,x_2}}.$$

(By default, $\frac{0}{0} = 0$.) Notice that, as in the classic Venn diagram of Shannon information, the quantity $\lambda_{x_1,-} + \lambda_{-,x_2} - \lambda_{x_1,x_2}$ resembles the "mutual information rate" between x_1 and x_2 which characterizes the bit rate that is shared between x_1 and x_2 in a trace.

In addition to information dependency, one can also define information flow as

$$F(x_1;x_2) = \frac{\lambda_{x_1,-} + \lambda_{-,x_2} - \lambda_{x_1,x_2}}{\lambda_{-,x_2}}.$$

Intuitively, this catches the amount of information that "flows" from x_1 to x_2 in a trace; i.e., the mutual information rate $\lambda_{x_1,-} + \lambda_{-,x_2} - \lambda_{x_1,x_2}$ is essentially the portion of information rate of x_2 that comes from x_1. Similarly, one can define the information flow from x_2 to x_1 as

$$F(x_2;x_1) = \frac{\lambda_{x_1,-} + \lambda_{-,x_2} - \lambda_{x_1,x_2}}{\lambda_{x_1,-}}.$$

The above definitions come from our previous work [16], where information flow is studied for various one-way automata. In this paper, our focus is on two-way nondeterministic finite automata (2NFAs). Notice that, even though 2NFAs are equivalent to their deterministic version in terms of language acceptance, the runs of 2NFAs (which are not necessarily regular) are still difficult to handle when computing their information rate [7]. In this paper, instead of looking at the runs, we study the sampled executions or traces, where the idea of sampling used in [16] is generalized. We need the following fundamental result.

Theorem 1. *The information rate of a regular language is computable [3].*

As pointed out in [3], the information rate can actually be efficiently computed using a matrix algorithm. Recently, we have implemented the algorithm [26] and confirmed the efficiency when approximately computing the information rates of some fairly large C programs [7]. It turns out that the notion itself is useful in software analysis and testing [5,6,27].

4.2 Information Dependency and Information Flow in 2NFAs

Let M be a 2NFA. On its two-way input tape, M reads the input, say w, and performs state transitions. As usual, M starts from its initial state, and when it enters an accepting state, w is said to be accepted by M. The state sequence in the state transition sequence witnessing the acceptance is called an accepting run. Since M is nondeterministic, there could be more than one accepting run for the given w. We use $L(M)$ to denote the set of all words w accepted by M and use $L_{\text{run},M} = \{\alpha_w : \alpha_w$ is an accepting run, $w \in L(M)\}$ to denote the set of all accepting runs of M. Clearly, $L_{\text{run},M}$ is not necessarily a regular language on alphabet S (the states in M).

We show an example below of using a 2NFA to specify a security monitoring system in a building.

Example 1. Suppose that k surveillance cameras are installed in a building. Clearly, it could be true that all the cameras, no matter where they are installed, cannot completely cover the entire building. Suppose that a person's movement inside the building is modeled as a 2NFA (e.g., every input symbol resembles a door). A natural question arises: Where shall we install the k cameras so that we can obtain the maximal amount of information about the person?

In practice, the state space S is often expressed as the Cartesian product of the state spaces of a number of finite state variables $x_1,\cdots,x_k$, for some k. In this way, every state s in S is a k-arity vector of values. Let $A \subseteq \{x_1,\cdots,x_k\}$ be a set of variables. Let $s\downarrow_A$ be the result of dropping the components in vector s that correspond to variables not in A. Therefore, $s\downarrow_A$ is an $|A|$-arity vector. For a sequence $\alpha = s_0\cdots s_n$, we denote the sequence $s_0\downarrow_A \cdots s_n\downarrow_A$ as $\alpha\downarrow_A$. When α is an accepting run, $\alpha\downarrow_A$ is a *trace* wrt A. Clearly, all such traces form a language $L^A_{\mathrm{run},M} = \{\alpha\downarrow_A : \alpha \in L_{\mathrm{run},M}\}$.

We now give a more general definition of information dependency and information flow [16]. Let L be a set of state sequences and $A,B \subseteq \{x_1,\cdots,x_k\}$ be two sets of variables. We use λ_{L^A}, λ_{L^B} and $\lambda_{L^{A\cup B}}$ to denote the information rates of languages L^A $(=\{\alpha\downarrow_A : \alpha \in L\})$, L^B, and $L^{A\cup B}$, respectively. The *information dependency* between A and B wrt L is defined as

$$D(A;B|L) = \frac{\lambda_{L^A} + \lambda_{L^B} - \lambda_{L^{A\cup B}}}{\lambda_{L^{A\cup B}}}. \tag{4.1}$$

The *information flow* from A to B is defined as

$$F(A;B|L) = \frac{\lambda_{L^A} + \lambda_{L^B} - \lambda_{L^{A\cup B}}}{\lambda_{L^B}}. \tag{4.2}$$

In the definition of a trace wrt A, we require that a trace must record the values of the variables in A at *every* step of the run. This is not always possible considering the fact that the system that M specifies may run at a speed much faster than the values can be recorded. In practice, it often suffices to sample a trace (e.g., record the values once every 100 steps), and as a result, a subsequence of a trace is obtained. We call such a subsequence as a sampled trace. Depending on the scheme used in sampling, the information dependency and information flow may or may not be computable. Notice that the information rate on a sampled trace may not be even a faithful approximation to the information rate on a unsampled trace. Therefore, a decision problem arises, when given a sampling scheme, whether the scheme would roughly maintain the bit rate of the traces of the system.

Example 2. We continue with the previous example. Suppose that each camera is replaced with some sensors (e.g., an infrared sensor for body temperature, a motion sensor for movement, etc.) to monitor an array of m physical characteristics. Hence, a "run" now is the person's behavior sampled from the k sets of sensors. Clearly, if, from the samples, there is a high dependency between two kinds of sensors, then it highly suggests that the two kinds of sensors can be reduced into one (i.e., a less expensive monitoring system is sufficient).

Recall that M works on a two-way input. As pointed out in [7], it is currently an open problem whether the information rate of the set of all accepting runs of a 2NFA is computable or not, even when the input is unary. The difficulty is that all such runs form a non-regular language whose information rate is known computable only for limited cases.

We now consider a way to sample a run of a 2NFA M. In the literature, a crossing sequence is often used for a 2NFA; i.e., to record the state of M whenever the head is under a given position of the input. We now generalize the concept to multiple positions. Let $\mathbf{c} = (c_1,\cdots,c_k)$, for some k, be a monotonic sequence of rational numbers with each $0 \le c_i \le 1$. Let w be a non-null (i.e., $|w| > 0$) input to M. The numbers $p_i = \max(1, \lfloor c_i|w| \rfloor)$ are the *sampling positions* for the input. Notice that, when $c_i = 0$ (resp. $c_i = 1$), it is the first (resp. last) symbol of the input. When M runs, whenever the head is at a sampling position, the state of M is recorded. Therefore, when a run is an accepting run, we may record a sequence of states, called a $\mathbf{c}$-sampling accepting run. We use $L_{\mathrm{run},\mathbf{c}(M)}$ to denote the set of all $\mathbf{c}$-sampling accepting runs for all inputs.

We first consider the case when $k = 1$ and $c_1 = 0$ (hence, only the left end of the two-way input is sampled). In this case, $L_{\mathrm{run},\mathbf{c}(M)}$ is written as $L_{\mathrm{run},\mathrm{lend}(M)}$.

Lemma 1. *Let M be a 2NFA. Then, $L_{\mathrm{run},\mathrm{lend}(M)}$ is a regular language.*

Proof. Without loss of generality, we assume that M enters an accepting state and returns to the left end of the input when it accepts the input. On an input w, we define T_w to be the set of all pairs (s, s') satisfying the following: M starts with state s at the left end of the input w, runs on the w while the head of M is not under the left end of the input, and eventually returns to the left end with state s'. Hence, during the process, the head is at the left end only twice. Since the state set S of M is finite, there are only finitely many distinct T_w's for all $w \in L(M)$. We use $\mathcal{T}$ to denote all of them, which can be constructed effectively. This is because, as an exercise, one can show the following: For each $T \in \mathcal{T}$, the set $\{w : T_w = T\}$ is a regular language. Now, every such T defines a graph with directed edges $(s, s') \in T$. Clearly, a state sequence is in $L_{\mathrm{run},\mathrm{lend}(M)}$ iff, for some $T \in \mathcal{T}$, the state sequence is a walk on the graph T from the initial state of M to an accepting state of M. Again, all such walks form a regular language and the result follows. □

Lemma 1 can be generalized. We first need some definitions.

A counter is a nonnegative integer variable that can be incremented by 1, decremented by 1, or stay unchanged. In addition, a counter can be tested against 0. Let k be a nonnegative integer. A *nondeterministic k-counter machine (NCM)* is a one-way nondeterministic finite automaton, with input alphabet Σ, augmented with k counters. For a nonnegative integer r, we use NCM(k,r) to denote the class of k-counter machines where each counter is *r-reversal-bounded*; i.e., it makes at most r alternations between non-decreasing and non-increasing modes in any computation; e.g., the following counter value sequence

$$0\ 0\ 1\ 2\ 2\ 3\ \underline{3\ 2}\ 1\ 0\ \underline{0\ 1}\ 1$$

is of 2-reversal, where the reversals are underlined. For convenience, we sometimes refer to a machine M in the class as an NCM(k,r). In particular, when k and r are implicitly given, we call M as a *reversal-bounded NCM*. When M is deterministic, we use 'D' in place of 'N'; e.g., DCM. As usual, $L(M)$ denotes the language that M accepts.

Reversal-bounded NCMs have been extensively studied since their introduction in 1978 [14]; many generalizations are identified, e.g., ones equipped with multiple tapes, with two-way tapes, with a stack, etc. In particular, reversal-bounded NCMs have found applications in areas like Alur and Dill's [1] time-automata [8,9], Paun's [21] membrane computing systems [15], and Diophantine equations [29].

Let Y be a finite set of integer variables. An atomic Presburger formula on Y is either a linear constraint $\sum_{y \in Y} a_y y < b$, or a modulus constraint $x \equiv_d c$, where a_y, b, c and d are integers with $0 \leq c < d$. A Presburger formula can always be constructed from atomic Presburger formulas using $\neg$ and $\wedge$. Presburger formulas are closed under quantification. Let S be a set of k-tuples in N^k. S is Presburger definable if there is a Presburger formula $P(y_1, \cdots, y_k)$ such that the set of nonnegative integer solutions is exactly S.

Theorem 2. *Let M be a 2NFA and* $\mathbf{c}$ *be an array of rational numbers between 0 and 1 (inclusive). Then, $L_{\mathrm{run},\mathbf{c}(M)}$ is (effectively) a regular language.*

Proof. Without loss of generality, we assume that M enters an accepting state and returns to the right end of the input when it accepts the input. Also without loss of generality, we assume that $\mathbf{c}$ contains the two end points 0 and 1, and there are k numbers in $\mathbf{c}$. Implicitly, they represent k positions on the input word. Clearly, if the input word w is long enough, the k sampling positions given by $\max(1, \lfloor c_i |w| \rfloor)$ must be distinct. Since there are only finitely many "short" w's, it suffices for us to assume that the k sampling positions are indeed distinct. To ease our presentation, we further assume that these positions are marked; i.e., when originally the input symbol at such a position is a, in the marked input, the symbol is a marked new symbol $\dot{a}$. By assumption, the two ends of the input symbols are also marked. On a marked input word, a *block* is a subword between two neighboring marked symbols where the two marked symbols are included; e.g., when the input word is $\dot{a}bcb\dot{d}ca\dot{c}$, we have two blocks: $\dot{a}bcb\dot{d}$ and $\dot{d}ca\dot{c}$. When M works on a marked input word, it works as if the marked symbols are unmarked. Hence, marked symbols do not alter the behaviors of M. We now consider a block, say $w = \dot{a}u\dot{b}$, where $\dot{a}, \dot{b}$ are marked symbols, and u is an unmarked word. Similar to the proof of Lemma 1, we define the following sets, shown in Fig. 4.1 and described below.

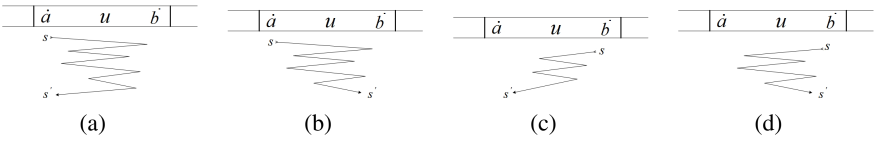


Fig. 4.1 Generating regular sets when reading the block $\dot{a}u\dot{b}$

(a) $[s, \dot{a}, \mathrm{stay}, s']$ is the set of u such that there is a run starting from state s of M as follows. When M is reading $\dot{a}$ of the block at state s, M will come back to this $\dot{a}$. During the run, M works on u two-way. Herein, M's read head is under $\dot{a}$ only twice (the begin and the end of the run) and never moves out of the right of u.

(b) $[s, \dot{a}, \mathrm{right}, s']$ is the set of u such that there is a run starting from state s of M as follows. When M is reading $\dot{a}$ of the block at state s, M does not come back to this $\dot{a}$ again, and works on u two-way. The first time when M moves out of the u, the read head is under the marked symbol to the right of u and with state s'.

(c) $[s,\dot{b},\text{left},s']$ is the set of u such that there is a run starting from state s of M as follows. When M is reading $\dot{b}$ of the block at state s, M does not come back to this $\dot{b}$ again, and works on the u two-way. The first time when M moves out of the u, the read head is under the marked symbol to the left of u and with state s'.

(d) $[s,\dot{b},\text{stay},s']$ is the set of u such that there is a run starting from state s of M as follows. When M is reading $\dot{b}$ of the block at state s, M will come back to this $\dot{b}$. During the run, M works on the u two-way. Herein, M's read head is under $\dot{b}$ only twice (the begin and the end of the run) and never moves out of the left of u.

Again, it is an exercise to show that every set (which could be empty) defined above is regular. We now consider a sequence of k marked symbols, say $B = \dot{b}_1 \cdots \dot{b}_k$ and a marked word $w = \dot{b}_1 u_1 \cdots u_k \dot{b}_k$. We construct a table T_w which consists of all tuples, for all states s, s', and for all $1 \le i < k$,

- $(s,\dot{b}_i,\text{right},s')$ if $u_i \in [s,\dot{b}_i,\text{right},s']$;
- $(s,\dot{b}_i,\text{rstay},s')$ if $u_i \in [s,\dot{b}_i,\text{rstay},s']$;
- $(s,\dot{b}_{i+1},\text{left},s')$ if $u_i \in [s,\dot{b}_{i+1},\text{left},s']$;
- $(s,\dot{b}_{i+1},\text{lstay},s')$ if $u_i \in [s,\dot{b}_{i+1},\text{lstay},s']$.

Notice that T_w defines the transition table for a two-way finite automaton, also denoted by T_w, working on the input B (of fixed length k). The initial and accepting states of T_w are the same as those in M. Note that, by the construction of T_w,

> (*) the accepting runs of M on w, sampled at positions corresponding to the marked symbols $\dot{b}_1, \cdots, \dot{b}_k$, are exactly the accepting runs, denoted by $\text{run}(T_w)$, of T_w on word B.

Also note that there are only finitely many B's (since its length is a constant k) and, by definition of T_w, there are only finitely many distinct T_w's. Using the statement in (*), we have the following brute-force procedure to compute $L_{\text{run},\mathbf{c}(M)}$.

We fix a $B = \dot{b}_1 \cdots \dot{b}_k$ and consider a Boolean function $\mathcal{B}$ which gives Boolean values (true or false) $\mathcal{B}(s,\dot{b}_i,\text{right},s')$, $\mathcal{B}(s,\dot{b}_i,\text{rstay},s')$, $\mathcal{B}(s,\dot{b}_{i+1},\text{left},s')$, and $\mathcal{B}(s,\dot{b}_{i+1},\text{lstay},s')$, for all $1 \le i < k$ and all states s and s'. There are only finitely many such distinct Boolean functions. Each such Boolean function $\mathcal{B}$ defines a two-way finite automaton $T_{\mathcal{B}}$ working on B, whose transitions are exactly those t that make $\mathcal{B}(t)$ true. Clearly, not every $\mathcal{B}$ is valid.

More precisely, $\mathcal{B}$ is valid wrt B if there is a marked word $w = \dot{b}_1 u_1 \cdots u_k \dot{b}_k$ such that:

- The marked symbols in w are the sampling positions defined by $\mathbf{c}$. That is, for all $1 \le i < k$,
$$\max(1, \lfloor c_i |w| \rfloor) = 1 + \Sigma_{j=1}^{i-1}(|u_j| + 1). \tag{4.3}$$
- w is *consistent*; i.e., for all states s and s',
  - $\mathcal{B}(s,\dot{b}_i,\text{right},s')$ iff $u_i \in [s,\dot{b}_i,\text{right},s']$;
  - $\mathcal{B}(s,\dot{b}_i,\text{rstay},s')$ iff $u_i \in [s,\dot{b}_i,\text{rstay},s']$;
  - $\mathcal{B}(s,\dot{b}_{i+1},\text{left},s')$ iff $u_i \in [s,\dot{b}_{i+1},\text{left},s']$;
  - $\mathcal{B}(s,\dot{b}_{i+1},\text{lstay},s')$ iff $u_i \in [s,\dot{b}_{i+1},\text{lstay},s']$.

Using the aforementioned statement (*), we have

$$L_{\text{run},\mathbf{c}(M)} = \bigcup_{B} \bigcup_{\substack{\mathcal{B} \text{ is valid} \\ \text{wrt } B}} \text{run}(T_{\mathcal{B}}). \tag{4.4}$$

Since each $\text{run}(T_{\mathcal{B}})$ is regular, the result follows after we show that it is decidable whether a given $\mathcal{B}$ is valid wrt a given B.

We now prove that the decidability indeed holds. Closely looking at the definition, $\mathcal{B}$ is valid wrt B iff the set, denoted by $L_{\mathbf{c}}$, of w satisfying (4.3) and the set, denoted by $L_{\mathcal{B}}$, of w being consistent, satisfy

$$L_{\mathbf{c}} \cap L_{\mathcal{B}} \neq \emptyset. \tag{4.5}$$

Notice that the condition in (4.3) is definable as a Presburger formula on the lengths of the u_i's in w (where the rational numbers in $\mathbf{c}$ are constants such as $\frac{3}{4}$). Hence, it is an exercise to construct a reversal-bounded NCM M' to accept $L_{\mathbf{c}}$. Also notice that $L_{\mathcal{B}}$ is a regular language. This is because, as we have mentioned earlier, sets like $[s, \dot{b}_i, \text{right}, s']$ in defining the language are regular. Hence, the intersection $L_{\mathbf{c}} \cap L_{\mathcal{B}}$ in (4.5) is again accepted by a reversal-bounded NCM denoted by M'', whose emptiness is known decidable [14]. □

Now, we turn back to the traces of a 2NFA M. In M, the $\mathbf{c}$-*sampling information dependency* between A and B is defined as $D(A;B|L_{\text{run},\mathbf{c}(M)})$, and the $\mathbf{c}$-*sampling information flow* from A to B is defined as $F(A;B|L_{\text{run},\mathbf{c}(M)})$. Clearly, from Theorem 2 and Theorem 1, both $D(A;B|L_{\text{run},\mathbf{c}(M)})$ and $F(A;B|L_{\text{run},\mathbf{c}(M)})$ are computable from the given $\mathbf{c}$ and M.

We now consider the following *maximal sampling problem* using position sampling:

Given: a 2NFA M, a variable set A in M, and a number $k \geq 1$.

Goal: Find a k-arity vector $\mathbf{c}$ such that the information rate of $L^{A}_{\text{run},\mathbf{c}(M)}$ is maximal (among all choices of $\mathbf{c}$).

The maximal sampling problem is to find a best set of k positions in order to maximally maintain the bit rate for the given set of (observable) state variables. Similarly, the *maximal information dependency (*resp. *information flow) problem using position sampling* is as follows:

Given: a 2NFA M, two disjoint variable sets A and B in M, and a number $k \geq 1$.

Goal: Find a k-arity vector $\mathbf{c}$ such that $D(A;B|L_{\text{run},\mathbf{c}(M)})$
(resp. $F(A;B|L_{\text{run},\mathbf{c}(M)})$) is maximal (among all choices of $\mathbf{c}$).

Intuitively, the goal here is to find best input positions (for the given k) in order to observe the maximal dependency or information flow. All three problems are solvable.

Theorem 3. *For 2NFAs, maximal sampling, maximal information dependency and maximal information flow problems using position sampling are solvable.*

Proof. (sketch.) Let M be a 2NFA. Closely inspecting the proof of Theorem 2, there are only finitely many distinct regular languages $L_{\text{run},\mathbf{c}(M)}$, as shown in (4.4), whereas there are infinitely many choices of sampling positions **c**. The result follows. □

Observe that the set of sampling positions specified in **c** is a linear function of the input length, while there are no nontrivial relationships between these positions. We now consider cases where the k sampling positions are constrained. For an input of length n, let $0 \leq p_1, \cdots, p_k < n$ be the k sampling positions on the input. Let $\mathbf{c}(p_1, \cdots, p_k, n)$ be a Presburger formula to specify a *sampling position constraint* satisfying the following condition that $(p_1, \cdots, p_k)$ is a (not necessarily total) function of n:

> For all n, there are no $(p_1, \cdots, p_k) \neq (p'_1, \cdots, p'_k)$ such that $\mathbf{c}(p_1, \cdots, p_k, n)$ and $\mathbf{c}(p'_1, \cdots, p'_k, n)$ hold.

(The condition can be verified algorithmically for this **c** since the condition itself is a closed Presburger formula.) Hence, once the input length is given, the sampling positions, if they exist, are unique, hence they depend only on the length. For instance, such a **c** can be the following constraint

$$p_1 = p_2 - p_1 = \cdots = p_k - p_{k-1} = n - p_k - 1$$

indicating that all sampling positions are distributed evenly on the input. For notational convenience, we abuse **c** as a Presburger formula here so that the previous definitions can be reused. For instance, for the 2NFA M, $L_{\text{run},\mathbf{c}(M)}$ still denotes the set of all **c**-sampling (at positions $p_1, \cdots, p_k$ satisfying **c**) accepting runs for all inputs. We can show Theorem 2 can be generalized.

Theorem 4. *Let M be a 2NFA and* **c** *be a Presburger sampling position constraint. Then, $L_{\text{run},\mathbf{c}(M)}$ is a regular language.*

Proof. (sketch.) Let M be a 2NFA. Similar to the proof of Theorem 3, we look at the proof of Theorem 2: there are only finitely many distinct regular languages $L_{\text{run},\mathbf{c}(M)}$, as shown in (4.4), for infinitely many choices of (rational) sampling positions **c**. We use $L_1, \cdots, L_m$ to denote these languages. In particular, all the **c**'s that make $L_{\text{run},\mathbf{c}(M)} = L_q$ for a given q form a constraint written $Q(\mathbf{c})$. For an input of length n, let $0 \leq p_1, \cdots, p_k < n$ be the k sampling positions on the input. We use p_i to denote $\Sigma_{j=1}^{i-1}(|u_j| + 1)$ in (4.3) and n to replace $|w|$, the length of input. It is not hard to follow from the proof of Theorem 2 that all $(p_1, \cdots, P_k, n)$ satisfying the replaced (4.3) where the rational **c** satisfy the $Q(\mathbf{c})$ form a Presburger formula $P_q(p_1, \cdots, p_k, n)$. For the given Presburger sampling position constraint $\mathbf{c}(p_1, \cdots, p_k, n)$, the resulting $L_{\text{run},\mathbf{c}(M)}$ is simply the union of all regular languages L_q with $P_q(p_1, \cdots, p_k, n) \wedge \mathbf{c}(p_1, \cdots, p_k, n)$ being satisfiable. The satisfiability is decidable and the result follows. □

To formulate the maximal sampling problem, we further abuse the Presburger formula **c** as follows. Let $\mathbf{c}(p_1, \cdots, p_k, n; t_1, \cdots, t_m)$ be a Presburger formula, where $p_1, \cdots, p_k$ are sampling positions, and $t_1, \cdots, t_m$ are parameters. This **c** further satisfies a condition such that, for any given $t_1, \cdots, t_m$, $(p_1, \cdots, p_k)$ is a (not necessarily total) function of n. Again, such a condition can be verified algorithmically for this $\mathbf{c}$. Notice that $L_{\text{run},\mathbf{c}(M)}$ depends on the values of the parameters. Now, we define the *maximal sampling problem* using Presburger position sampling:

Given: a 2NFA M, a variable set A in M, and a Presburger sampling constraint $\mathbf{c}$ with parameters $t_1,\cdots,t_m$.

Goal: Find values for parameters $t_1,\cdots,t_m$ such that the information rate of $L^A_{\text{run},\mathbf{c}(M)}$ is maximal (among all choices of the values of the parameters).

Similarly, the *maximal information dependency (*resp. *information flow) problem using Presburger position sampling* is as follows:

Given: a 2NFA M, two disjoint variable sets A and B in M, and a Presburger sampling constraint $\mathbf{c}$ with parameters $t_1,\cdots,t_m$.

Goal: Find values for parameters $t_1,\cdots,t_m$ such that $D(A;B|L_{\text{run},\mathbf{c}(M)})$ (resp. $F(A;B|L_{\text{run},\mathbf{c}(M)})$) is maximal (among all choices of the values of the parameters).

Theorem 3 can be generalized.

Theorem 5. *For 2NFAs, maximal sampling, maximal information dependency and maximal information flow problems using Presburger position sampling are solvable.*

Proof. (sketch.) Let M be a 2NFA. From the proof of Theorem 4, for the given Presburger sampling position constraint $\mathbf{c}(p_1,\cdots,p_k,n;t_1,\cdots,t_m)$, for each given $(t_1,\cdots,t_m)$, the resulting $L_{\text{run},\mathbf{c}(M)}$ is simply the union of all regular languages L_q with $P_q(p_1,\cdots,p_k,n)\wedge\mathbf{c}(p_1,\cdots,p_k,n;t_1,\cdots,t_m)$ being satisfiable. Notice that there are only finitely many choices of distinct $L_{\text{run},\mathbf{c}(M)}$ for all $(t_1,\cdots,t_m)$. The results follows. □

4.3 Language Properties of Sampled Runs of Some Two-Way Infinite-State Automata

In this section, we show that position sampling may result in quite complex languages when the 2NFA is augmented with unbounded storage. We start with finite crossing-finite automata augmented with reversal-bounded counters [14]. Finite-crossing here means that there is some fixed k such that in every accepting computation on any input, the number of times the input head crosses the boundary between any two adjacent symbols of the input is at most k. Note that the number of turns (i.e., changes in direction from left-to-right and right-to-left and vice-versa) the input head makes on the input may be unbounded. We assume that when we are given a finite-crossing machine, the integer k for which the machine is k-crossing is also specified.

Let $C(M)$ denote the crossing sequences at crossing location c, where the symbol at location c is marked (e.g., at the beginning, end, or middle of the input). Notice that the sampled run at this location is essentially the crossing sequence. We assume here that the machine M "knows" the sampling position (since it is marked). The following summarizes our results (the proofs will appear in the journal version of this paper):

- It is decidable, given a two-way DFA with one reversal-bounded counter (no restriction on the the two-way input) M and a regular language L, whether $C(M) \cap L$ is empty.
- It is undecidable, given a two-way DFA with two reversal-bounded counters (no restriction on the the two-way input) M and a regular language L, whether $C(M) \cap L$ is empty.
- It is decidable, given a finite-crossing two-way NFA with multiple reversal-bounded counters M and a regular language L, whether $C(M) \cap L$ is empty.

We now consider the case where the marked symbols on an input are unbounded. Let Σ_1 be an alphabet and $\Sigma_2 = \{a' \mid a \in \Sigma_1\}$. Thus, the symbols in Σ_2 are the "marked" symbols in Σ_1. Let $\Sigma = \Sigma_1 \cup \Sigma_2$.

Let M be a two-way acceptor (with end markers) possibly with infinite storage over the input alphabet Σ. We assume that M on input w in Σ "ignores" the marks, i.e., treat symbols a and a' in the same way.

Let M_c be another acceptor over Σ (one-way or two-way, possibly with infinite storage) over Σ. This time in its computation, M_c distinguishes the marked symbols from the unmarked symbols. Thus, e.g., on input w, M_c accepts if every other symbol is marked (or the number of marked symbols is even; etc.). Intuitively, this M_c specifies where the marked symbols are on an input to M.

Let L be a language over sequences of states of M (specified by another acceptor M_L). Let $C(M)$ be the set of crossing sequences of states that are encountered when M crosses a marked symbol on the input.

We can show the following: It is decidable, given a finite-crossing two-way NFA with reversal-bounded counters M, a two-way finite-crossing NFA with reversal-bounded counters M_c, and an NPDA M_L with reversal-bounded counters accepting L, whether $C(M) \cap L$ is empty. (The result also holds when M_L is a finite-crossing two-way NFA with reversal-bounded counters.)

A special case of the above is the following, which can be simply observed. If M is a one-way NFA and M_c (which specifies the "marked" locations) is an NFA, then $C(M)$ can be accepted by a one-way NFA (and hence it is regular).

There is a well-studied acceptor, called a checking stack automaton (CSA), introduced by Ginsburg *et al.* [10]. A CSA is a one-way NFA with a special stack. The special stack is like a pushdown stack: it can write but not pop, but it can enter the stack in a two-way read-only mode. However, once it enters the stack, it can no longer write, but it can continue using the stack in a read-only mode. Recall that, for a 2NFA M , the set (language) $L_{run,M}$ is not necessarily regular. We can show that $L_{run,M}$ is accepted by a CSA.

Finally, consider the case when there are at most k crossing positions, for a constant k. That is, every marked input accepted by M_c contains at most k marked symbols. We can show the following results:

- If M is a two-way NFA and M_c is an NFA, then $C(M)$ is accepted by a two-way NFA which can be converted to a one-way NFA, so it is also regular.

- If M is a one-way NCM (i.e., NFA augmented with 1-reversal counters) and M_c is an NPDA with reversal-bounded counters, then $C(M)$ is accepted by an NPDA with reverse-bounded counters.
- If M and M_c are both finite-crossing NCMs, then $C(M)$ is accepted by a finite-crossing NCM, hence it can also be accepted by a one-way NCM [13].

4.4 Conclusions

In this paper, we have studied position sampling in a 2-way nondeterministic finite automaton (2NFA) to measure the information dependency and information flow between state variables, based on the information-theoretic sampling technique proposed in [16]. We have proved that for a 2NFA, the language generated by position sampling is regular. We have also showed that for a 2NFA, we can effectively find a vector of sampling positions that maximizes dependency and information flow in a run of the 2NFA. Finally, we have summarized some language properties of sampled runs of 2NFAs augmented with restricted unbounded storage.

In the future, we will conduct experiments and show the practical usefulness of information rates resulting from sampling a two-way automaton.

Acknowledgment. We would like to thank Eric Wang for comments that improved the presentation of our results.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Cavadini, S.: Secure slices of insecure programs. In: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008, pp. 112–122 (2008)
3. Chomsky, N., Miller, G.A.: Finite state languages. Information and Control 1, 91–112 (1958)
4. Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security 15 (2007)
5. Cui, C., Dang, Z.: Fischer: Bit rate of programs (submitted)
6. Cui, C., Dang, Z., Fischer, T., Ibarra, O.: Similarity in languages and programs. Theoretical Computer Science 498, 58–75 (2013)
7. Cui, C., Dang, Z., Fischer, T.R., Ibarra, O.H.: Execution information rate for some classes of automata. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 226–237. Springer, Heidelberg (2013)
8. Dang, Z.: Pushdown timed automata: a binary reachability characterization and safety verification. Theoretical Computer Science 302(1-3), 93–121 (2003)
9. Dang, Z., Ibarra, O., Bultan, T., Kemmerer, R., Su, J.: Binary reachability analysis of discrete pushdown timed automata. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 69–84. Springer, Heidelberg (2000)
10. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack automata. Journal of the ACM (JACM) 14(2), 389–418 (1967)

11. Goguen, J.A., Meseguer, J.: Security Policies and Security Models. In: IEEE Symposium on Security and Privacy, pp. 11–20 (1982)
12. Gray I, J.W.: Toward a mathematical foundation for information flow security. In: Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 21–34 (May 1991)
13. Gurari, E.M., Ibarra, O.H.: The complexity of decision problems for finite-turn multicounter machines. Journal of Computer and System Sciences 22(2), 220–229 (1981)
14. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. ACM 25(1), 116–133 (1978)
15. Ibarra, O.H., Dang, Z., Egecioglu, O., Saxena, G.: Characterizations of catalytic membrane computing systems. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 480–489. Springer, Heidelberg (2003)
16. Li, Q., Dang, Z.: Sampling automata and programs (submitted)
17. McLean, J.: Security models and information flow. In: Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 180–187 (May 1990)
18. McCamant, S., Ernst, M.D.: Quantitative information flow as network flow capacity. SIGPLAN Not. 43(6), 193–205 (2008)
19. McIver, A., Morgan, C.: A probabilistic approach to information hiding. In: McIver, A., Morgan, C. (eds.) Programming Methodology. Monographs in Computer Science, pp. 441–460. Springer, New York (2003)
20. Millen, J.K.: Covert channel capacity. In: IEEE Symposium on Security and Privacy, Oakland, CA, pp. 60–66 (1987)
21. Paun, G.: Membrane Computing, an Introduction. Springer, Heidelberg (2000)
22. Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. University of Illinois Press (1949)
23. Smith, G., Volpano, D.: Secure information flow in a multi-threaded imperative language. In: Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 1998, pp. 355–364. ACM, New York (1998)
24. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)
25. Terauchi, T.: A type system for observational determinism. In: IEEE 21st Computer Security Foundations Symposium, CSF 2008, pp. 287–300 (June 2008)
26. Yang, L., Cui, C., Dang, Z., Fischer, T.R.: An information-theoretic complexity metric for labeled graphs (submitted)
27. Wang, E., Cui, C., Dang, Z., Fischer, T.R., Yang, L.: Zero-knowledge blackbox testing: where are the faults? International Journal of Foundations of Computer Science (to appear)
28. Weiser, M.: Program Slicing. IEEE Transactions on Software Engineering 10, 352–357 (1984)
29. Xie, G., Dang, Z., Ibarra, O.H.: A solvable class of quadratic diophantine equations with applications to verification of infinite-state systems. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 668–680. Springer, Heidelberg (2003)
30. Xu, B., Qian, J., Zhang, X., Wu, Z., Chen, L.: A brief survey of program slicing. ACM Sigsoft Software Engineering Notes 30, 1–36 (2005)

Chapter 5
Maurice Margenstern's Contributions to the Field of Small Universal Turing Machines

Turlough Neary[*] and Damien Woods[**]

Abstract. On the occasion of his 65th birthday we survey some of the work of Maurice Margenstern on small universal Turing machines. Margenstern has been one of the most prolific contributors to this field, having constructed numerous small universal programs for a number of Turing machine models. These positive results are complemented by Margenstern's negative results, or lower bounds, on universal program size. Finally, he has even explored the space in-between the known program size upper and lower bounds by giving small programs that iterate the Collatz function, which suggests that proving negative results on programs of this size will be at least as hard as resolving the Collatz problem.

5.1 Introduction

The topic of small universal Turing machines is concerned with putting upper and lower bounds on the program size of universal Turing machines. In other words, finding the shortest programs that are capable of universal computation and showing that below this threshold such complicated behaviour is impossible. More generally, the computational complexity of small programs includes investigating the time, space and encoding complexity costs we pay by having tiny general-purpose programs. Simulation of small universal Turing machines and other simple univer-

Turlough Neary
Institute for Neuroinformatics, University of Zürich and ETH Zürich, Switzerland
e-mail: tneary@ini.phys.ethz.ch

Damien Woods
California Institute of Technology, Pasadena, CA 91125, USA
e-mail: woods@caltech.edu

[*] Supported by Swiss National Science Foundation grant 200021-141029.

[**] Supported by the US National Aeronautics and Space Administration (NASA) grant NNX13AJ56G, and National Science Foundation (NSF) grants 0832824 & 1317694 (The Molecular Programming Project), CCF-1219274 and CCF-1162589.

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_5

sal models such as Post's tag systems [36] and the cellular automaton rule 110 [4] is by now a standard way to prove that a large number of other models of computation, including a variety of physically-inspired systems, are computationally universal.

Maurice Margenstern has been one of the most prolific authors in the field. He has constructed numerous small universal Turing machines, for a variety of Turing machine models, and has complemented this by giving lower bounds on the size of universal Turing machine programs. As Figure 5.1 shows, although we have upper and lower bounds on universal program size there remain a large number of intermediate program sizes for which we don't know whether or not there are universal programs. Margenstern introduced an interesting way to explore this space of programs by giving intermediate-size programs that simulate iterations of the Collatz function [8]. It follows that solving many questions about the long term dynamics of these machines is at least as hard as resolving the Collatz problem.

Margenstern has been a great proponent and organiser for the field, mainly by his technical contributions and analysis of novel syntactical program properties other than size, but also by other means: his survey [20] was an excellent introduction to the field for us, and the international conference series he started, *Machine Computations and Universality*, has been a welcoming home for new results. In this short note we highlight a small selection of Margenstern's work. More of his and others' results can be found in a number of surveys, including those by Margenstern [17, 19, 20, 25, 26, 32].

In Section 5.2 we discuss Margenstern's small Collatz simulators and where they lie in relation to the best-known upper and lower bounds on universal program size for the standard Turing machine model. In Section 5.3 we give Margenstern's upper and lower bounds results for a number of Turing machine models. Finally, in Section 5.4 we discuss some of Margenstern's universal machines for restrictions of the standard Turing machine model and compare his algorithm for simulating 2-tag systems to the original algorithm introduced by Minsky [28].

5.2 Simulating the Collatz Function with Small Turing Machines

Historically much of the research into small universal Turing machines is concerned with the deterministic single-tape model with the usual notion of blank symbol. We often refer to this as the *standard model*. The size of a program is defined as the number of its states and symbols written as a (state, symbol) pair. Their product gives an upper bound on the number of instructions in the program.

For the standard model, the smallest known Turing machines were given by Rogozhin [38] ($(2,18)$, $(4,6)$, and $(5,5)$), Kudlek and Rogozhin [7] $(3,9)$, and Neary and Woods [31] ($(5,5)$, $(6,4)$, $(9,3)$ and $(15,2)$). These machines are plotted as circles in Figure 5.1. The halting problem has been shown to be decidable for the following state-symbol pairs: $(2,2)$ [6, 33], $(3,2)$ [35], $(2,3)$ (Pavlotskaya, unpublished), $(1,n)$ [5] and $(n,1)$ (trivial) for $n \geqslant 1$. Thus there are no universal machines

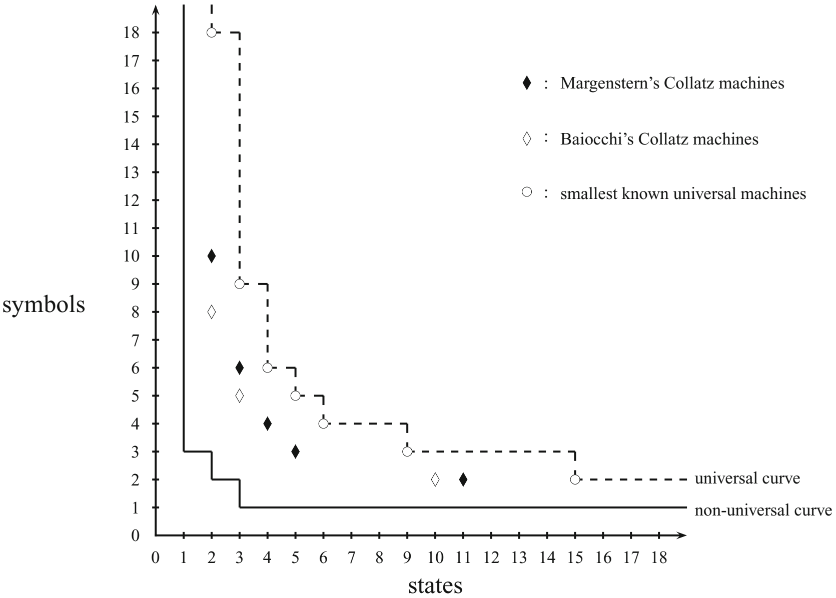


Fig. 5.1 State-symbol plot of the smallest known universal Turing machines for the standard model. These machines induce the universal curve. The non-universal curve shows Turing machines that are known to have a decidable halting problem. Margenstern's machines that simulate iterations of the Collatz function are shown as solid diamonds.

of this size that have the property of halting exactly when the simulated Turing machine halts. These results are plotted as the "non-universal" curve in Figure 5.1.

The space between the universal and non-universal curves in Figure 5.1 contains 39 state-symbols pairs for which the universality/non-universality question remains open. Margenstern [19, 20] found an interesting way to explore this space of open questions. He constructed machines with state-symbol pairs of $(11,2)$, $(5,3)$, $(4,4)$, $(3,6)$ and $(2,10)$ that simulate iterations of the Collatz function and are shown as solid diamonds in Figure 5.1. The Collatz function [8] is

$$f(x) = \begin{cases} 3x+1 & \text{if } x = 1 \bmod 2, \\ x/2 & \text{if } x = 0 \bmod 2. \end{cases} \tag{5.1}$$

Associated with this function is the well-known question: for each $x \in \mathbb{N}$ is there an n such that $f^n(x) = 1$? The answer is conjectured to be yes. The question has drawn the interest of many authors and has resisted all attempts at a solution [8–10]. So for the class of machines with program size equal (or greater than) Margenstern's Collatz function simulators the following problem is at least as difficult as solving the Collatz conjecture: give an algorithm that takes one of these machines, an initial configuration and a target configuration as input, and decides whether or not the machine reaches the target from the initial configuration.

Baiocchi [1] reduced the size of some of these Collatz simulators to give Turing machines with state-symbol pairs of $(10,2)$, $(5,3)$, $(4,4)$, $(3,5)$ and $(2,8)$. All of Baiocchi's machines and the $(4,4)$, $(3,6)$ and $(2,10)$ machines of Margenstern use a unary encoding and so simulate a single iteration of the Collatz function on x in time $O(x)$. The $(11,2)$ and $(5,3)$ machines of Margenstern use a binary encoding and so simulate a single iteration of the Collatz function x in time $O(\log x)$. It would be of interest to see if Margenstern's efficient encoding could be extended to other state-symbol pairs as a way to explore possible trade-offs between program size and time/space efficiency, even for these non-universal machines. For more on research in this direction see [30, 32, 40].

5.3 Frontiers between Universality and Non-universality

Margenstern has found matching upper and lower bounds (which he calls a *frontier*) on various syntactic properties of universal 2-symbol Turing machine programs. So although we are currently unable to find matching upper and lower bounds on the particular syntactic property of universal program size, his work shows us that there are some syntactic properties for which we can find such frontiers.

One such property is the number of *colours* of a 2-symbol machine, defined as the number of distinct triples (α, D, δ), where α is the read symbol, D is the move direction, and δ is the write symbol of a transition rule. Pavlotskaya [33, 34] has shown that there are standard universal Turing machines with 3 colours and no standard universal Turing machines with 2 colours. Margenstern [13] has shown that there are non-erasing universal Turing machines with 5 colours and no non-erasing universal Turing machines with 4 colours.

Laterality number is another property examined by Margenstern and is defined as the minimum of the number of left-move instructions and the number of right-move instructions of a 2-symbol Turing machine. Margenstern and Pavlotskaya [22, 33] have shown that there are universal Turing machines with laterality number 2 and no universal machines with laterality number 1. Margenstern [15, 18] has shown that there are universal non-erasing Turing machines with laterality number 3 and no universal non-erasing machines with laterality number 2. For more on these results see [11, 13–16, 21].

So far in this section we have considered Turing machines that are restrictions on the standard model. Margenstern and Pavlotskaya [24] have also considered a generalisation of the standard model that consists of a (Turing machine, finite automaton) pair. Their machine operates as follows: The Turing machine has a one-way infinite tape and operates in the usual way until it moves onto a tape cell that is blank, i.e. a cell that did not contain a symbol from the initial input and has not previously been visited by the tape head. When it moves onto the blank cell a symbol is written to that cell by the finite automaton that depends on the current states of both the automaton and Turing machine. After a symbol is written by the finite automaton the Turing machine continues to compute in the normal manner until the next blank cell is entered. Margenstern and Pavlotskaya [23, 24] gave a 2-state, 3-symbol Turing machine that uses only 5 instructions and is universal when coupled with a finite automaton as described above. They also showed, via a 58-page proof, that the halting problem is decidable for such machines with 4 instructions [24], giving us another frontier.

5.4 Restricted Turing Machines

We can suitably restrict the standard Turing machine model so that the problem of finding universal machines with small state-symbol products becomes increasingly difficult or even impossible. For example, Margenstern [13, 16, 18, 21] and others before him [27, 29, 39, 41] have looked at non-erasing Turing machines, that is machines that are permitted to only overwrite blank symbols, other symbols can not be changed. As discussed in Section 5.3, Margenstern has also given universal machines for other restrictions of the standard model and in Section 5.4.1 we look at details of the simulation algorithms used by some of these machines.

5.4.1 Margenstern's Tag System Simulation Algorithms

Like many authors, Margenstern used the technique of 2-tag system simulation in some of his small universal Turing machines. Tag systems where introduced by Post [36], and 2-tag systems were shown to be universal by Cocke and Minsky [3]. Minsky [28] constructed a 7-state, 4-symbol universal Turing machine that simulates 2-tag systems and his machine's simulation algorithm was the inspiration for many of the small universal machines to follow [2, 7, 37, 38]. Below, we introduce 2-tag systems and describe Minsky's simulation algorithm, and we then discuss why Margenstern had to come up with a completely new simulation algorithm for his restricted machine model. This is followed by an overview of the operation of Margenstern's 2-symbol, 59-state machine with 6 left-move instructions, and his 2-symbol, 190-state machine with 3 left-move instructions, both of which simulate 2-tag systems. Margenstern gives an overview of these machines in [21], with more details in [12, 14, 16].

Definition 1 (2-tag system). A tag system consists of a finite alphabet of symbols $\Sigma = \{\sigma_1, \sigma_2, \ldots \sigma_l\}$, a finite set of rules of the form $\sigma \rightarrow P(\sigma)$ with $P(\sigma) \in \Sigma^*$, and a deletion number $\beta \in \mathbb{N}$, $\beta \geqslant 1$. For a 2-tag system, $\beta = 2$.

A 2-tag system acts on a dataword $w = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_n}$. The entire configuration is given by w. In a computation step, the first two symbols $\sigma_{i_1}\sigma_{i_2}$ of w are deleted and the rule $\sigma_{i_1} \rightarrow P(\sigma_{i_1})$ is applied by appending the word $P(\sigma_{i_1})$ to the right of w:

$$\sigma_{i_1}\sigma_{i_2}\sigma_{i_3}\ldots\sigma_{i_n} \vdash \sigma_{i_3}\ldots\sigma_{i_n}P(\sigma_{i_1})$$

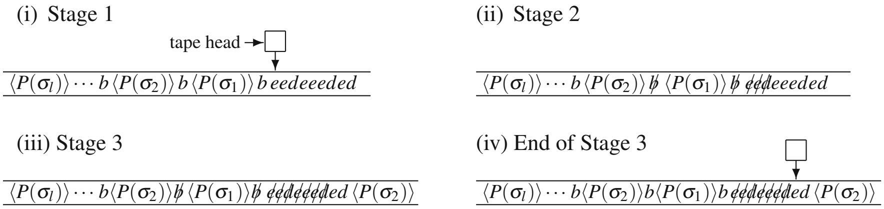

(i) Stage 1
tape head
$\langle P(\sigma_l)\rangle \cdots b\langle P(\sigma_2)\rangle b\langle P(\sigma_1)\rangle b\, eedeeeded$
(ii) Stage 2
$\langle P(\sigma_l)\rangle \cdots b\langle P(\sigma_2)\rangle \not{b}\langle P(\sigma_1)\rangle \not{b}\, \not{e}\not{e}\not{d}eeeded$
(iii) Stage 3
$\langle P(\sigma_l)\rangle \cdots b\langle P(\sigma_2)\rangle \not{b}\langle P(\sigma_1)\rangle \not{b}\, \not{e}\not{e}\not{d}\not{e}\not{e}\not{e}\not{d}ed\langle P(\sigma_2)\rangle$
(iv) End of Stage 3
$\langle P(\sigma_l)\rangle \cdots b\langle P(\sigma_2)\rangle b\langle P(\sigma_1)\rangle b\, \not{e}\not{e}\not{d}\not{e}\not{e}\not{e}\not{d}ed\langle P(\sigma_2)\rangle$

Fig. 5.2 An overview of Minsky's Turing machine algorithm [28] to simulate an arbitrary step of a tag system $\sigma_2\sigma_3\sigma_1 \vdash \sigma_1 P(\sigma_2)$ where σ_2 is read, $\sigma_2\sigma_3$ is deleted, production $P(\sigma_2)$ is appended and σ_i is encoded in unary as the string $e^i d$. (i) Initial configuration of Stage 1 (from here, locate production $P(\sigma_2)$ by marking off, and thus deleting, the first tag system symbol σ_2 encoded as *eed*, as well as 2 b symbols). (ii) Initial configuration of Stage 2 (print production $P(\sigma_2)$ and delete the second encoded tag system symbol σ_3 encoded as *eeed*). (iii) Initial configuration of Stage 3 (unmark the 2 b symbols that were used to index the production $P(\sigma_2)$). (iv) Simulation of tag system step complete: two encoded symbols have been deleted, a new one appended, and we are reading the symbol σ_1 encoded as *ed*.

Figure 5.2 gives an overview of a simple algorithm for simulating a single tag system step $\sigma_2\sigma_3\sigma_1 \vdash \sigma_1 P(\sigma_2)$. This was introduced by Minsky [28] and lends itself well to implementation via small programs. Figure 5.2(i) gives the initial Turing machine tape contents. On the left side of Turing machine tape the word $\langle P(\sigma_l)\rangle \cdots b\langle P(\sigma_2)\rangle b\langle P(\sigma_1)\rangle b$ encodes the list of productions $P(\sigma_l)\cdots P(\sigma_2)P(\sigma_1)$, where $\langle P(\sigma_i)\rangle$ is the encoding of $P(\sigma_i)$ and b is a special marker symbol that separates adjacent productions. On the right side of the tape is the word *eedeeeded* which encodes the dataword $\sigma_2\sigma_3\sigma_1$, where each symbol σ_i is encoded as the sequence of Turing machine tape symbols $e^i d$.

The algorithm in Figure 5.2 has 3 stages. Stage 1 begins with the configuration shown in Figure 5.2(i) and uses the encoding *eed* of the leftmost symbol σ_2 as a unary index to locate the encoded production $\langle P(\sigma_2)\rangle$. During this stage the tape head repeatedly scans left and right marking off one b marker for each e in the encoding of σ_2. Stage 2 simulates the deletion of the second symbol σ_3 by marking off *eeed*, and then repeatedly scans left and right copying the encoded production $\langle P(\sigma_2)\rangle$ and printing it at the right end of the encoded dataword. Finally, Stage 3 unmarks the list of encoded productions.

Even though the algorithm given in Figure 5.2 for simulating 2-tag systems allows for the construction of some of the smallest known universal Turing machines, it still uses left and right scans of the tape in a number of different contexts (which costs states and symbols). During Stage 1 the head scans left to mark off b's, during Stage 2 it scans left to copy the symbols of $\langle P(\sigma)\rangle$ to be printed at the right end of the tape, and finally in Stage 3 it scans left to restore (unmark) the original encoding of the tag system program. In addition to this, the information in e, d, b, $\not{b}$ and $\langle P(\sigma)\rangle$ words must not be destroyed during these leftward scans which adds to the number of left-moving transition rules required. On a 2-symbol Turing machine one would expect the number of left-move instruction to further increase as e, d, b and $\not{b}$ would be encoded as binary words. The smallest known 2-symbol universal Turing

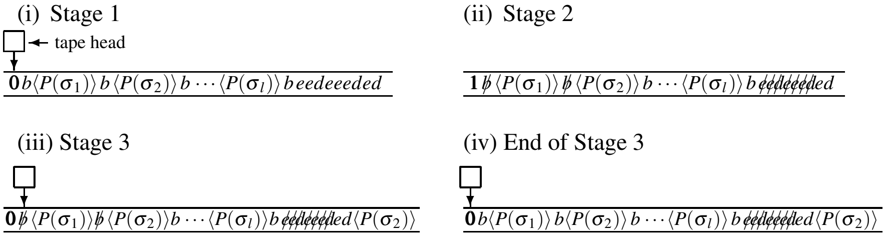

(i) Stage 1
tape head
$\mathbf{0}b\langle P(\sigma_1)\rangle b\langle P(\sigma_2)\rangle b\cdots\langle P(\sigma_l)\rangle beedeeeded$
(ii) Stage 2
$\mathbf{1}\cancel{b}\langle P(\sigma_1)\rangle\cancel{b}\langle P(\sigma_2)\rangle b\cdots\langle P(\sigma_l)\rangle b\cancel{e}\cancel{e}\cancel{d}\cancel{e}\cancel{e}\cancel{e}\cancel{e}\cancel{d}ed$
(iii) Stage 3
$\mathbf{0}\cancel{b}\langle P(\sigma_1)\rangle\cancel{b}\langle P(\sigma_2)\rangle b\cdots\langle P(\sigma_l)\rangle b\cancel{e}\cancel{e}\cancel{d}\cancel{e}\cancel{e}\cancel{e}\cancel{e}\cancel{d}ed\langle P(\sigma_2)\rangle$
(iv) End of Stage 3
$\mathbf{0}b\langle P(\sigma_1)\rangle b\langle P(\sigma_2)\rangle b\cdots\langle P(\sigma_l)\rangle b\cancel{e}\cancel{e}\cancel{d}\cancel{e}\cancel{e}\cancel{e}\cancel{e}\cancel{d}ed\langle P(\sigma_2)\rangle$

Fig. 5.3 Margenstern's algorithm to simulate an arbitrary tag system step $\sigma_2\sigma_3\sigma_1 \vdash \sigma_1 P(\sigma_2)$, see main text for details.

machine [31] uses 15 left-move instructions. Surprisingly, Margenstern's managed to give 2-symbol universal Turing machines with very few left-move instructions: a Turing machine with 59 states and only 6 left-move instructions [21], another machine with 190 states and only 3 left-move instructions [12, 21], and a 218-state non-erasing machine with only 3 left-move instructions [16,21]. Margenstern found some elegant techniques to overcome a limited ability to carry information to the left.

For Margenstern's [21] 2-symbol, 59-state machine that uses 6 left-move instructions a simulated computation step begins with a tape of the form shown in Figure 5.3(i). Note that the order of the encoded productions in Figure 5.3 is the reverse of that in Figure 5.2. Also, there is a special *stage bit* (bold **0**) at the left end of the tape. This stage bit is used to record which of Stages 1 or 2 is currently being executed. The 6 left-move instructions are split into two disjoint sets of 3 instructions. The first set scans left when the stage bit should remain unchanged and the second set scans left when the stage bit should change. Beginning with the initial configuration shown in Figure 5.3(i) the tape head scans right and marks off the leftmost b on the tape and continues its scan right (leaving all subsequent b markers unchanged) until it marks off the leftmost e on the tape. It then scans left in the set of left-move rules that leaves the stage bit unchanged. Following this, there is another scan to the right to mark off another b and another e. This is repeated until a d is read during a scan right signaling that it has finished reading the leftmost encoded symbol in the encoded dataword. In this case it continues its scan right and deletes the next encoded tag system symbol, before scanning left in the set of states that change the stage bit (in this case from 0 to 1) causing the next stage (Stage 2) of the algorithm to begin. In Stage 2 repeated rightward and leftward scans copy the encoded production $\langle P(\sigma_2)\rangle$, which was located during Stage 1, to the right end of the encoded dataword. Each scan left occurs in the set of states that signal no change in the stage bit until the copying process is finished. After the encoded production has been entirely copied, it scans left in the set of states that signal a flip in the stage bit from 1 to 0. We are now in Stage 3, which ignores the stage bit: the machine carries out a single scan right to restore the encoded tag system program to its original value, then scans left in the set of sets that signal no change in the stage bit. The simulation of the computation step is complete and the machine goes to Stage 1. The machine is now ready to begin simulation of the next tag system step.

Margenstern used a similar algorithm in his 2-symbol, 190-state machine with 3 left-move instructions. However, the previous technique used for updating the stage bit cannot be used here. The reason for this is that with only 3 left-move instructions, in this particular setup, we can send only 1 type of signal to the left. To overcome this problem Margenstern's machine repeatedly copies the entire configuration to the right of its current location. Before a configuration is copied there are two words found at either end of the configuration, each of which records which stage is being executed (similar to the stage bit from above). When simulating that the stage bit is 0 the word at the left end is u and the word at the right end word is v, and when simulating that the stage bit is 1 the word at the left end is c and the word at the right end is k. After copying the entire configuration to the right of the rightmost word (v or k), this rightmost word is then positioned at left end of the newly copied configuration. During this process this word (v or k) is altered to become the new leftmost word (u or c) that encodes the stage. Note that as the configuration is copied to the right all of the information in the configuration is carried over what was originally the rightmost word and this allows it to be updated to record whether or not the encoding of the stage bit should change. Margenstern introduces some other techniques in this variant of his algorithm (such as the use of three simulated tape heads), which we do not discuss here. Margenstern's [21] non-erasing machine with 2 symbols, 218 states and only 3 left-move instructions uses a similar algorithm.

References

1. Baiocchi, C.: $3n+1$, UTM e tag-system. Tech. Rep. Pubblicazione 98/38, Dipartimento di Matematico, Università di Roma (1998) (in Italian)
2. Baiocchi, C.: Three small universal Turing machines. In: Margenstern, M., Rogozhin, Y. (eds.) MCU 2001. LNCS, vol. 2055, pp. 1–10. Springer, Heidelberg (2001)
3. Cocke, J., Minsky, M.: Universality of tag systems with $P = 2$. Journal of the Association for Computing Machinery (ACM) 11(1), 15–20 (1964)
4. Cook, M.: Universality in elementary cellular automata. Complex Systems 15(1), 1–40 (2004)
5. Hermann, G.T.: The uniform halting problem for generalized one state Turing machines. In: Proceedings of the Ninth Annual Symposium on Switching and Automata Theory (FOCS), pp. 368–372. IEEE Computer Society Press, Schenectady (1968)
6. Kudlek, M.: Small deterministic Turing machines. Theoretical Computer Science 168(2), 241–255 (1996)
7. Kudlek, M., Rogozhin, Y.: A universal Turing machine with 3 states and 9 symbols. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 311–318. Springer, Heidelberg (2002)
8. Lagarias, J.C.: The $3x+1$ problem and its generalizations. American Mathematical Monthly, 3–23 (1985)
9. Lagarias, J.C.: The $3x+1$ problem: An annotated bibliography (1963–1999). Tech. Rep. arXiv:math/0309224 [math.NT] (2003)
10. Lagarias, J.C.: The $3x+1$ problem: An annotated bibliography, II (2000-2009). Tech. Rep. arXiv:math/0608208 [math.NT] (2006)

11. Margenstern, M.: Sur la frontière entre machines de Turing á arrêt décidable et machines de Turing universelles. Tech. Rep. 92.83, LITP Institut Blaise Pascal (1992)
12. Margenstern, M.: Machine de Turing universelle sur $\{0,1\}$, à trois instructions gauches. Tech. Rep. 93.36, LITP Institut Blaise Pascal (1993)
13. Margenstern, M.: Non-erasing Turing machines: A frontier between a decidable halting problem and universality. In: Ésik, Z. (ed.) FCT 1993. LNCS, vol. 710, pp. 375–385. Springer, Heidelberg (1993)
14. Margenstern, M.: Une machine de Turing universelle sur $\{0,1\}$, non-effaçante et à trois instructions gauches. Tech. Rep. 94.08, LITP Institut Blaise Pascal (1994)
15. Margenstern, M.: Non-erasing Turing machines: A new frontier between a decidable halting problem and universality. In: Baeza-Yates, R.A., Goles, E., Poblete, P.V. (eds.) LATIN 1995. LNCS, vol. 911, pp. 386–397. Springer, Heidelberg (1995)
16. Margenstern, M.: Une machine de Turing universelle non-effaçante à exactement trois instructions gauches. CRAS, Paris 320(I), 101–106 (1995)
17. Margenstern, M.: Decidability and undecidability of the halting problem on Turing machines, a survey. In: Adian, S., Nerode, A. (eds.) LFCS 1997. LNCS, vol. 1234, pp. 226–236. Springer, Heidelberg (1997)
18. Margenstern, M.: The laterality problem for non-erasing Turing machines on $\{0,1\}$ is completely solved. Theoretical Informatics and Applications 31(2), 159–204 (1997)
19. Margenstern, M.: Frontier between decidability and undecidability: a survey. In: Margenstern, M. (ed.) Machines, Computations, and Universality (MCU), vol. 1, pp. 141–177. IUT, Metz (1998)
20. Margenstern, M.: Frontier between decidability and undecidability: a survey. Theoretical Computer Science 231(2), 217–251 (2000)
21. Margenstern, M.: On quasi-unilateral universal Turing machines. Theoretical Computer Science 257(1-2), 153–166 (2001)
22. Margenstern, M., Pavlotskaya, L.: Deux machines de Turing universelles á au plus deux instructions gauches. CRAS, Paris 320(I), 1395–1400 (1995)
23. Margenstern, M., Pavlotskaya, L.: Vers une nouvelle approche de l'universalité concernant les machines de Turing. Tech. Rep. 95.58, LITP Institut Blaise Pascal (1995)
24. Margenstern, M., Pavlotskaya, L.: On the optimal number of instructions for universal Turing machines connected with a finite automaton. International Journal of Algebra and Computation 13(2), 133–202 (2003)
25. Michel, P.: Small Turing machines and the generalized busy beaver competition. Theoretical Computer Science 326, 45–56 (2004)
26. Michel, P., Margenstern, M.: Generalized $3x+1$ Functions and the Theory of Computation. In: The Ultimate Challenge: The $3x+1$ Problem, pp. 105–130. American Mathematical Society (2010)
27. Minsky, M.: Recursive unsolvability of Post's problem of tag and other topics in theory of Turing machines. Annals of Mathematics 74(3), 437–455 (1961)
28. Minsky, M.: Size and structure of universal Turing machines using tag systems. In: Recursive Function Theory, Proceedings, Symposium in Pure Mathematics, vol. 5, pp. 229–238. American Mathematical Society, Provelence (1962)
29. Moore, E.F.: A simplified universal Turing machine. In: ACM National Meeting, pp. 50–54. ACM Press, Toronto (1952)
30. Neary, T., Woods, D.: Small fast universal Turing machines. Theoretical Computer Science 362(1-3), 171–195 (2006)
31. Neary, T., Woods, D.: Four small universal Turing machines. Fundamenta Informaticae 91(1), 123–144 (2009)

32. Neary, T., Woods, D.: The complexity of small universal Turing machines: a survey. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 385–405. Springer, Heidelberg (2012)
33. Pavlotskaya, L.: Solvability of the halting problem for certain classes of Turing machines. Mathematical Notes 13(6), 537–541 (1973); Translated from Matematicheskie Zametki 13(6), 899–909 (1973)
34. Pavlotskaya, L.: O minimal'nom chisle razlichnykh kodov vershin v grafe universal'noj mashiny T'juringa. Disketnyj Analiz, Sbornik Trudov Instituta Matematiki SO AN SSSR 27, 52–60 (1975); On the minimal number of distinct codes for the vertices of the graph of a universal Turing machine (in Russian)
35. Pavlotskaya, L.: Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring. Problemi Kibernetiki 27, 91–118 (1978); Sufficient conditions for the halting problem decidability of Turing machines (in Russian)
36. Post, E.: Formal reductions of the general combinatorial decision problem. American Journal of Mathmatics 65(2), 197–215 (1943)
37. Robinson, R.: Minsky's small universal Turing machine. International Journal of Mathematics 2(5), 551–562 (1991)
38. Rogozhin, Y.: Small universal Turing machines. Theoretical Computer Science 168(2), 215–240 (1996)
39. Wang, H.: A variant to Turing's theory of computing machines. Journal of the Association for Computing Machinery (ACM) 4(1), 63–92 (1957)
40. Woods, D., Neary, T.: On the time complexity of 2-tag systems and small universal Turing machines. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 439–448. IEEE, Berkeley (2006)
41. Zykin, G.P.: Remarks about a theorem of Hao Wang. Algebra i Logika 2, 33–35 (1963) (in Russian)

Chapter 6
Constructing Reversible Turing Machines by Reversible Logic Element with Memory

Kenichi Morita

Abstract. A reversible logic element is a primitive for composing reversible computing machines. There are two kinds of such elements, i.e., one without memory, which is commonly called a reversible logic gate, and one with memory. It is known that, in reversible computing, a reversible logic element with memory is useful as well as a reversible logic gate, since reversible computing machines can be constructed simply using such a type of elements. A rotary element (RE) is a typical instance of a reversible logic element with 1-bit memory, whose operations can be easily understood by an intuitive interpretation. In this survey, we discuss how RE is implemented in an idealized reversible physical system, and how reversible Turing machines (RTMs) can be constructed from REs. In particular, we give a new simpler construction method of RTMs than the previous one.

6.1 Introduction

Reversible computing is a paradigm of computation that is closely related to reversibility in physics. A reversible logic element is a primitive for composing reversible logic circuits, by which reversible computing machines can be implemented. The function of a reversible logic element is described by a one-to-one mapping. Thus, the behavior of a reversible logic circuit can be traced backward uniquely with respect to the time axis.

There are two types of reversible logic elements: one without memory, which is commonly called a reversible logic gate [5, 19, 20], and one with memory [10, 16]. The conventional design theory of logic circuits has been developed using logic gates as primitives (but in the study of asynchronous circuits, logic elements with memory are sometimes used [4, 6]). On the other hand, in the case of reversible computing, logic elements with memory are also useful. The main reason is that

Kenichi Morita
Hiroshima University, Higashi-Hiroshima 739-8527, Japan
e-mail: km@hiroshima-u.ac.jp

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_6

if we use an appropriate reversible logic element with memory, we can construct various reversible computing models easily. In [10] it is shown that any reversible Turing machine can be built using a rotary element (RE), a typical instance of a reversible logic element with memory (RLEM). In [8], another design method of reversible Turing machines using different kinds of RLEMs is given.

In this survey, we describe how RLEMs are defined, how they are related to physical reversibility, and how reversible Turing machines (RTMs) can be built by them. In particular, here we give a simpler construction method of RTMs by REs than the one given in [10].

6.2 Reversible Logic Element with Memory (RLEM)

We first give a definition of a sequential machine of Mealy type (i.e., a finite automaton with an output port as well as an input port), since a reversible logic element with memory is defined as a kind of a reversible sequential machine.

Definition 1. A *sequential machine* (SM) is a system defined by a quadruple $M = (Q, \Sigma, \Gamma, \delta)$. Here, Q is a finite set of internal states, Σ and Γ are finite sets of input and output symbols, and $\delta : Q \times \Sigma \to Q \times \Gamma$ is a move function. If δ is injective, M is called a *reversible sequential machine* (RSM). Note that if M is reversible, $|\Sigma| \leq |\Gamma|$ must hold.

Definition 2. A *reversible logic element with memory* (RLEM) is an RSM $M = (Q, \Sigma, \Gamma, \delta)$ such that $|\Sigma| = |\Gamma|$. It is also called a $|Q|$-state $|\Gamma|$-symbol RLEM.

A *rotary element* (RE) is an instance of 2-state 4-symbol RLEM that was first introduced in [10]. It is defined by $M_{\mathrm{RE}} = (\{\mathrm{H}, \mathrm{V}\}, \{n, e, s, w\}, \{n', e', s', w'\}, \delta_{\mathrm{RE}})$, where δ_{RE} is given in Table 6.1. For example, if its present state is H and the input is n, then the next state is V and the output is w', and hence $\delta_{\mathrm{RE}}(\mathrm{H}, n) = (\mathrm{V}, w')$.

Table 6.1 A tabular description of the move function δ_{RE} of a rotary element (RE)

	Input			
Present state	n	e	s	w
H	V w'	H w'	V e'	H e'
V	V s'	H n'	V n'	H s'

RE has the following intuitive interpretation. It is shown by a box that contains a rotatable bar (Figure 6.1). The direction of the bar can be either horizontal or vertical, and they represent the states H and V, respectively. To each input (output, respectively) symbol, there corresponds an input (output) line. A signal (or a particle) goes into or goes out from the box through these lines. The bar controls the move direction of an input signal. When no particle is coming, nothing happens on the RE. If a particle comes from the direction parallel to the bar, then it goes out from the output line of the opposite side without affecting the direction of the bar (Figure 6.2 (a)). If a particle comes from the direction orthogonal to the bar, then it makes a right turn, and rotates the bar by 90 degrees counterclockwise (Figure 6.2 (b)).

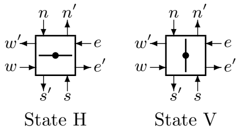


Fig. 6.1 Pictorial representation of two states of a rotary element (RE)

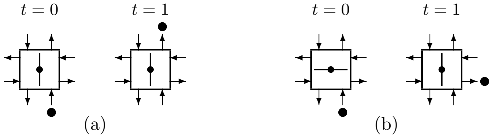


Fig. 6.2 An intuitive interpretation on the operations of RE: (a) the parallel case, and (b) the orthogonal case

It is shown that any RSM can be built using only REs [11], and that any reversible Turing machine is realized as an infinite circuit composed only of REs [10]. Hence, in such a sense RE is universal.

There are infinitely many RLEMs if the numbers of states and symbols are not bounded. There are $(2k)!$ kinds of 2-state k-symbol RLEMs. Hence, the total numbers of 2-state 2-, 3-, and 4-symbol RLEMs are 24, 720, and 40,320, respectively. However, if we regard two RLEMs to be "equivalent" when one is obtained from another by renaming input/output symbols and/or states, then the number of equivalence classes decreases considerably. In [16] 2-state RLEMs are classified, and the numbers of such equivalence classes of 2-state 2-, 3-, and 4-symbol RLEMs are 8, 24, and 82, respectively. There is also a "degenerate" 2-state RLEM that is further equivalent to a 1-state RLEM (hence it acts as mere connecting wires for a signal) or a 2-state RLEM having fewer symbols. The numbers of non-degenerate 2-state 2-, 3-, and 4-symbol RLEMs are 4, 14, and 55, respectively [16].

As discussed in [9] by Margenstern, it is important to know the frontier between universality and non-universality. In [15] it is proved that for every non-degenerate 2-state k-symbol RLEM, there is a circuit composed only of it that simulates RE if $k > 2$. Hence, every non-degenerate 2-state k-symbol RLEM is universal if $k > 2$. On the other hand, for four non-degenerate 2-state 2-symbol RLEMs, three of them has been proved to be non-universal [18], but it is left open for the remaining one.

6.3 Relation to Physical Reversibility

We now argue how a reversible logic element is related to physical reversibility. Here, we use the *billiard ball model* (BBM) proposed by Fredkin and Toffoli [5] as a reversible physical model. It is an idealized model of Newtonian mechanics consisting of balls and reflectors. Computation can be carried out by balls that collide with other balls or reflectors, and hence it is a kind of collision-based computing (see [1]). It is assumed that collisions are elastic, and there is no friction. Figure 6.3 shows a realization of an *interaction gate*, which is a 2-input 4-output reversible logic gate, in BBM [5].

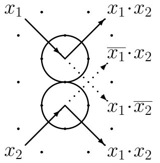


Fig. 6.3 An interaction gate realized in the billiard ball model (BBM) [5]

It is known that we can construct RE by reversible logic gates and delay elements [14]. Hence, it is in principle possible to realize RE in BBM. But, it is not a good method. This is because we need many reversible logic gates for constructing one RE. Moreover, when we want to implement a logic gate in BBM, exact synchronization of two or more moving balls is necessary. Fortunately, there is a simple way of directly realizing RE in BBM shown in Figure 6.4 [12]. The RE implemented in BBM consists of one stationary ball called a *state ball*, and many reflectors indicated by small rectangles. A state ball is placed at the position of H or V in Figure 6.4(a) depending on the state of the simulated RE. A moving ball called a *signal ball* can be given to any one of the input lines n, e, s, and w. Assume a state ball (a light-colored one) is at the position V, and a signal ball (a dark one) is given to the input line s as in Figure 6.4(b). Then, the signal ball moves straight ahead without interacting the state ball or reflectors, and finally goes out from the output line n'. This simulates the case of Figure 6.2(a). Next, consider the case that a state ball is at the position H, and a signal ball comes from the input line s (Figure 6.4(c)). Firstly, the two balls collide at the position 1. After that, they goes along the paths indicated by arrows. This is performed by appropriately placing reflectors. Then, the two balls collide again at the position 6, and the state ball stops there. The signal ball finally goes out from the output line e' (Figure 6.4(c)). This simulates the case of Figure 6.2(b). Other cases are also simulated correctly.

In this realization, the signal ball can be given to an input line *at any moment* and *at any speed*, since the state ball is stationary till they collide. Hence, there is no need to synchronize the input timing at all. Only the time interval between the first and the second collisions in Figure 6.4(c) should be adjusted exactly. In addition, when we consider a physical realization, the state ball need not be a movable object, but a suitable physical state that acts reversibly (such as a quantum state). Thus, the

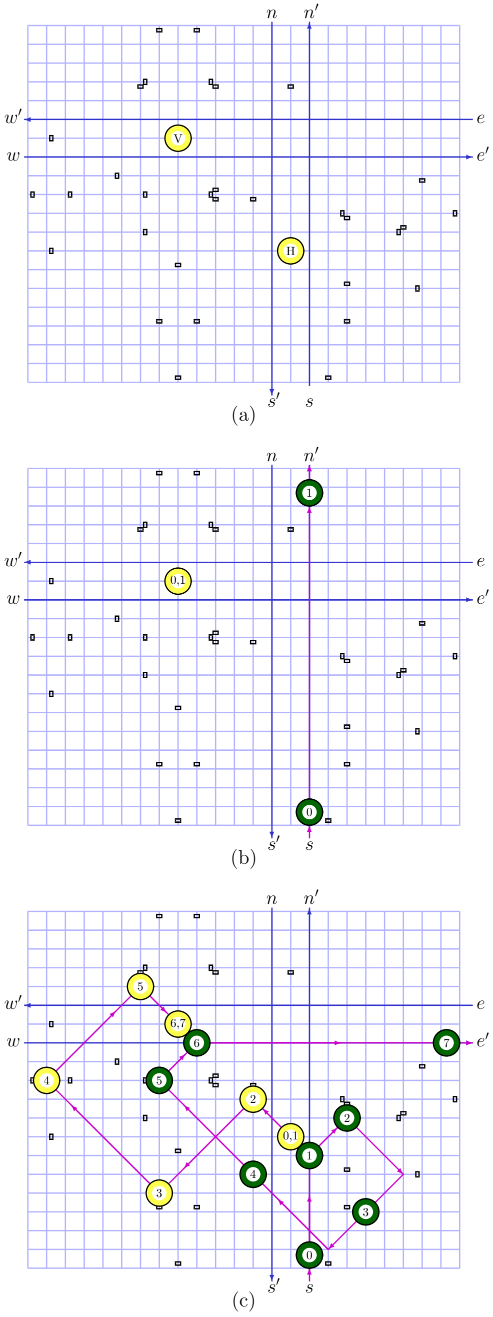


Fig. 6.4 RE realized in BBM. (a) H and V are the places where a state ball is put. Small rectangles indicate reflectors for balls. (b) The case that the state is V, and the input is *s* (corresponding to Figure 6.2(a)). (c) The case that the state is H, and the input is *s* (corresponding to Figure 6.2(b)).

above idea broaden the possibility of practical realization of RE in a real system having physical reversibility. It is noted that Mukai and Morita [17] showed any m-state k-symbol RLEM can be realized in BBM by a systematic method if $k \leq 4$.

6.4 Constructing Reversible Turing Machines by Rotary Element

In this section we show any reversible Turing machine (RTM) can be realized as a circuit composed only of REs. An RTM was first investigated by Lecerf [7] who showed unsolvability of their halting problem. Bennett [2,3] then studied them from the standpoint of thermodynamics of computing, and showed its universality. In [2] RTMs were defined in the quadruple form, since an "inverse" RTM is easily obtained from a given RTM if it is given in the quadruple form. But, here we use an RTM in the quintuple form, since this form is convenient to make a circuit that simulates the RTM simpler.

Definition 3. A *Turing machine* (TM) is defined by $T = (Q, S, q_0, F, s_0, \delta)$, where Q is a set of states of the finite control, S is a set of tape symbols, $q_0 \in Q$ is an initial state, $F \subset Q$ is a set of final states, and $s_0 \in S$ is a blank symbol. δ is a move relation, which is a subset of $(Q \times S \times S \times \{L, N, R\} \times Q)$, where L, N, and R stand for left-shift, no-shift, and right-shift of the head. Each element of δ is a quintuple of the form $[p, s, s', d, q]$. It means if T reads the symbol s in the state p, then write s', shift the head to the direction d, and go to the state q.

T is called *deterministic* iff the following condition holds for any pair of distinct quintuples $[p_1, s_1, s'_1, d_1, q_1]$ and $[p_2, s_2, s'_2, d_2, q_2]$ in δ: if $p_1 = p_2$, then $s_1 \neq s_2$.

T is called *reversible* iff the following condition holds for any pair of distinct quintuples $[p_1, s_1, s'_1, d_1, q_1]$ and $[p_2, s_2, s'_2, d_2, q_2]$ in δ: if $q_1 = q_2$, then $s'_1 \neq s'_2 \wedge d_1 = d_2$. Hereafter, by RTM we mean a deterministic and reversible TM.

Example 1. Let T_{parity} be an RTM defined as follows. $T_{\text{parity}} = (Q, \{0, 1\}, q_0, \{q_{\text{acc}}\}, 0, \delta)$, where $Q = \{q_0, q_1, q_2, q_{\text{acc}}, q_{\text{rej}}\}$, and $\delta = \{[\, q_0, 0, 1, R, q_1 \,], [\, q_1, 0, 1, L, q_{\text{acc}} \,], [\, q_1, 1, 0, R, q_2 \,], [\, q_2, 0, 1, L, q_{\text{rej}} \,], [\, q_2, 1, 0, R, q_1 \,]\}$. T_{parity} checks if a given unary number n is even or odd. If it is even, T_{parity} halts in the accepting state q_{acc}, otherwise halts in the rejecting state q_{rej}. All the symbols read by T_{parity} are complemented (see Figure 6.5). It is easy to see that T_{parity} satisfies the condition for reversibility.

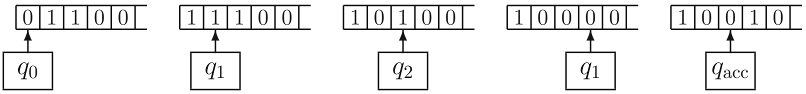


Fig. 6.5 An example of a computing process of the RTM T_{parity} for the unary input 11

In [10] it is shown that any RTM can be realized by a circuit composed only of REs. There, two kinds of modules called a memory cell (i.e., one square of a tape), and a finite-state control of an RTM are formalized as RSMs. Then, the two modules are constructed by REs. Here, we show a simpler formalization and construction method of these modules than the one given in [10]. We assume the given RTM has only one tape that is semi-infinite to the right, and uses two kinds of tape symbols 0 and 1. An RTM having multiple tapes and/or many symbols can also be constructed in a similar manner as below.

A *memory cell* is an RSM that acts as one square of a tape of an RTM. Connecting an infinite number of memory cells rightward, we can obtain a *tape unit* for the RTM. The memory cell keeps a tape symbol $s \in \{0,1\}$, and the information h whether the head of the RTM is on this cell ($h = 1$) or not ($h = 0$). Hence, its state set is $\{(h,s) \,|\, h,s \in \{0,1\}\}$. It has ten kinds of input symbols listed in Table 6.2, which are interpreted as instruction signals to the tape unit or response signals to the finite-state control. For each input symbol, there is a corresponding output symbol, which is indicated by a one with $'$. Thus, it has also ten kinds of output symbols.

Table 6.2 Ten kinds of input symbols of a memory cell, and their meanings

Symbol	Instruction/Response	Meaning
W0	Write 0	Instruction of writing the symbol 0 at the head position. By this instruction, read operation is also performed.
W1	Write 1	Instruction of writing the symbol 1 at the head position. By this instruction, read operation is also performed.
R0	Read 0	Response signal telling the read symbol is 0.
R1	Read 1	Response signal telling the read symbol is 1.
SL	Shift-left	Instruction of shift-left operation.
SLI	Shift-left immediate	Instruction of placing the head on this cell by shifting-left.
SLc	Shift-left completed	Response signal of shift-left operation.
SR	Shift-right	Instruction of shift-right operation.
SRI	Shift-right immediate	Instruction of placing the head on this cell by shifting-right.
SRc	Shift-right completed	Response signal of shift-right operation.

The input symbol W0 or W1 is an instruction to the tape unit for writing the tape symbol 0 or 1, respectively, in the memory cell at the head position. After the writing operation, the output symbol R0$'$ or R1$'$ is generated as a response signal depending on the old tape symbol at the head position, and is sent to the finite-state control. Hence, the write instruction performs not only the write operation but also the read operation. Otherwise, reversibility of the memory cell does not hold. Assume a symbol W0 or W1 is given to a memory cell. If the tape head is not at the cell (i.e., $h = 0$), then it generates an output symbol W0$'$ or W1$'$, respectively, and send it to the right-neighboring memory cell. If the tape head is at the cell (i.e., $h = 1$), then the cell sets its new tape symbol s to 0 (if W0 is given) or 1 (if W1 is given). It also generates an output symbol R0$'$ or R1$'$ depending on the old tape symbol, and send it to the left-neighboring memory cell.

Note that, if an RTM needs to read a tape symbol, then it is performed by the instruction W0. By this, the finite-state control obtains a response signal R0 or R1, and the tape symbol at the head position is cleared to 0. After that, if the RTM gives the instruction W0 or W1, then the tape symbol 0 or 1 is written at the head position, and finally the finite-state control obtains the response signal R0.

In the previous construction of the memory cell in [10], read and write instructions are separated (where an instruction of "write complementary symbol" was used for keeping reversibility). Due to this, "inverse-read instructions" that undo the read operation were also necessary, because "branching" by read instruction should be reversibly "merged" afterwards. By this, eight kinds of instruction and response symbols for read and write operations were used. But here, by making good use of symmetry between reading and writing, only four kinds of symbols W0, W1, R0, and R1 are used, and thus the circuit for the memory cell is simplified considerably.

The input symbol SL is an instruction for shifting the tape head to the left. Assume a symbol SL is given to a memory cell. If the tape head is not at the cell (i.e., $h = 0$), then it simply sends an output symbol SL$'$ to the right-neighboring memory cell. If the tape head is at the cell (i.e., $h = 1$), then the cell sets the new h to 0, and sends an output symbol SLI$'$ to the left-neighboring memory cell. If the latter memory cell receive an input symbol SLI, then it sets the new h to 1, and sends an output symbol SLc$'$ to the left. By such a process, shift-left operation is performed correctly. The input symbols SR, SRI, and SRc are similar to the symbols SL, SLI, and SLc, except that an output symbol SRI$'$ is sent to the right-neighboring cell.

By above, the memory cell is formalized as the following RSM M_{C}.

$$M_{\mathrm{C}} = (Q_{\mathrm{C}}, \Sigma_{\mathrm{C}}, \Gamma_{\mathrm{C}}, \delta_{\mathrm{C}})$$

$$\begin{array}{lll}
Q_{\mathrm{C}} = \{(h,s)\,|\,h,s \in \{0,1\}\} & & \\
\Sigma_{\mathrm{C}} = \{\text{ W0, W1, R0, R1, SL, SLI, SLc, SR, SRI, SRc }\} & & \\
\Gamma_{\mathrm{C}} = \{x'\,|\,x \in \Sigma_{\mathrm{C}}\} & & \\
\delta_{\mathrm{C}}((0,s),y) & = ((0,s),y') & (y \in \Sigma_{\mathrm{C}} - \{\text{SLI, SRI}\}) \\
\delta_{\mathrm{C}}((0,s),\text{SLI}) & = ((1,s),\text{SLc}') & \\
\delta_{\mathrm{C}}((0,s),\text{SRI}) & = ((1,s),\text{SRc}') & \\
\delta_{\mathrm{C}}((1,0),\text{W0}) & = ((1,0),\text{R0}') & \\
\delta_{\mathrm{C}}((1,1),\text{W0}) & = ((1,0),\text{R1}') & \\
\delta_{\mathrm{C}}((1,0),\text{W1}) & = ((1,1),\text{R0}') & \\
\delta_{\mathrm{C}}((1,1),\text{W1}) & = ((1,1),\text{R1}') & \\
\delta_{\mathrm{C}}((1,s),\text{SL}) & = ((0,s),\text{SLI}') & \\
\delta_{\mathrm{C}}((1,s),\text{SR}) & = ((0,s),\text{SRI}') &
\end{array}$$

Figure 6.6 shows a circuit composed only of REs that realizes the RSM M_{C}. Each of the rotary elements at the positions h and s is set to state V if the value is 0, and state H if it is 1. Connecting an infinite number of copies of this circuit to the right, we have a tape unit for a 2-symbol RTM. At the left end of the array of memory cells, a circuit of a finite-state control explained below will be attached.

A finite-state control of an RTM can be also realized by rotary elements easily, since it is a kind of an RSM. In this circuit, each quintuple of an RTM is executed

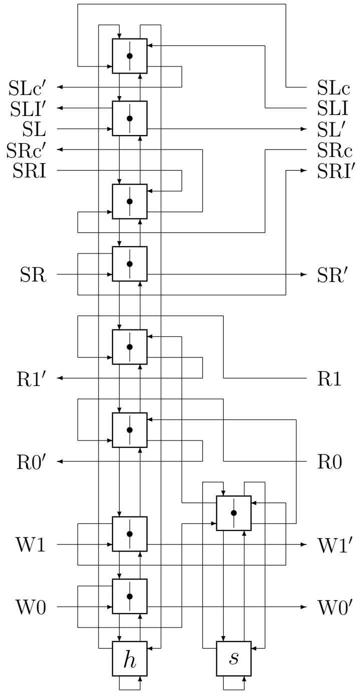


Fig. 6.6 A memory cell implemented by REs

by producing read, write, and shift instructions consecutively, and sending them to an array of memory cells. Firstly, giving the W0 instruction, a read operation is performed. As a result, a completion signal R0 or R1 is obtained. Then, giving W0 or W1 instruction, a write operation is performed, and a completion signal R0 is obtained. Finally, giving SL or SR instruction, shift operation is performed. A circuit of a finite-state control for T_{parity} in Example 1 is shown in Figure 6.7.

In the circuit of a finite-state control, we lay REs in 4 rows. In Figure 6.7, three elements in the 4th row correspond to q_0, q_1 and q_2, respectively. They send W0 instruction to read the symbol at the head position. If they receive a signal R0 or R1, they determine which quintuple of T_{parity} must be executed. Giving an instruction W0 or W1 according to the chosen quintuple, the 4 elements of the 3rd row perform writing and state-change. Then the elements of the 1st or 2nd rows perform a head shift.

Figure 6.8 is a circuit that simulates the RTM T_{parity} in Example 1. If we give a signal (or a particle) to the input port “Begin,” then it starts to compute. Finally, the particle comes out from the output port “Accept” or “Reject” depending on the parity of the given input on the tape.

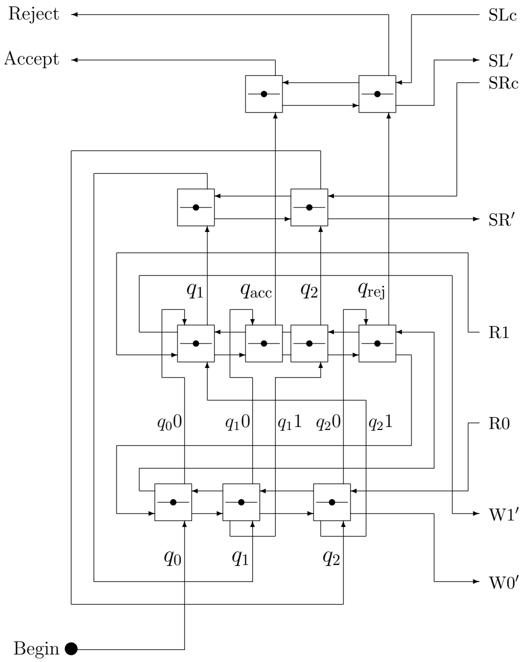


Fig. 6.7 A circuit that simulates the finite-state control of T_{parity} in Example 1

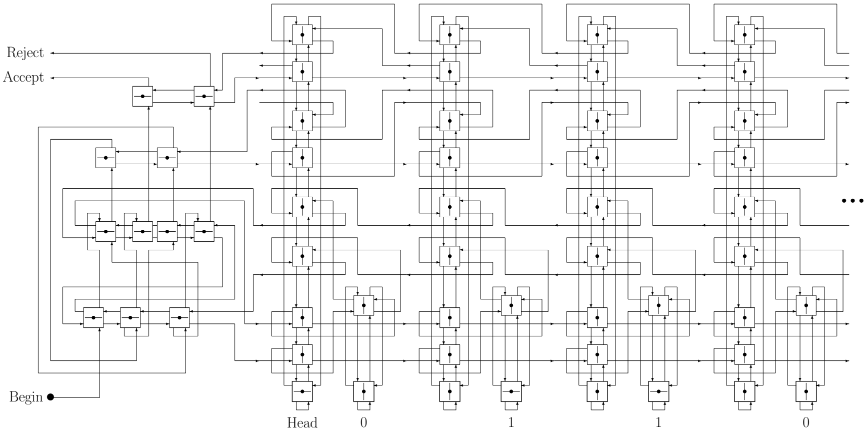


Fig. 6.8 A circuit made of REs that simulates the RTM T_{parity} in Example 1. An example of its whole computing process is shown in 4406 figures in [13].

6.5 Concluding Remarks

In this survey/tutorial paper, we discussed how a reversible logic element with memory (RLEM) is defined, how it is related to physical reversibility, and how it is used to construct reversible computing systems. In particular, reversible Turing machines (RTMs) can be constructed rather simply as shown in Figure 6.8 using rotary element (RE), a typical RLEM. It should be noted, if we further implement each RE by a mechanism in the billiard ball model (BBM) as in Figure 6.4, then the whole system of the RTM can be realized in the space of BBM. Here, we used RE to construct RTMs. However, since it is known "all" non-degenerate 2-state RLEMs except only four are universal [15], we can also use any one of them for composing RTMs.

Acknowledgements. This work was supported by JSPS KAKENHI Grant Number 24500017.

References

1. Adamatzky, A.: Collision-Based Computing. Springer (2002), doi:10.1007/978-1-4471-0129-1
2. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. 17, 525–532 (1973), doi:10.1147/rd.176.0525
3. Bennett, C.H.: The thermodynamics of computation—a review. Int. J. Theoret. Phys. 21, 905–940 (1982), doi:10.1007/BF02084158
4. Büning, H., Priese, L.: Universal asynchronous iterative arrays of Mealy automata. Acta Informatica 13, 269–285 (1980), doi:10.1007/BF00288646
5. Fredkin, E., Toffoli, T.: Conservative logic. Int. J. Theoret. Phys. 21, 219–253 (1982), doi:10.1007/BF01857727
6. Keller, R.: Towards a theory of universal speed-independent modules. IEEE Trans. Computers C-23, 21–33 (1974), doi:10.1109/T-C.1974.223773
7. Lecerf, Y.: Machines de Turing réversibles — récursive insolubilité en $n \in \mathbf{N}$ de l'équation $u = \theta^n u$, où θ est un isomorphisme de codes. Comptes Rendus Hebdomadaires des Séances de L'académie des Sciences 257, 2597–2600 (1963)
8. Lee, J., Yang, R., Morita, K.: Design of 1-tape 2-symbol reversible Turing machines based on reversible logic elements. Theoret. Comput. Sci. 460, 78–88 (2012), doi:10.1016/j.tcs.2012.07.027
9. Margenstern, M.: Frontier between decidability and undecidability: a survey. Theoret. Comput. Sci. 231, 217–251 (2000), doi:10.1016/S0304-3975(99)00102-4
10. Morita, K.: A simple universal logic element and cellular automata for reversible computing. In: Margenstern, M., Rogozhin, Y. (eds.) MCU 2001. LNCS, vol. 2055, pp. 102–113. Springer, Heidelberg (2001)
11. Morita, K.: A new universal logic element for reversible computing. In: Martin-Vide, C., Mitrana, V. (eds.) Grammars and Automata for String Processing, pp. 285–294. Taylor & Francis, London (2003), doi:10.1201/9780203009642.ch28
12. Morita, K.: Reversible computing and cellular automata — A survey. Theoret. Comput. Sci. 395, 101–131 (2008), doi:10.1016/j.tcs.2008.01.041
13. Morita, K.: Constructing a reversible Turing machine by a rotary element, a reversible logic element with memory. Hiroshima University Institutional Repository (2010), `http://ir.lib.hiroshima-u.ac.jp/00029224`

14. Morita, K.: Reversible Computing. Kindai Kagaku-sha Co., Ltd., Tokyo (2012) ISBN 978-4-7649-0422-4 (in Japanese)
15. Morita, K., Ogiro, T., Alhazov, A., Tanizawa, T.: Non-degenerate 2-state reversible logic elements with three or more symbols are all universal. J. Multiple-Valued Logic and Soft Computing 18, 37–54 (2012)
16. Morita, K., Ogiro, T., Tanaka, K., Kato, H.: Classification and universality of reversible logic elements with one-bit memory. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 245–256. Springer, Heidelberg (2005)
17. Mukai, Y., Morita, K.: Realizing reversible logic elements with memory in the billiard ball model. Int. J. of Unconventional Computing 8(1), 47–59 (2012)
18. Mukai, Y., Ogiro, T., Morita, K.: Universality problems on reversible logic elements with 1-bit memory. Int. J. Unconventional Computing (to appear)
19. Toffoli, T.: Reversible computing. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 632–644. Springer, Heidelberg (1980)
20. Toffoli, T.: Bicontinuous extensions of invertible combinatorial functions. Math. Syst. Theory 14, 12–23 (1981), doi:10.1007/BF01752388

Chapter 7
The Grossone Methodology Perspective on Turing Machines

Yaroslav D. Sergeyev and Alfredo Garro

Abstract. This chapter discusses how the mathematical language used to describe and to observe automatic computations influences the accuracy of the obtained results. The chapter presents results obtained by describing and observing different kinds of Turing machines (single and multi-tape, deterministic and non-deterministic) through the lens of a new mathematical language named Grossone. This emerging language is strongly based on three methodological ideas borrowed from Physics and applied to Mathematics: the distinction between the object (indeed mathematical object) of an observation and the instrument used for this observation; interrelations holding between the object and the tool used for the observation; the accuracy of the observation determined by the tool. In the chapter, the new results are compared to those achievable by using traditional languages. It is shown that both languages do not contradict each other but observe and describe the same object (Turing machines) but with different accuracies.

7.1 Introduction

Turing machines represent one of the simple abstract computational devices that can be used to investigate the limits of computability . In this chapter, they are considered from several points of view that emphasize the importance and the relativity of

Yaroslav D. Sergeyev
Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica (DIMES), Università della Calabria, Rende (CS), Italy
N.I. Lobatchevsky State University, Nizhni Novgorod, Russia
Istituto di Calcolo e Reti ad Alte Prestazioni, C.N.R., Rende (CS), Italy
e-mail: yaro@si.dimes.unical.it

Alfredo Garro
Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica (DIMES), Università della Calabria, Rende (CS), Italy
e-mail: garro@dimes.unical.it

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_7

mathematical languages used to describe the Turing machines. A deep investigation is performed on the interrelations between mechanical computations and their mathematical descriptions emerging when a human (the researcher) starts to describe a Turing machine (the object of the study) by different mathematical languages (the instruments of investigation).

In particular, we focus our attention on different kinds of Turing machines (single and multi-tape, deterministic and non-deterministic) by organizing and discussing the results presented in [42] and [43] so to provide a compendium of our multi-year research on this subject.

The starting point is represented by numeral systems [1] that we use to write down numbers, functions, models, etc. and that are among our tools of investigation of mathematical and physical objects. It is shown that numeral systems strongly influence our capabilities to describe both the mathematical and physical worlds. A new numeral system introduced in [30,32,37]) for performing computations with infinite and infinitesimal quantities is used for the observation of mathematical objects and studying Turing machines. The new methodology is based on the principle 'The part is less than the whole' introduced by Ancient Greeks (see, e.g., Euclid's Common Notion 5) and observed in practice. It is applied to all sets and processes (finite and infinite) and all numbers (finite, infinite, and infinitesimal).

In order to see the place of the new approach in the historical panorama of ideas dealing with infinite and infinitesimal, see [19–21,35,36,42,43]. The new methodology has been successfully applied for studying a number of applications: percolation (see [13,45]), Euclidean and hyperbolic geometry (see [22,29]), fractals (see [31,33,40,45]), numerical differentiation and optimization (see [7,34,38,48]), ordinary differential equations (see [41]), infinite series (see [35,39,47]), the first Hilbert problem (see [36]), and cellular automata (see [8]).

The rest of the chapter is structured as follows. In Section 7.2, Single and Multi-tape Turing machines are introduced along with "classical" results concerning their computational power and related equivalences; in Section 7.3 a brief introduction to the new language and methodology is given whereas their exploitation for analyzing and observing the different types of Turing machines is discussed in Section 7.4. It shows that the new approach allows us to observe Turing machines with a higher accuracy giving so the possibility to better characterize and distinguish machines which are equivalent when observed within the classical framework. Finally, Section 7.5 concludes the chapter.

[1] We are reminded that a *numeral* is a symbol or group of symbols that represents a *number*. The difference between numerals and numbers is the same as the difference between words and the things they refer to. A *number* is a concept that a *numeral* expresses. The same number can be represented by different numerals. For example, the symbols '7', 'seven', and 'VII' are different numerals, but they all represent the same number.

7.2 Turing Machines

The Turing machine is one of the simple abstract computational devices that can be used to model computational processes and investigate the limits of computability. In the following subsections, deterministic Single and Multi-tape Turing machines are described along with important classical results concerning their computational power and related equivalences (see Section 7.2.1 and 7.2.2 respectively); finally, non-deterministic Turing machines are introduced (see Section 7.2.3).

7.2.1 Single-Tape Turing Machines

A Turing machine (see, e.g., [12, 44]) can be defined as a 7-tuple

$$\mathcal{M} = \left\langle Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta \right\rangle, \tag{7.1}$$

where Q is a finite and not empty set of states; Γ is a finite set of symbols; $\bar{b} \in \Gamma$ is a symbol called blank; $\Sigma \subseteq \{\Gamma - \bar{b}\}$ is the set of input/output symbols; $q_0 \in Q$ is the initial state; $F \subseteq Q$ is the set of final states; $\delta : \{Q - F\} \times \Gamma \mapsto Q \times \Gamma \times \{R, L, N\}$ is a partial function called the transition function, where L means left, R means right, and N means no move .

Specifically, the machine is supplied with: (i) a *tape* running through it which is divided into cells each capable of containing a symbol $\gamma \in \Gamma$, where Γ is called the tape alphabet, and $\bar{b} \in \Gamma$ is the only symbol allowed to occur on the tape infinitely often; (ii) a *head* that can read and write symbols on the tape and move the tape left and right one and only one cell at a time. The behavior of the machine is specified by its *transition function* δ and consists of a sequence of computational steps ; in each step the machine reads the symbol under the head and applies the *transition function* that, given the current state of the machine and the symbol it is reading on the tape, specifies (if it is defined for these inputs): (i) the symbol $\gamma \in \Gamma$ to write on the cell of the tape under the head; (ii) the move of the tape (L for one cell left, R for one cell right, N for no move); (iii) the next state $q \in Q$ of the machine.

7.2.1.1 Classical Results for Single-Tape Turing Machines

Starting from the definition of Turing machine introduced above, classical results (see, e.g., [1]) aim at showing that different machines in terms of provided tape and alphabet have the same computational power, i.e., they are able to execute the same computations. In particular, two main results are reported below in an informal way.

Given a Turing machine $\mathcal{M} = \{Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta\}$, which is supplied with an infinite tape, it is always possible to define a Turing machine $\mathcal{M}' = \{Q', \Gamma', \bar{b}, \Sigma', q_0', F', \delta'\}$ which is supplied with a semi-infinite tape (e.g., a tape with a left boundary) and is equivalent to $\mathcal{M}$, i.e., is able to execute all the computations of $\mathcal{M}$.

Given a Turing machine $\mathcal{M} = \{Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta\}$, it is always possible to define a Turing machine $\mathcal{M}' = \{Q', \Gamma', \bar{b}, \Sigma', q_0', F', \delta'\}$ with $|\Sigma'| = 1$ and $\Gamma' = \Sigma' \cup \{\bar{b}\}$, which is equivalent to $\mathcal{M}$, i.e., is able to execute all the computations of $\mathcal{M}$.

It should be mentioned that these results, together with the usual conclusion regarding the equivalences of Turing machines, can be interpreted in the following, less obvious, way: they show that when we observe Turing machines by exploiting the classical framework we are not able to distinguish, from the computational point of view, Turing machines which are provided with alphabets having different number of symbols and/or different kind of tapes (infinite or semi-infinite) (see [42] for a detailed discussion).

7.2.2 Multi-tape Turing Machines

Let us consider a variant of the Turing machine defined in (7.1) where a machine is equipped with multiple tapes that can be simultaneously accessed and updated through multiple heads (one per tape). These machines can be used for a more direct and intuitive resolution of different kind of computational problems. As an example, in checking if a string is palindrome it can be useful to have two tapes on which represent the input string so that the verification can be efficiently performed by reading a tape from left to right and the other one from right to left.

Moving towards a more formal definition, a k-tapes, $k \geq 2$, Turing machine (see [12]) can be defined (cf. (7.1)) as a 7-tuple

$$\mathcal{M}_K = \left\langle Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta^{(k)} \right\rangle, \tag{7.2}$$

where $\Sigma = \bigcup_{i=1}^{k} \Sigma_i$ is given by the union of the symbols in the k input/output alphabets $\Sigma_1, \ldots, \Sigma_k$; $\Gamma = \Sigma \cup \{\bar{b}\}$ where $\bar{b}$ is a symbol called blank; Q is a finite and not empty set of states; $q_0 \in Q$ is the initial state; $F \subseteq Q$ is the set of final states; $\delta^{(k)} : \{Q - F\} \times \Gamma_1 \times \cdots \times \Gamma_k \mapsto Q \times \Gamma_1 \times \cdots \times \Gamma_k \times \{R, L, N\}^k$ is a partial function called the transition function, where $\Gamma_i = \Sigma_i \cup \{\bar{b}\}, i = 1, \ldots, k$, L means left, R means right, and N means no move .

This definition of $\delta^{(k)}$ means that the machine executes a transition starting from an internal state q_i and with the k heads (one for each tape) above the characters $a_{i1}, \ldots, a_{ik}$, i.e., if $\delta^{(k)}(q_1, a_{i1}, \ldots, a_{ik}) = (q_j, a_{j_1}, \ldots, a_{j_k}, z_{j_1}, \ldots, z_{j_k})$ the machine goes in the new state q_j, write on the k tapes the characters $a_{j_1}, \ldots, a_{j_k}$ respectively, and moves each of its k heads left, right or no move, as specified by the $z_{j_l} \in \{R, L, N\}, l = 1, \ldots, k$.

A machine can adopt for each tape a different alphabet, in any case, for each tape, as for the Single-tape Turing machines, the minimum portion containing characters distinct from $\bar{b}$ is usually represented. In general, a typical configuration of a Multi-tape machine consists of a read-only input tape, several read and write work tapes, and a write-only output tape, with the input and output tapes accessible only in one direction. In the case of a k-tapes machine, the instant configuration of the machine, as for the Single-tape case, must describe the internal state, the contents of the tapes and the positions of the heads of the machine.

More formally, for a k-tapes Turing machine $\mathcal{M}_K = \left\langle Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta^{(k)} \right\rangle$ with $\Sigma = \bigcup_{i=1}^{k} \Sigma_i$ (see 7.2) a configuration of the machine is given by:

$$q\#\alpha_1 \uparrow \beta_1\#\alpha_2 \uparrow \beta_2\#\ldots\#\alpha_k \uparrow \beta_k, \tag{7.3}$$

where $q \in Q$; $\alpha_i \in \Sigma_i\Gamma_i^* \cup \{\varepsilon\}$ and $\beta_i \in \Gamma_i^*\Sigma_i \cup \{\bar{b}\}$. A configuration is *final* if $q \in F$.

The *starting* configuration usually requires the input string x on a tape, e.g., the first tape so that $x \in \Sigma_1^*$, and only $\bar{b}$ symbols on all the other tapes. However, it can be useful to assume that, at the beginning of a computation, these tapes have a starting symbol $Z_0 \notin \Gamma = \bigcup_{i=1}^{k} \Gamma_i$. Therefore, in the initial configuration the head on the first tape will be on the first character of the input string x, whereas the heads on the other tapes will observe the symbol Z_0, more formally, by re-placing $\Gamma_i = \Sigma_i \cup \{\bar{b}, Z_0\}$ in all the previous definition, a configuration $q\#\alpha_1 \uparrow \beta_1\#\alpha_2 \uparrow \beta_2\#\ldots\#\alpha_k \uparrow \beta_k$ is an *initial configuration* if $\alpha_i = \varepsilon, i = 1,\ldots,k, \beta_1 \in \Sigma_1^*, \beta_i = Z_0, i = 2,\ldots,k$ and $q = q_0$.

The application of the transition function $\delta^{(k)}$ at a machine configuration (c.f. (7.3)) defines a *computational step* of a Multi-tape Turing machine . The set of computational steps which bring the machine from the initial configuration into a final configuration defines the *computation* executed by the machine. As an example, the computation of a Multi-tape Turing machine $\mathcal{M}_K$ which computes the function $f_{\mathcal{M}_K}(x)$ can be represented as follows:

$$q_0\# \uparrow x\# \uparrow Z_0\#\ldots\# \uparrow Z_0 \overset{\rightarrow}{\mathcal{M}_K} q\# \uparrow x\# \uparrow f_{\mathcal{M}_K}(x)\# \uparrow \bar{b}\#\ldots\# \uparrow \bar{b} \tag{7.4}$$

where $q \in F$ and $\overset{\rightarrow}{\mathcal{M}_K}$ indicates the transition among machine configurations.

7.2.2.1 Classical Results for Multi-tape Turing Machines

It is worth noting that, although the k-tapes Turing machine can be used for a more direct resolution of different kind of computational problems, in the classical framework it has the same computational power of the Single-tape Turing machine. More formally, given a Multi-tape Turing machine it is always possible to define a Single-tape Turing machine which is able to fully simulate its behavior and therefore to completely execute its computations. In particular, the Single-tape Turing machines adopted for the simulation use a particular kind of the tape which is divided into tracks (multi-track tape). In this way, if the tape has m tracks, the head is able to access (for reading and/or writing) all the m characters on the tracks during a single operation. If for the m tracks the alphabets $\Gamma_1, \ldots \Gamma_m$ are adopted respectively, the machine alphabet Γ is such that $|\Gamma| = |\Gamma_1 \times \cdots \times \Gamma_m|$ and can be defined by an injective function from the set $\Gamma_1 \times \cdots \times \Gamma_m$ to the set Γ; this function will associate the symbol $\bar{b}$ in Γ to the tuple $(\bar{b}, \bar{b}, \ldots, \bar{b})$ in $\Gamma_1 \times \cdots \times \Gamma_m$. In general, the elements of Γ which correspond to the elements in $\Gamma_1 \times \cdots \times \Gamma_m$ can be indicated by $[a_{i1}, a_{i2}, \ldots, a_{im}]$ where $a_{ij} \in \Gamma_j$.

By adopting this notation it is possible to demonstrate that given a k-tapes Turing machine $\mathcal{M}_K = \{Q,\Gamma,\bar{b},\Sigma,q_0,F,\delta^{(k)}\}$ it is always possible to define a Single-tape Turing machine which is able to simulate t computational steps of $\mathcal{M}_K =$ in $O(t^2)$ transitions by using an alphabet with $O((2|\Gamma|)^k)$ symbols (see [1]) .

The proof is based on the definition of a machine $\mathcal{M}' = \{Q',\Gamma',\bar{b},\Sigma',q_0',F',\delta'\}$ with a Single-tape divided into $2k$ tracks (see [1]); k tracks for storing the characters in the k tapes of $\mathcal{M}_K$ and k tracks for signing through the marker $\downarrow$ the positions of the k heads on the k tapes of $\mathcal{M}_k$. As an example, this kind of tape can represent the content of each tapes of $\mathcal{M}_k$ and the position of each machine heads in its even and odd tracks respectively. As discussed above, for obtaining a Single-tape machine able to represent these $2k$ tracks, it is sufficient to adopt an alphabet with the required cardinality and define an injective function which associates a 2k-ple characters of a cell of the multi-track tape to a symbols in this alphabet.

The transition function $\delta^{(k)}$ of the k-tapes machine is given by $\delta^{(k)}(q_1,a_{i1},\ldots,a_{ik}) = (q_j,a_{j_1},\ldots,a_{j_k},z_{j_1},\ldots,z_{j_k})$, with $z_{j_1},\ldots,z_{j_k} \in \{R,L,N\}$; as a consequence the corresponding transition function δ' of the Single-tape machine, for each transition specified by $\delta^{(k)}$ must individuate the current state and the position of the marker for each track and then write on the tracks the required symbols, move the markers and go in another internal state. For each computational step of $\mathcal{M}_K$, the machine $\mathcal{M}'$ must execute a sequence of steps for covering the portion of tapes between the two most distant markers. As in each computational step a marker can move at most of one cell and then two markers can move away each other at most of two cells, after t steps of $\mathcal{M}_K$ the markers can be at most $2t$ cells distant, thus if $\mathcal{M}_K$ executes t steps, $\mathcal{M}'$ executes at most: $2\sum_{i=1}^{t} i = t^2 + t = O(t^2)$ steps .

Moving to the cost of the simulation in terms of the number of required characters for the alphabet of the Single-tape machine, we recall that $|\Gamma_1| = |\Sigma_1| + 1$ and that $|\Gamma_i| = |\Sigma_i| + 2$ for $2 \leq i \leq k$. So by multiplying the cardinalities of these alphabets we obtain that: $|\Gamma'| = 2^k(|\Sigma_1|+1)\prod_{i=2}^{k}(|\Sigma_i|+2) = O((2max_{1\leq i\leq k}|\Gamma_i|)^k)$.

7.2.3 Non-deterministic Turing Machines

A non-deterministic Turing machine (see [12]) can be defined (cf. (7.1)) as a 7-tuple

$$\mathcal{M}_N = \langle Q,\Gamma,\bar{b},\Sigma,q_0,F,\delta_N \rangle, \tag{7.5}$$

where Q is a finite and not empty set of states; Γ is a finite set of symbols; $\bar{b} \in \Gamma$ is a symbol called blank; $\Sigma \subseteq \{\Gamma - \bar{b}\}$ is the set of input/output symbols; $q_0 \in Q$ is the initial state; $F \subseteq Q$ is the set of final states; $\delta_N : \{Q - F\} \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{R,L,N\})$ is a partial function called the transition function, where L means left, R means right, and N means no move .

As for a deterministic Turing machine (see (7.1)), the behavior of $\mathcal{M}_N$ is specified by its transition function δ_N and consists of a sequence of computational steps . In each step, given the current state of the machine and the symbol it is reading on the tape, the transition function δ_N returns (if it is defined for these inputs) a set of triplets each of which specifies: (i) a symbol $\gamma \in \Gamma$ to write on the cell of the tape under the head; (ii) the move of the tape (L for one cell left, R for one cell right, N for no move); (iii) the next state $q \in Q$ of the Machine. Thus, in each computational step, the machine can *non-deterministically* execute different computations, one for each triple returned by the transition function.

An important characteristic of a non-deterministic Turing machine (see, e.g., [1]) is its non-deterministic degree

$$d = \nu(\mathcal{M}_N) = \max_{q \in Q-F, \gamma \in \Gamma} |\delta_N(q, \gamma)|$$

defined as the maximal number of different configurations reachable in a single computational step starting from a given configuration. The behavior of the machine can be then represented as a tree whose branches are the computations that the machine can execute starting from the initial configuration represented by the node 0 and nodes of the tree at the levels 1, 2, etc. represent subsequent configurations of the machine.

Let us consider an example shown in Fig. 7.1 where a non-deterministic machine $\mathcal{M}_N$ having the non-deterministic degree $d = 3$ is presented. The depth of the computational tree is equal to k. In this example, it is supposed that the computational tree of $\mathcal{M}_N$ is complete (i.e., each node has exactly d children). Then, obviously, the computational tree of $\mathcal{M}_N$ has $d^k = 3^k$ leaf nodes.

7.2.3.1 Classical Results for Non-deterministic Turing Machines

An important result for the classic theory on Turing machines (see e.g., [1]) is that for any non-deterministic Turing machine $\mathcal{M}_N$ there exists an equivalent deterministic Turing machine $\mathcal{M}_D$. Moreover, if the depth of the computational tree generated by $\mathcal{M}_N$ is equal to k, then for simulating $\mathcal{M}_N$, the deterministic machine $\mathcal{M}_D$ will execute at most

$$K_{\mathcal{M}_D} = \sum_{j=0}^{k} j d^j = O(kd^k)$$

computational steps.

Intuitively, for simulating $\mathcal{M}_N$, the deterministic Turing machine $\mathcal{M}_D$ executes a breadth-first visit of the computational tree of $\mathcal{M}_N$. If we consider the example from Fig. 7.1 with $k = 3$, then the computational tree of $\mathcal{M}_N$ has $d^k = 27$ leaf nodes and $d^k = 27$ computational paths consisting of $k = 3$ branches (i.e., computational steps) . Then, the tree contains $d^{k-1} = 9$ computational paths consisting of $k-1 = 2$ branches and $d^{k-2} = 3$ computational paths consisting of $k-2 = 1$ branches . Thus, for simulating all the possible computations of $\mathcal{M}_N$, i.e., for complete visiting the computational tree of $\mathcal{M}_N$ and considering all the possible computational paths of

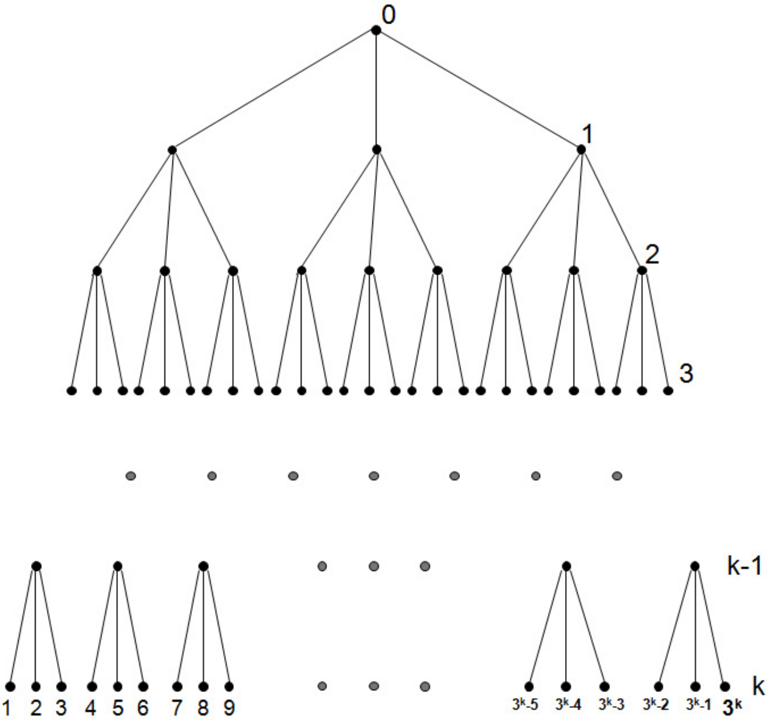


Fig. 7.1 The computational tree of a non-deterministic Turing machine $\mathcal{M}_N$ having the non-deterministic degree $d = 3$

j computational steps for each $0 \leqslant j \leqslant k$, the deterministic Turing machine $\mathcal{M}_D$ will execute $K_{\mathcal{M}_D}$ steps. In particular, if $\mathcal{M}_N$ reaches a final configuration (e.g., it accepts a string) in $k \geqslant 0$ steps and if $\mathcal{M}_D$ could consider only the d^k computational paths which consist of k computational steps, it will executes at most kd^k steps for reaching this configuration.

These results show an exponential growth of the time required for reaching a final configuration by the deterministic Turing machine $\mathcal{M}_D$ with respect to the time required by the non-deterministic Turing machine $\mathcal{M}_N$, assuming that the time required for both machines for a single step is the same. However, in the classic theory on Turing machines it is not known if there is a more efficient simulation of $\mathcal{M}_N$. In other words, it is an important and open problem of Computer Science theory to demonstrate that it is not possible to simulate a non-deterministic Turing machine by a deterministic Turing machine with a sub-exponential numbers of steps.

7.3 The Grossone Language and Methodology

In this section, we give just a brief introduction to the methodology of the new approach [30,32] dwelling only on the issues directly related to the subject of the chapter. This methodology will be used in Section 7.4 to study Turing machines and to obtain some more accurate results with respect to those obtainable by using the traditional framework [4,44] .

In order to start, let us remind that numerous trials have been done during the centuries to evolve existing numeral systems in such a way that numerals representing infinite and infinitesimal numbers could be included in them (see [2,3,5, 17,18,25,28,46]). Since new numeral systems appear very rarely, in each concrete historical period their significance for Mathematics is very often underestimated (especially by pure mathematicians). In order to illustrate their importance, let us remind the Roman numeral system that does not allow one to express zero and negative numbers. In this system, the expression III-X is an indeterminate form. As a result, before appearing the positional numeral system and inventing zero mathematicians were not able to create theorems involving zero and negative numbers and to execute computations with them.

There exist numeral systems that are even weaker than the Roman one. They seriously limit their users in executing computations. Let us recall a study published recently in *Science* (see [11]). It describes a primitive tribe living in Amazonia (Pirahã). These people use a very simple numeral system for counting: one, two, many. For Pirahã, all quantities larger than two are just 'many' and such operations as 2+2 and 2+1 give the same result, i.e., 'many'. Using their weak numeral system Pirahã are not able to see, for instance, numbers 3, 4, 5, and 6, to execute arithmetical operations with them, and, in general, to say anything about these numbers because in their language there are neither words nor concepts for that.

In the context of the present chapter, it is very important that the weakness of Pirahã's numeral system leads them to such results as

$$\text{`many'} + 1 = \text{`many'}, \qquad \text{`many'} + 2 = \text{`many'}, \tag{7.6}$$

which are very familiar to us in the context of views on infinity used in the traditional calculus

$$\infty + 1 = \infty, \qquad \infty + 2 = \infty. \tag{7.7}$$

The arithmetic of Pirahã involving the numeral 'many' has also a clear similarity with the arithmetic proposed by Cantor for his Alephs[2]:

$$\aleph_0 + 1 = \aleph_0, \qquad \aleph_0 + 2 = \aleph_0, \qquad \aleph_1 + 1 = \aleph_1, \qquad \aleph_1 + 2 = \aleph_1. \tag{7.8}$$

[2] This similarity becomes even more pronounced if one considers another Amazonian tribe – Mundurukú (see [26]) – who fail in exact arithmetic with numbers larger than 5 but are able to compare and add large approximate numbers that are far beyond their naming range. Particularly, they use the words 'some, not many' and 'many, really many' to distinguish two types of large numbers using the rules that are very similar to ones used by Cantor to operate with $\aleph_0$ and $\aleph_1$, respectively.

Thus, the modern mathematical numeral systems allow us to distinguish a larger quantity of finite numbers with respect to Pirahã but give results that are similar to those of Pirahã when we speak about infinite quantities. This observation leads us to the following idea: *Probably our difficulties in working with infinity is not connected to the nature of infinity itself but is a result of inadequate numeral systems that we use to work with infinity, more precisely, to express infinite numbers.*

The approach developed in [30, 32, 37] proposes a numeral system that uses the same numerals for several different purposes for dealing with infinities and infinitesimals: in Analysis for working with functions that can assume different infinite, finite, and infinitesimal values (functions can also have derivatives assuming different infinite or infinitesimal values); for measuring infinite sets; for indicating positions of elements in ordered infinite sequences ; in probability theory, etc. (see [7, 8, 13, 22, 29, 31, 33–36, 38–40, 45, 47, 48]). It is important to emphasize that the new numeral system avoids situations of the type (7.6)–(7.8) providing results ensuring that if a is a numeral written in this system then for any a (i.e., a can be finite, infinite, or infinitesimal) it follows $a+1>a$.

The new numeral system works as follows. A new infinite unit of measure expressed by the numeral ① called *grossone* is introduced as the number of elements of the set, $\mathbb{N}$, of natural numbers. Concurrently with the introduction of grossone in the mathematical language all other symbols (like ∞, Cantor's ω, $\aleph_0, \aleph_1, ...$, etc.) traditionally used to deal with infinities and infinitesimals are excluded from the language because grossone and other numbers constructed with its help not only can be used instead of all of them but can be used with a higher accuracy[3]. Grossone is introduced by describing its properties postulated by the Infinite Unit Axiom (see [32, 37]) added to axioms for real numbers (similarly, in order to pass from the set, $\mathbb{N}$, of natural numbers to the set, $\mathbb{Z}$, of integers a new element – zero expressed by the numeral 0 – is introduced by describing its properties) .

The new numeral ① allows us to construct different numerals expressing different infinite and infinitesimal numbers and to execute computations with them. Let us give some examples. For instance, in Analysis, indeterminate forms are not present and, for example, the following relations hold for ① and $①^{-1}$ (that is infinitesimal), as for any other (finite, infinite, or infinitesimal) number expressible in the new numeral system

$$0\cdot ① = ①\cdot 0 = 0, \quad ① - ① = 0, \quad \frac{①}{①} = 1, \quad ①^0 = 1, \quad 1^{①} = 1, \quad 0^{①} = 0, \qquad (7.9)$$

$$0\cdot ①^{-1} = ①^{-1}\cdot 0 = 0, \quad ①^{-1} > 0, \quad ①^{-2} > 0, \quad ①^{-1} - ①^{-1} = 0, \qquad (7.10)$$

$$\frac{①^{-1}}{①^{-1}} = 1, \quad \frac{①^{-2}}{①^{-2}} = 1, \quad (①^{-1})^0 = 1, \quad ①\cdot ①^{-1} = 1, \quad ①\cdot ①^{-2} = ①^{-1}. \qquad (7.11)$$

The new approach gives the possibility to develop a new Analysis (see [35]) where functions assuming not only finite values but also infinite and infinitesimal

[3] Analogously, when the switch from Roman numerals to the Arabic ones has been done, numerals X, V, I, etc. have been excluded from records using Arabic numerals.

ones can be studied. For all of them it becomes possible to introduce a new notion of continuity that is closer to our modern physical knowledge. Functions assuming finite and infinite values can be differentiated and integrated.

By using the new numeral system it becomes possible to measure certain infinite sets and to see, e.g., that the sets of even and odd numbers have $\text{①}/2$ elements each. The set, $\mathbb{Z}$, of integers has $2\text{①}+1$ elements (① positive elements, ① negative elements, and zero). Within the countable sets and sets having cardinality of the continuum (see [20, 36, 37]) it becomes possible to distinguish infinite sets having different number of elements expressible in the numeral system using grossone and to see that, for instance,

$$\frac{\text{①}}{2} < \text{①}-1 < \text{①} < \text{①}+1 < 2\text{①}+1 < 2\text{①}^2-1 < 2\text{①}^2 < 2\text{①}^2+1 <$$

$$2\text{①}^2+2 < 2^{\text{①}}-1 < 2^{\text{①}} < 2^{\text{①}}+1 < 10^{\text{①}} < \text{①}^{\text{①}}-1 < \text{①}^{\text{①}} < \text{①}^{\text{①}}+1. \quad (7.12)$$

Another key notion for our study of Turing machines is that of infinite sequence. Thus, before considering the notion of the Turing machine from the point of view of the new methodology, let us explain how the notion of the infinite sequence can be viewed from the new positions.

7.3.1 Infinite Sequences

Traditionally, an *infinite sequence* $\{a_n\}, a_n \in A$, $n \in \mathbb{N}$, is defined as a function having the set of natural numbers, $\mathbb{N}$, as the domain and a set A as the codomain. A *subsequence* $\{b_n\}$ is defined as a sequence $\{a_n\}$ from which some of its elements have been removed . In spite of the fact that the removal of the elements from $\{a_n\}$ can be directly observed, the traditional approach does not allow one to register, in the case where the obtained subsequence $\{b_n\}$ is infinite, the fact that $\{b_n\}$ has less elements than the original infinite sequence $\{a_n\}$.

Let us study what happens when the new approach is used. From the point of view of the new methodology, an infinite sequence can be considered in a dual way: either as an object of a mathematical study or as a mathematical instrument developed by human beings to observe other objects and processes. First, let us consider it as a mathematical object and show that the definition of infinite sequences should be done more precise within the new methodology. In the finite case, a sequence $a_1, a_2, \dots, a_n$ has n elements and we extend this definition directly to the infinite case saying that an infinite sequence $a_1, a_2, \dots, a_n$ has n elements where n is expressed by an infinite numeral such that the operations with it satisfy the Postulate 3 of the Grossone methodology[4]. Then the following result (see [30, 32]) holds. We reproduce here its proof for the sake of completeness.

[4] The Postulate 3 states: *The part is less than the whole* is applied to all numbers (finite, infinite, and infinitesimal) and to all sets and processes (finite and infinite), see [30].

Theorem 1. *The number of elements of any infinite sequence is less or equal to ①.*

Proof. The new numeral system allows us to express the number of elements of the set $\mathbb{N}$ as ①. Thus, due to the sequence definition given above, any sequence having $\mathbb{N}$ as the domain has ① elements.

The notion of subsequence is introduced as a sequence from which some of its elements have been removed. This means that the resulting subsequence will have less elements than the original sequence. Thus, we obtain infinite sequences having the number of members less than grossone. □

It becomes appropriate now to define the *complete sequence* as an infinite sequence containing ① elements . For example, the sequence of natural numbers is complete, the sequences of even and odd natural numbers are not complete because they have $\frac{①}{2}$ elements each (see [30, 32]). Thus, the new approach imposes a more precise description of infinite sequences than the traditional one: to define a sequence $\{a_n\}$ in the new language, it is not sufficient just to give a formula for a_n, we should determine (as it happens for sequences having a finite number of elements) its number of elements and/or the first and the last elements of the sequence. If the number of the first element is equal to one, we can use the record $\{a_n : k\}$ where a_n is, as usual, the general element of the sequence and k is the number (that can be finite or infinite) of members of the sequence; the following example clarifies these concepts.

Example 1. Let us consider the following three sequences:

$$\{a_n : ①\} = \{4, \quad 8, \quad \dots \quad 4(① - 1), \quad 4①\}; \tag{7.13}$$

$$\{b_n : \frac{①}{2} - 1\} = \{4, \quad 8, \quad \dots \quad 4(\frac{①}{2} - 2), \quad 4(\frac{①}{2} - 1)\}; \tag{7.14}$$

$$\{c_n : \frac{2①}{3}\} = \{4, \quad 8, \quad \dots \quad 4(\frac{2①}{3} - 1), \quad 4\frac{2①}{3}\}. \tag{7.15}$$

The three sequences have $a_n = b_n = c_n = 4n$ but they are different because they have different number of members. Sequence $\{a_n\}$ has ① elements and, therefore, is complete, $\{b_n\}$ has $\frac{①}{2} - 1$, and $\{c_n\}$ has $2\frac{①}{3}$ elements. □

Let us consider now infinite sequences as one of the instruments used by mathematicians to study the world around us and other mathematical objects and processes. The first immediate consequence of Theorem 1 is that any *sequential* process can have at maximum ① elements. This means that a process of sequential observations of any object cannot contain more than ① steps[5]. We are not able

[5] It is worthy to notice a deep relation of this observation to the Axiom of Choice. Since Theorem 1 states that any sequence can have at maximum ① elements, so this fact holds for the process of a sequential choice, as well. As a consequence, it is not possible to choose sequentially more than ① elements from a set. This observation also emphasizes the fact that the parallel computational paradigm is significantly different with respect to the sequential one because p parallel processes can choose $p \cdot ①$ elements from a set.

to execute any infinite process physically but we assume the existence of such a process; moreover, only a finite number of observations of elements of the considered infinite sequence can be executed by a human who is limited by the numeral system used for the observation. Indeed, we can observe only those members of a sequence for which there exist the corresponding numerals in the chosen numeral system; to better clarify this point the following example is discussed.

Example 2. Let us consider the numeral system, $\mathcal{P}$, of Pirahã able to express only numbers 1 and 2. If we add to $\mathcal{P}$ the new numeral ①, we obtain a new numeral system (we call it $\widehat{\mathcal{P}}$). Let us consider now a sequence of natural numbers $\{n : ①\}$. It goes from 1 to ① (note that both numbers, 1 and ①, can be expressed by numerals from $\widehat{\mathcal{P}}$). However, the numeral system $\widehat{\mathcal{P}}$ is very weak and it allows us to observe only ten numbers from the sequence $\{n : ①\}$ represented by the following numerals

$$\underbrace{1,2,}_{finite} \quad \ldots \quad \underbrace{\frac{①}{2}-2, \frac{①}{2}-1, \frac{①}{2}, \frac{①}{2}+1, \frac{①}{2}+2,}_{infinite} \quad \ldots \quad \underbrace{①-2, ①-1, ①}_{infinite}. \qquad (7.16)$$

The first two numerals in (7.16) represent finite numbers, the remaining eight numerals express infinite numbers, and dots represent members of the sequence of natural numbers that are not expressible in $\widehat{\mathcal{P}}$ and, therefore, cannot be observed if one uses only this numeral system for this purpose. □

In the light of the limitations concerning the process of sequential observations, the researcher can choose how to organize the required sequence of observations and which numeral system to use for it, defining so which elements of the object he/she can observe. This situation is exactly the same as in natural sciences: before starting to study a physical object, a scientist chooses an instrument and its accuracy for the study.

Example 3. Let us consider the set A=$\{1,2,3,\ldots,2①\text{-}1,2①\}$ as an object of our observation. Suppose that we want to organize the process of the sequential counting of its elements. Then, due to Theorem 1, starting from the number 1 this process can arrive at maximum to ①. If we consider the complete counting sequence $\{n : ①\}$, then we obtain

$$\underbrace{1,2,\ 3,\ 4,\ \ldots\ ①-2, ①-1, ①,}_{①\ \text{steps}} ①+1, ①+2, ①+3, \ldots, 2①-1, 2① \qquad (7.17)$$

Analogously, if we start the process of the sequential counting from 5, the process arrives at maximum to $① + 4$:

$$1,2,3,4,\underbrace{5\ \ldots\ ①-1, ①, ①+1, ①+2, ①+3, ①+4,}_{①\ \text{steps}} ①+5, \ldots, 2①-1, 2① \qquad (7.18)$$

The corresponding complete sequence used in this case is $\{n+4 : ①\}$. We can also change the length of the step in the counting sequence and consider, for instance, the complete sequence $\{2n-1 : ①\}$:

$$\underbrace{1,2,3,4,\ \dots\ ①\text{-}1,①,①\text{+}1,①\text{+}2,\ \dots\ 2①\text{-}3,2①\text{-}2,2①\text{-}1}_{①\ \text{steps}},2① \tag{7.19}$$

If we use again the numeral system $\widehat{\mathcal{P}}$, then among finite numbers it allows us to see only number 1 because already the next number in the sequence, 3, is not expressible in $\widehat{\mathcal{P}}$. The last element of the sequence is $2①-1$ and $\widehat{\mathcal{P}}$ allows us to observe it. □

The introduced definition of the sequence allows us to work not only with the first but with any element of any sequence if the element of our interest is expressible in the chosen numeral system independently whether the sequence under our study has a finite or an infinite number of elements. Let us use this new definition for studying infinite sets of numerals, in particular, for calculating the number of points at the interval $[0,1)$ (see [30, 32]). To do this we need a definition of the term 'point' and mathematical tools to indicate a point. If we accept (as is usually done in modern Mathematics) that a *point* A belonging to the interval $[0,1)$ is determined by a numeral x, $x \in \mathbb{S}$, called *coordinate of the point* A where $\mathbb{S}$ is a set of numerals, then we can indicate the point A by its coordinate x and we are able to execute the required calculations.

It is worthwhile to emphasize that giving this definition we have not used the usual formulation "*x belongs to the set,* $\mathbb{R}$*, of real numbers*". This has been done because we can express coordinates only by numerals and different choices of numeral systems lead to different sets of numerals and, as a result, to different sets of numbers observable through the chosen numerals. In fact, we can express coordinates only after we have fixed a numeral system (our instrument of the observation) and this choice defines which points we can observe, namely, points having coordinates expressible by the chosen numerals. This situation is typical for natural sciences where it is well known that instruments influence the results of observations. Remind the work with a microscope: we decide the level of the precision we need and obtain a result which is dependent on the chosen level of accuracy. If we need a more precise or a more rough answer, we change the lens of our microscope.

We should decide now which numerals we shall use to express coordinates of the points. After this choice we can calculate the number of numerals expressible in the chosen numeral system and, as a result, we obtain the number of points at the interval $[0,1)$. Different variants (see [30, 32]) can be chosen depending on the precision level we want to obtain. For instance, we can choose a positional numeral system with a finite radix b that allows us to work with numerals

$$(0.a_1a_2\dots a_{(①-1)}a_{①})_b, \qquad a_i \in \{0,1,\dots b-2,b-1\}, \quad 1 \le i \le ①. \tag{7.20}$$

Then, the number of numerals (7.20) gives us the number of points within the interval $[0,1)$ that can be expressed by these numerals. Note that a number using the positional numeral system (7.20) cannot have more than grossone digits (contrarily to sets discussed in Example 3) because a numeral having $g > ①$ digits would not be observable in a sequence. In this case ($g > ①$) such a record becomes useless in sequential computations because it does not allow one to identify numbers entirely since $g - ①$ numerals remain non observed.

Theorem 2. *If coordinates of points $x \in [0,1)$ are expressed by numerals (7.20), then the number of the points x over $[0,1)$ is equal to $b^{①}$.*

Proof. In the numerals (7.20) there is a sequence of digits, $a_1 a_2 \dots a_{(①-1)} a_{①}$, used to express the fractional part of the number. Due to the definition of the sequence and Theorem 1, any infinite sequence can have at maximum ① elements. As a result, there is ① positions on the right of the dot that can be filled in by one of the b digits from the alphabet $\{0,1,\dots,b-1\}$ that leads to $b^{①}$ possible combinations. Hence, the positional numeral system using the numerals of the form (7.20) can express $b^{①}$ numbers. □

Corollary 1. *The number of numerals*

$$(a_1 a_2 a_3 \dots a_{①-2} a_{①-1} a_{①})_b, \qquad a_i \in \{0,1,\dots b-2, b-1\}, \qquad 1 \le i \le ①, \tag{7.21}$$

expressing integers in the positional system with a finite radix b in the alphabet $\{0,1,\dots b-2, b-1\}$ is equal to $b^{①}$.

Proof. The proof is a straightforward consequence of Theorem 2 and is so omitted. □

Corollary 2. *If coordinates of points $x \in (0,1)$ are expressed by numerals (7.20), then the number of the points x over $(0,1)$ is equal to $b^{①} - 1$.*

Proof. The proof follows immediately from Theorem 2. □

Note that Corollary 2 shows that it becomes possible now to observe and to register the difference of the number of elements of two infinite sets (the interval $[0,1)$ and the interval $(0,1)$, respectively) even when only one element (the point 0, expressed by the numeral $0.00\dots0$ with ① zero digits after the decimal point) has been excluded from the first set in order to obtain the second one.

7.4 Observing Turing Machines through the Lens of the Grossone Methodology

In this Section the different types of Turing machines introduced in Section 7.2 are analyzed and observed by using as instruments of observation the Grossone language and methodology presented in Section 7.3 . In particular, after introducing a distiction between physical and ideal Turing machine (see Section 7.4.1), some results for Single-tape and Multi-tape Turing machines are summarized (see Sections 7.4.2 and 7.4.3 respectively), then a discussion about the equivalence between Single and Multi-tape Turing machine is reported in Section 7.4.4. Finally, a comparison between deterministic and non-deterministic Turing machines through the lens of the Grossone methodology is presented in Section 7.4.5.

7.4.1 Physical and Ideal Turing Machines

Before starting observing Turing machines by using the Grossone methodology, it is useful to recall the main results showed in the previous Section: (i) a (complete) sequence can have maximum ① elements; (ii) the elements which we are able to observe in this sequence depend on the adopted numeral system. Moreover, a distiction between physical and ideal Turing machines should be introduced. Specifically, the machines defined in Section 7.2 (e.g. the Single-Tape Turing machine of Section 7.2.1) are called ideal Turing machine, $\mathcal{T}^{\mathcal{I}}$. Howerver, in order to study the limitations of practical automatic computations, we also consider machines, $\mathcal{T}^{\mathcal{P}}$, that can be constructed physically. They are identical to $\mathcal{T}^{\mathcal{I}}$ but are able to work only a finite time and can produce only finite outputs. In this Section, both kinds of machines are analyzed from the point of view of their outputs, called by Turing ‘computable numbers’ or ‘computable,sequences’, and from the point of view of computations that the machines can execute .

Let us consider first a physical machine $\mathcal{T}^{\mathcal{P}}$ and discuss about the number of computational steps it can execute and how the obtained results then can be interpreted by a human observer (e.g. the researcher) . We suppose that its output is written on the tape using an alphabet Σ containing b symbols $\{0,1,\dots b-2,b-1\}$ where b is a finite number (Turing uses $b = 10$).Thus, the output consists of a sequence of digits that can be viewed as a number in a positional system $\mathcal{B}$ with the radix b. By definition, $\mathcal{T}^{\mathcal{P}}$ should stop after a finite number of iterations. The magnitude of this value depends on the physical construction of the machine, the way the notion ‘iteration’ has been defined, etc., but in any case this number is finite.

A physical machine $\mathcal{T}^{\mathcal{P}}$ stops in two cases: (i) it has finished the execution of its program and stops; (ii) it stops because its breakage. In both cases the output sequence

$$(a_1a_2a_3\dots a_{k-1},a_k)_b, \qquad a_i \in \{0,1,\dots b-2,b-1\}, \quad 1 \leq i \leq k,$$

of $\mathcal{T}^{\mathcal{P}}$ has a finite length k.

If the maximal length of the output sequence that can be computed by $\mathcal{T}^{\mathcal{P}}$ is equal to a finite number $K_{\mathcal{T}^{\mathcal{P}}}$, then it follows $k \leq K_{\mathcal{T}^{\mathcal{P}}}$. This means that there exist problems that cannot be solved by $\mathcal{T}^{\mathcal{P}}$ if the length of the output outnumbers $K_{\mathcal{T}^{\mathcal{P}}}$. If a physical machine $\mathcal{T}^{\mathcal{P}}$ has stopped after it has printed $K_{\mathcal{T}^{\mathcal{P}}}$ symbols, then it is not clear whether the obtained output is a solution or just a result of the depletion of its computational resources.

In particular, with respect to the halting problem it follows that all algorithms stop on $\mathcal{T}^{\mathcal{P}}$.

In order to be able to read and to understand the output, the researcher (the user) should know a positional numeral system $\mathcal{U}$ with an alphabet $\{0,1,\dots u-2,u-1\}$ where $u \geq b$. Otherwise, the output cannot be decoded by the user. Moreover, the researcher must be able to observe a number of symbols at least equal to the maximal length of the output sequence that can be computed by machine (i.e., $K_{\mathcal{U}} \geq K_{\mathcal{T}^{\mathcal{P}}}$).

If the situation $K_{\mathcal{U}} < K_{\mathcal{T}^{\mathcal{P}}}$ holds, then this means that the user is not able to interpret the obtained result. Thus, the number $K^* = \min\{K_{\mathcal{U}}, K_{\mathcal{T}^{\mathcal{P}}}\}$ defines the length of the outputs that can be computed and interpreted by the user.

As a consequence, algorithms producing outputs having more than K^* positions become less interesting from the practical point of view.

After having introduced the distinction between physical and ideal Turing machines, let us analyze and observe them through the lens of the Grossone Methodology. Specifically, the results obtained and discussed in [42] for deterministic and non-deterministic Single-tape Turing machines are summarized in Section 7.4.2 and 7.4.4 respectively; whereas, Section 7.4.3 reports additional results for Multi-tape Turing machines (see [43]).

7.4.2 Observing Single-Tape Turing Machines

As stated in Section 7.4.1, single-tape ideal Turing machines $\mathcal{M}^I$ (see Section 7.2.1) can produce outputs with an infinite number of symbols k. However, in order to be observable in a sequence, an output should have $k \leq \text{①}$ (see Section 7.3). Starting from these considerations the following theorem can be introduced.

Theorem 3. *Let M be the number of all possible complete computable sequences that can be produced by ideal single-tape Turing machines using outputs being numerals in the positional numeral system $\mathcal{B}$. Then it follows $M \leq b^{\text{①}}$.*

Proof. This result follows from the definitions of the complete sequence and the form of numerals

$$(a_{-1}a_{-2}\dots a_{-(\text{①}-1)}a_{-\text{①}})_b, \ a_{-i} \in \{0,1,\dots b-2, b-1\}, \ 1 \leq i \leq \text{①},$$

that are used in the positional numeral system $\mathcal{B}$. □

Corollary 3. *Let us consider an ideal Turing machine $\mathcal{M}^I_1$ working with the alphabet $\{0,1,2\}$ and computing the following complete computable sequence*

$$\underbrace{0,1,2,0,1,2,0,1,2,\ \ldots\ 0,1,2,0,1,2}_{\text{①}\ \textit{positions}}. \tag{7.22}$$

Then ideal Turing machines working with the output alphabet $\{0,1\}$ *cannot produce observable in a sequence outputs computing (7.22).*

Since the numeral 2 does not belong to the alphabet $\{0,1\}$ it should be coded by more than one symbol. One of codifications using the minimal number of symbols in the alphabet $\{0,1\}$ necessary to **code** numbers $0,1,2$ is $\{00,01,10\}$. Then the output corresponding to (7.22) and computed in this codification should be

$$00,01,10,00,01,10,00,01,10,\ \ldots\ 00,01,10,00,01,10. \tag{7.23}$$

Since the output (7.22) contains grossone positions, the output (7.23) should contain 2① positions. However, in order to be observable in a sequence, (7.23) should not have more than grossone positions. This fact completes the proof. □

The mathematical language used by Turing did not allow one to distinguish these two machines. Now we are able to distinguish a machine from another also when we consider infinite sequences. Turing's results and the new ones do not contradict each other. Both languages observe and describe the same object (computable sequences) but with different accuracies.

It is not possible to describe a Turing machine (the object of the study) without the usage of a numeral system (the instrument of the study). As a result, it becomes not possible to speak about an absolute number of all possible Turing machines $\mathcal{T}^{\mathcal{I}}$. It is always necessary to speak about the number of all possible Turing machines $\mathcal{T}^{\mathcal{I}}$ expressible in a fixed numeral system (or in a group of them).

Theorem 4. *The maximal number of complete computable sequences produced by ideal Turing machines that can be enumerated in a sequence is equal to* ①.

We have established that the number of complete computable sequences that can be computed using a fixed radix b is less or equal $b^{\text{①}}$. However, we do not know how many of them can be results of computations of a Turing machine. Turing establishes that their number is enumerable. In order to obtain this result, he used the mathematical language developed by Cantor and this language did not allow him to distinguish sets having different infinite numbers of elements. The introduction of grossone gives a possibility to execute a more precise analysis and to distinguish within enumerable sets infinite sets having different numbers of elements. For instance, the set of even numbers has $\frac{\text{①}}{2}$ elements and the set of integer numbers has $2\text{①}+1$ elements. If the number of complete computable sequences, $M_{\mathcal{T}^{\mathcal{I}}}$, is larger than ①, then there can be differen sequential processes that enumerate different sequences of complete computable sequences. In any case, each of these enumerating sequential processes cannot contain more than grossone members.

7.4.3 Observing Multi-tape Turing Machines

Before starting to analyze the computations performed by an ideal k-tapes Turing machine (with $k \geq 2$) $\mathcal{M}_K^I = \left\langle Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta^{(k)} \right\rangle$ (see (7.1), see Section 7.2.2), it is worth to make some considerations about the process of observation itself in the light of the Grossone methodology. As discussed above, if we want to observe the process of computation performed by a Turing machine while it executes an algorithm, then we have to execute observations of the machine in a sequence of moments. In fact, it is not possible to organize a continuous observation of the machine. Any instrument used for an observation has its accuracy and there always be a minimal period of time related to this instrument allowing one to distinguish two different moments of time and, as a consequence, to observe (and to register) the states of the object in these two moments. In the period of time passing between these two moments the object remains unobservable.

Since our observations are made in a sequence, the process of observations can have at maximum ① elements. This means that inside a computational process it is possible to fix more than grossone steps (defined in a way) but it is not possible to count them one by one in a sequence containing more than grossone elements. For instance, in a time interval $[0, 1)$, up to $b^{①}$ numerals of the type (7.20) can be used to identify moments of time but not more than grossone of them can be observed in a sequence. Moreover, it is important to stress that any process itself, considered independently on the researcher, is not subdivided in iterations, intermediate results, moments of observations, etc. The structure of the language we use to describe the process imposes what we can say about the process (see [42] for a detailed discussion).

On the basis of the considerations made above, we should choose the accuracy (granularity) of the process of the observation of a Turing machine; for instance we can choose a single operation of the machine such as reading a symbol from the tape, or moving the tape, etc. However, in order to be close as much as possible to the traditional results, we consider an application of the transition function of the machine as our observation granularity (see Section 7.2).

Moreover, concerning the output of the machine, we consider the symbols written on all the k tapes of the machine by using, on each tape i, with $1 \leq i \leq k$, the alphabet Σ_i of the tape, containing b_i symbols, plus the blank symbol ($\bar{b}$). Due to the definition of complete sequence (see Section 7.3) on each tape at least ① symbols can be produced and observed. This means that on a tape i, after the last symbols belonging to the tape alphabet Σ_i, if the sequence is not complete (i.e., if it has less than ① symbols) we can consider a number of blank symbols ($\bar{b}$) necessary to complete the sequence. We say that we are considering a *complete output* of a k-tapes Turing machine when on each tape of the machine we consider a complete sequence of symbols belonging to $\Sigma_i \cup \{\bar{b}\}$.

Theorem 5. *Let $\mathcal{M}_K^I = \left\langle Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta^{(k)} \right\rangle$ be an ideal k-tapes, $k \geq 2$, Turing machine. Then, a complete output of the machine will results in k① symbols.*

Proof. Due to the definition of the complete sequence, on each tape at maximum ① symbols can be produced and observed and thus by considering a complete sequence on each of the k tapes of the machine the complete output of the machine will result in k① symbols. □

Having proved that a complete output that can be produced by a k-tapes Turing machine results in k① symbols, it is interesting to investigate what part of the complete output produced by the machine can be observed in a sequence taking into account that it is not possible to observe in a sequence more than ① symbols (see Section 7.3). As examples, we can decide to make in a sequence one of the following observations: (i) ① symbols on one among the k-tapes of the machine, (ii) $\frac{①}{k}$ symbols on each of the k-tapes of the machine; (iii) $\frac{①}{2}$ symbols on 2 among the k-tapes of the machine, an so on.

Theorem 6. *Let $\mathcal{M}_K^I = \left\langle Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta^{(k)} \right\rangle$ be an ideal k-tapes, $k \geq 2$, Turing machine. Let M be the number of all possible complete outputs that can be produced by $\mathcal{M}_K^I$. Then it follows $M = \prod_{i=1}^{k} (b_i + 1)^{①}$.*

Proof. Due to the definition of the complete sequence, on each tape i, with $1 \leq i \leq k$, at maximum ① symbols can be produced and observed by using the b_i symbols of the alphabet Σ_i of the tape plus the blank symbol ($\bar{b}$); as a consequence, the number of all the possible complete sequences that can be produced and observed on a tape i is $(b_i + 1)^{①}$. A complete output of the machine is obtained by considering a complete sequence on each of the the k-tapes of the machine, thus by considering all the possible complete sequences that can be produced and observed on each of the k tapes of the machine, the number M of all the possible complete outputs will results in $\prod_{i=1}^{k} (b_i + 1)^{①}$. □

As the number $M = \prod_{i=1}^{k} (b_i + 1)^{①}$ of complete outputs that can be produced by $\mathcal{M}_K$ is larger than grossone, then there can be different sequential enumerating processes that enumerate complete outputs in different ways, in any case, each of these enumerating sequential processes cannot contain more than grossone members (see Section 7.3).

7.4.4 Comparing Different Multi-tape Machines and Multi and Single-Tape Machines

In the classical framework ideal k-tape Turing machines have the same computational power of Single-tape Turing machines and given a Multi-tape Turing machine $\mathcal{M}_K^I$ it is always possible to define a Single-tape Turing machine which is able to fully simulate its behavior and therefore to completely execute its computations. As showed for Single-tape Turing machine (see [42]), the Grossone methodology allows us to give a more accurate definition of the equivalence among different machines as it provides the possibility not only to separate different classes of infinite sets with respect to their cardinalities but also to measure the number of elements of some of them. With reference to Multi-tape Turing machines, the Single-tape Turing machines adopted for their simulation use a particular kind of tape which is divided into tracks (multi-track tape). In this way, if the tape has m tracks, the head is able to access (for reading and/or writing) all the m characters on the tracks during a single operation. This tape organization leads to a straightforward definition of the behavior of a Single-tape Turing machine able to completely execute the computations of a given Multi-tape Turing machine (see Section 7.2.2). However, the so defined Single-tape Turing machine $\mathcal{M}^I$, to simulate t computational steps of $\mathcal{M}^I_K$, needs to execute $O(t^2)$ transitions (t^2+t in the worst case) and to use an alphabet with $2^k(|\Sigma_1|+1)\prod_{i=2}^{k}(|\Sigma_i|+2)$ symbols (again see Section 7.2.2). By exploiting the Grossone methodology is is possibile to obtain the following result that has a higher accuracy with respect to that provided by the traditional framework.

Theorem 7. *Let us consider* $\mathcal{M}^I_K = \left\langle Q, \Gamma, \bar{b}, \Sigma, q_0, F, \delta^{(k)} \right\rangle$*,a k-tapes,* $k \geq 2$*, Turing machine, where* $\Sigma = \bigcup_{i=1}^{k} \Sigma_i$ *is given by the union of the symbols in the k tape alphabets* $\Sigma_1, \ldots, \Sigma_k$ *and* $\Gamma = \Sigma \cup \{\bar{b}\}$*. If this machine performs t computational steps such that*

$$t \leqslant \frac{1}{2}(\sqrt{4\text{①}+1}-1), \tag{7.24}$$

then there exists $\mathcal{M}^I_1 = \{Q', \Gamma', \bar{b}, \Sigma', q'_0, F', \delta'\}$*, an equivalent Single-tape Turing machine with* $|\Gamma'| = 2^k(|\Sigma_1|+1)\prod_{i=2}^{k}(|\Sigma_i|+2)$*, which is able to simulate* $\mathcal{M}^I_K$ *and can be observed in a sequence.*

Proof. Let us recall that the definition of $\mathcal{M}^I_1$ requires for a Single-tape to be divided into $2k$ tracks; k tracks for storing the characters in the k tapes of $\mathcal{M}^I_K$ and k tracks for signing through the marker $\downarrow$ the positions of the k heads on the k tapes of $\mathcal{M}^I_k$ (see Section 7.2.2). The transition function $\delta^{(k)}$ of the k-tapes machine is given by $\delta^{(k)}(q_1, a_{i1}, \ldots, a_{ik}) = (q_j, a_{j_1}, \ldots, a_{j_k}, z_{j_1}, \ldots, z_{j_k})$, with $z_{j_1}, \ldots, z_{j_k} \in \{R, L, N\}$; as a consequence the corresponding transition function δ' of the Single-tape machine, for each transition specified by $\delta^{(k)}$ must individuate the current state and the position of the marker for each track and then write on the tracks the required symbols, move the markers and go in another internal state. For each computational step of $\mathcal{M}^I_K$, $\mathcal{M}^I_1$ must execute a sequence of steps for covering the portion of tapes between the two most distant markers. As in each computational step a marker can move at most of one cell and then two markers can move away each other at most of two cells, after t steps of $\mathcal{M}^I_K$ the markers can be at most $2t$ cells distant, thus if $\mathcal{M}^I_K$ executes t steps, $\mathcal{M}^I_1$ executes at most: $2\sum_{i=1}^{t} i = t^2+t$ steps. In order to be observable in a sequence the number t^2+t of steps, performed by $\mathcal{M}^I_1$ to simulate t steps of $\mathcal{M}^I_K$, must be less than or equal to ①. Namely, it should be $t^2+t \leqslant \text{①}$. The fact that this inequality is satisfied for $t \leqslant \frac{1}{2}(\sqrt{4\text{①}+1}-1)$ completes the proof. □

7.4.5 Comparing Deterministic and Non-deterministic Turing Machines

Let us discuss the traditional and new results regarding the computational power of deterministic and non-deterministic Turing machines.

Classical results show an exponential growth of the time required for reaching a final configuration by the deterministic Turing machine $\mathcal{M}_D$ with respect to the time required by the non-deterministic Turing machine $\mathcal{M}_N$, assuming that the time required for both machines for a single step is the same. However, in the classic theory on Turing machines it is not known if there is a more efficient simulation of $\mathcal{M}_N$. In other words, it is an important and open problem of Computer Science theory to demonstrate that it is not possible to simulate a non-deterministic Turing machine by a deterministic Turing machine with a sub-exponential numbers of steps.

Let us now return to the new mathematical language. Since the main interest to non-deterministic Turing machines (7.5) is related to their theoretical properties, hereinafter we start by a comparison of ideal deterministic Turing machines, $\mathcal{T}^{\mathcal{I}}$, with ideal non-deterministic Turing machines $\mathcal{T}^{\mathcal{I}\mathcal{N}}$. Physical machines $\mathcal{T}^{\mathcal{P}}$ and $\mathcal{T}^{\mathcal{P}\mathcal{N}}$ are considered at the end of this section. By taking into account the results of Section 7.4.4, the proposed approach can be applied both to single and multi-tape machines, however, single-tape machines are considered in the following.

Due to the analysis made in Section 7.4.3, we should choose the accuracy (granularity) of processes of observation of both machines, $\mathcal{T}^{\mathcal{I}}$ and $\mathcal{T}^{\mathcal{I}\mathcal{N}}$. In order to be close as much as possible to the traditional results, we consider again an application of the transition function of the machine as our observation granularity. With respect to $\mathcal{T}^{\mathcal{I}\mathcal{N}}$ this means that the nodes of the computational tree are observed. With respect to $\mathcal{T}^{\mathcal{I}}$ we consider sequences of such nodes. For both cases the initial configuration is not observed, i.e., we start our observations from level 1 of the computational tree.

This choice of the observation granularity is particularly attractive due to its accordance with the traditional definitions of Turing machines (see definitions (7.1) and (7.5)). A more fine granularity of observations allowing us to follow internal operations of the machines can be also chosen but is not so convenient. In fact, such an accuracy would mix internal operations of the machines with operations of the algorithm that is executed. A coarser granularity could be considered, as well. For instance, we could define as a computational step two consecutive applications of the transition function of the machine. However, in this case we do not observe all the nodes of the computational tree. As a consequence, we could miss some results of the computation as the machine could reach a final configuration before completing an observed computational step and we are not able to observe when and on which configuration the machine stopped. Then, fixed the chosen level of granularity the following result holds immediately.

Theorem 8. *(i) With the chosen level of granularity no more than ① computational steps of the machine $\mathcal{T}^{\mathcal{I}}$ can be observed in a sequence. (ii) In order to give possibility to observe at least one computational path of the computational tree of $\mathcal{T}^{\mathcal{I}\mathcal{N}}$*

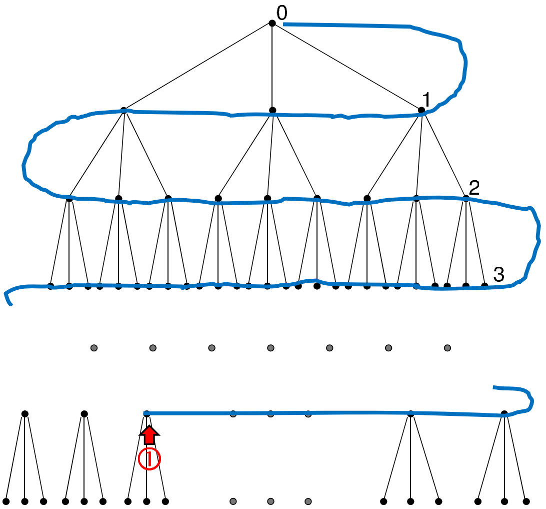


Fig. 7.2 The maximum number of computational steps of the machine $\mathcal{T}^{\mathcal{I}}$ that can be observed in a sequence

from the level 1 to the level k, the depth, $k \geq 1$, *of the computational tree cannot be larger than grossone, i.e.,* $k \leq$ ①.

Proof. Both results follow from the analysis made in Section 7.3.1 and Theorem 1. □

In Figure 7.2 the first result of Theorem 8 concerning the maximum number of computational steps of the machine $\mathcal{T}^{\mathcal{I}}$ that can be observed in a sequence is exemplified with reference to the computational tree of the machine introduced in Section 7.2.3.

Similarly, the second result of Theorem 8 concerning the depth of the computational tree of $\mathcal{T}^{\mathcal{IN}}$ is exemplified in Figure 7.3.

Corollary 4. *Suppose that d is the non-deterministic degree of* $\mathcal{T}^{\mathcal{IN}}$ *and S is the number of leaf nodes of the computational tree with a depth k representing the possible results of the computation of* $\mathcal{T}^{\mathcal{IN}}$. *Then it is not possible to observe all S possible results of the computation of* $\mathcal{T}^{\mathcal{IN}}$ *if the computational tree of* $\mathcal{T}^{\mathcal{IN}}$ *is complete and* $d^k >$①.

Proof. For the number of leaf nodes of the tree, S, of a generic non-deterministic Turing machine $\mathcal{T}^{\mathcal{IN}}$ the estimate $S \leq d^k$ holds. In particular, $S = d^k$ if the computational tree is complete, that is our case. On the other hand, it follows from Theorem 1 that any sequence of observations cannot have more than grossone elements. As a consequence, the same limitation holds for the sequence of observations of the leaf nodes of the computational tree. This means that we are not able to observe all the possible results of the computation of our non-deterministic Turing machine $\mathcal{T}^{\mathcal{IN}}$ if $d^k >$①. □

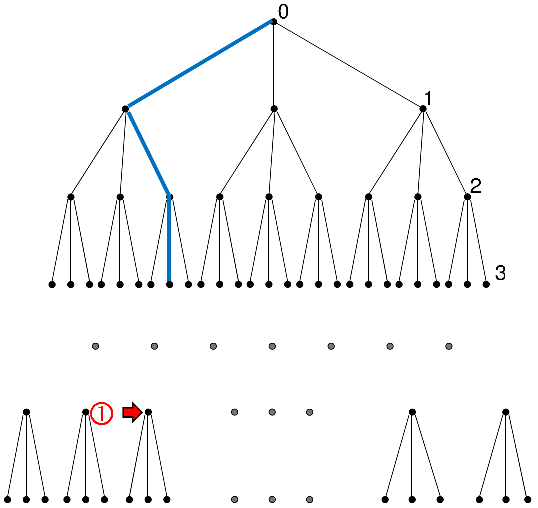


Fig. 7.3 An observable computational path of the machine $\mathcal{T}^{\mathcal{I}}$

In Figure 7.4 the result of Corollary 4 concerning the maximum number of computational results of the machine $\mathcal{T}^{\mathcal{I}}$ that can be observed in a sequence is exemplified with reference to the computational tree of the machine introduced in Section 7.2.3 .

Corollary 5. *Any sequence of observations of the nodes of the computational tree of a non-deterministic Turing machine $\mathcal{T}^{\mathcal{IN}}$ cannot observe all the nodes of the tree if the number of nodes N is such that $N >$①.*

Proof. The corollary follows from Theorems 1, 8, and Corollary 4. □

These results lead to the following theorem again under the same assumption about the chosen level of granularity of observations, i.e., the nodes of the computational tree of $\mathcal{T}^{\mathcal{IN}}$ representing configurations of the machine are observed.

Theorem 9. *Given a non-deterministic Turing machine $\mathcal{T}^{\mathcal{IN}}$ with a depth, k, of the computational tree and with a non-deterministic degree d such that*

$$\frac{d(kd^{k+1}-(k+1)d^{k}+1)}{(d-1)^{2}} \leqslant \text{①}, \tag{7.25}$$

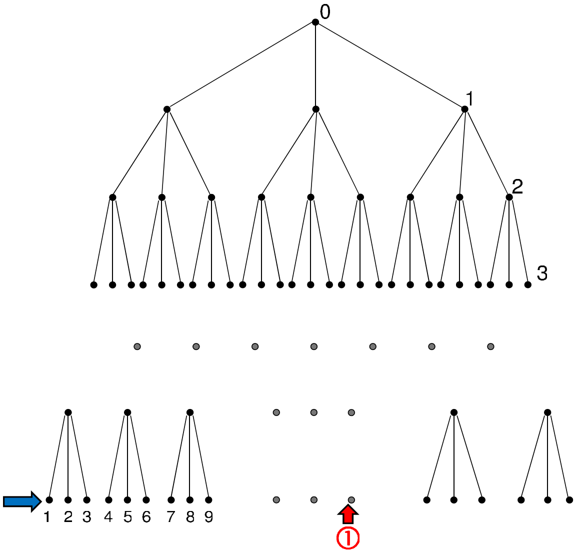


Fig. 7.4 Observable results of of the machine $\mathcal{T}^{\mathcal{I}}$

then there exists an equivalent deterministic Turing machine $\mathcal{T}^{\mathcal{I}}$ *which is able to simulate* $\mathcal{T}^{\mathcal{IN}}$ *and can be observed.*

Proof. For simulating $\mathcal{T}^{\mathcal{IN}}$, the deterministic machine $\mathcal{T}^{\mathcal{I}}$ executes a breadth-first visit of the computational tree of $\mathcal{T}^{\mathcal{IN}}$. In this computational tree, whose depth is $1 \leqslant k \leqslant ①$, each node has, by definition, a number of children c where $0 \leqslant c \leqslant d$. Let us suppose that the tree is complete, i.e., each node has $c = d$ children. In this case the tree has d^k leaf nodes and d^j computational paths of length j for each $1 \leqslant j \leqslant k$. Thus, for simulating all the possible computations of $\mathcal{T}^{\mathcal{IN}}$, i.e., for a complete visiting the computational tree of $\mathcal{T}^{\mathcal{IN}}$ and considering all the possible computational paths consisting of j computational steps for each $1 \leqslant j \leqslant k$, the deterministic machine $\mathcal{T}^{\mathcal{I}}$ will execute

$$K_{\mathcal{T}^{\mathcal{I}}} = \sum_{j=1}^{k} jd^j \tag{7.26}$$

steps (note that if the computational tree of $\mathcal{T}^{\mathcal{IN}}$ is not complete, $\mathcal{T}^{\mathcal{I}}$ will execute less than $K_{\mathcal{T}^{\mathcal{I}}}$). Due to Theorems 1 and 8, and Corollary 5, it follows that in order to prove the theorem it is sufficient to show that under conditions of the theorem it follows that

$$K_{\mathcal{T}^{\mathcal{I}}} \leqslant ①. \tag{7.27}$$

To do this let us use the well known formula

$$\sum_{j=0}^{k} d^j = \frac{d^{k+1}-1}{d-1}, \tag{7.28}$$

and derive both parts of (7.28) with respect to d. As the result we obtain

$$\sum_{j=1}^{k} jd^{j-1} = \frac{kd^{k+1}-(k+1)d^k+1}{(d-1)^2}. \tag{7.29}$$

Notice now that by using (7.26) it becomes possible to represent the number $K_{\mathcal{T}\mathcal{I}}$ as

$$K_{\mathcal{T}\mathcal{I}} = \sum_{j=1}^{k} jd^j = d\sum_{j=1}^{k} jd^{j-1}.$$

This representation together with (7.29) allow us to write

$$K_{\mathcal{T}\mathcal{I}} = \frac{d(kd^{k+1}-(k+1)d^k+1)}{(d-1)^2} \tag{7.30}$$

Due to assumption (7.25), it follows that (7.27) holds. This fact concludes the proof of the theorem. □

Corollary 6. *Suppose that the length of the input sequence of symbols of a nondeterministic Turing machine $\mathcal{T}^{\mathcal{I}\mathcal{N}}$ is equal to a number n and $\mathcal{T}^{\mathcal{I}\mathcal{N}}$ has a complete computational tree with the depth k such that $k = n^l$, i.e., polynomially depends on the length n. Then, if the values d, n, and l satisfy the following condition*

$$\frac{d(n^l d^{n^l+1}-(n^l+1)d^{n^l}+1)}{(d-1)^2} \leqslant \text{①}, \tag{7.31}$$

then: (i) there exists a deterministic Turing machine $\mathcal{T}^{\mathcal{I}}$ that can be observed and able to simulate $\mathcal{T}^{\mathcal{I}\mathcal{N}}$; (ii) the number, $K_{\mathcal{T}\mathcal{I}}$, of computational steps required to a deterministic Turing machine $\mathcal{T}^{\mathcal{I}}$ to simulate $\mathcal{T}^{\mathcal{I}\mathcal{N}}$ for reaching a final configuration exponentially depends on n.

Proof. The first assertion follows immediately from theorem 9. Let us prove the second assertion. Since the computational tree of $\mathcal{T}^{\mathcal{I}\mathcal{N}}$ is complete and has the depth k, the corresponding deterministic Turing machine $\mathcal{T}^{\mathcal{I}}$ for simulating $\mathcal{T}^{\mathcal{I}\mathcal{N}}$ will execute $K_{\mathcal{T}\mathcal{I}}$ steps where $K_{\mathcal{T}\mathcal{I}}$ is from (7.27). Since condition (7.31) is satisfied for $\mathcal{T}^{\mathcal{I}\mathcal{N}}$, we can substitute $k = n^l$ in (7.30). As the result of this substitution and (7.31) we obtain that

$$K_{\mathcal{T}\mathcal{I}} = \frac{d(n^l d^{n^l+1}-(n^l+1)d^{n^l}+1)}{(d-1)^2} \leqslant \text{①}, \tag{7.32}$$

i.e., the number of computational steps required to the deterministic Turing machine $\mathcal{T}^{\mathcal{I}}$ to simulate the non-deterministic Turing machine $\mathcal{T}^{\mathcal{IN}}$ for reaching a final configuration is $K_{\mathcal{T}^{\mathcal{I}}} \leqslant ①$ and this number exponentially depends on the length of the sequence of symbols provided as input to $\mathcal{T}^{\mathcal{IN}}$. □

Results described in this section show that the introduction of the new mathematical language including grossone allows us to perform a more subtle analysis with respect to traditional languages and to introduce in the process of this analysis the figure of the researcher using this language (more precisely, to emphasize the presence of the researcher in the process of the description of automatic computations). These results show that there exist limitations for simulating non-deterministic Turing machines by deterministic ones. These limitations can be viewed now thanks to the possibility (given because of the introduction of the new numeral ①) to observe final points of sequential processes for both cases of finite and infinite processes.

Theorems 8, 9, and their corollaries show that the discovered limitations and relations between deterministic and non-deterministic Turing machines have strong links with our mathematical abilities to describe automatic computations and to construct models for such descriptions. Again, as it was in the previous cases studied in this chapter, there is no contradiction with the traditional results because both approaches give results that are correct with respect to the languages used for the respective descriptions of automatic computations.

We conclude this section by the note that analogous results can be obtained for physical machines $\mathcal{T}^{\mathcal{P}}$ and $\mathcal{T}^{\mathcal{PN}}$, as well. In the case of ideal machines, the possibility of observations was limited by the mathematical languages. In the case of physical machines they are limited also by technical factors (we remind again the analogy: the possibilities of observations of physicists are limited by their instruments). In any given moment of time the maximal number of iterations, K_{max}, that can be executed by physical Turing machines can be determined. It depends on the speed of the fastest machine $\mathcal{T}^{\mathcal{P}}$ available at the current level of development of the humanity, on the capacity of its memory, on the time available for simulating a non-deterministic machine, on the numeral systems known to human beings, etc. Together with the development of technology this number will increase but it will remain finite and fixed in any given moment of time. As a result, theorems presented in this section can be re-written for $\mathcal{T}^{\mathcal{P}}$ and $\mathcal{T}^{\mathcal{PN}}$ by substituting grossone with K_{max} in them.

7.5 Concluding Remarks

Since the beginning of the last century, the fundamental nature of the concept of *automatic computations* attracted a great attention of mathematicians and computer scientists (see [4, 14–16, 23, 24, 27, 44]). The first studies had as their reference context the David Hilbert programme, and as their reference language that introduced by Georg Cantor [3]. These approaches lead to different mathematical models of computing machines (see [1, 6, 9]) that, surprisingly, were discovered to be equivalent (e.g., anything computable in the λ-calculus is computable by a Turing machine). Moreover, these results, and expecially those obtained by Alonzo Church, Alan Turing [4, 10, 44] and Kurt Gödel, gave fundamental contributions to demonstrate that David Hilbert programme, which was based on the idea that all of the Mathematics could be precisely axiomatized, cannot be realized.

In spite of this fact, the idea of finding an adequate set of axioms for one or another field of Mathematics continues to be among the most attractive goals for contemporary mathematicians. Usually, when it is necessary to define a concept or an object, logicians try to introduce a number of axioms describing the object in the absolutely best way. However, it is not clear how to reach this absoluteness; indeed, when we describe a mathematical object or a concept we are limited by the expressive capacity of the language we use to make this description. A richer language allows us to say more about the object and a weaker language – less. Thus, the continuous development of the mathematical (and not only mathematical) languages leads to a continuous necessity of a transcription and specification of axiomatic systems. Second, there is no guarantee that the chosen axiomatic system defines ‘sufficiently well’ the required concept and a continuous comparison with practice is required in order to check the goodness of the accepted set of axioms. However, there cannot be again any guarantee that the new version will be the last and definitive one. Finally, the third limitation already mentioned above has been discovered by Gödel in his two famous incompleteness theorems (see [10]).

Starting from these considerations, in the chapter, Single and Multi-tape Turing machines have been described and observed through the lens of the Grossone language and methodology . This new language, differently from the traditional one, makes it possible to distinguish among infinite sequences of different length so enabling a more accurate description of Single and Multi-tape Turing machines. The possibility to express the length of an infinite sequence explicitly gives the possibility to establish more accurate results regarding the equivalence of machines in comparison with the observations that can be done by using the traditional language.

It is worth noting that the traditional results and those presented in the chapter do not contradict one another. They are just written by using different mathematical languages having different accuracies. Both mathematical languages observe and describe the same objects – Turing machines – but with different accuracies. As a result, both traditional and new results are correct with respect to the mathematical languages used to express them and correspond to different accuracies of the observation. This fact is one of the manifestations of the relativity of mathematical results formulated by using different mathematical languages in the same way as the usage of a stronger lens in a microscope gives a possibility to distinguish more objects within an object that seems to be unique when viewed by a weaker lens.

Specifically, the Grossone language has allowed us to give the definition of *complete output* of a Turing machine, to establish when and how the output of a machine can be observed, and to establish a more accurate relationship between

Multi-tape and Single-tape Turing machines as well as between deterministic and non-deterministic ones. Future research efforts will be geared to apply the Grossone language and methodology to the description and observation of new and emerging computational paradigms.

References

1. Ausiello, G., D'Amore, F., Gambosi, G.: Linguaggi, modelli, complessit*à*, 2nd edn. Franco Angeli Editore, Milan (2006)
2. Benci, V., Di Nasso, M.: Numerosities of labeled sets: a new way of counting. Advances in Mathematics 173, 50–67 (2003)
3. Cantor, G.: Contributions to the founding of the theory of transfinite numbers. Dover Publications, New York (1955)
4. Church, A.: An unsolvable problem of elementary number theory. American Journal of Mathematics 58, 345–363 (1936)
5. Conway, J.H., Guy, R.K.: The Book of Numbers. Springer, New York (1996)
6. Barry Cooper, S.: Computability Theory. Chapman Hall/CRC (2003)
7. De Cosmis, S., De Leone, R.: The use of Grossone in mathematical programming and operations research. Applied Mathematics and Computation 218(16), 8029–8038 (2012)
8. D'Alotto, L.: Cellular automata using infinite computations. Applied Mathematics and Computation 218(16), 8077–8082 (2012)
9. Davis, M.: Computability & Unsolvability. Dover Publications, New York (1985)
10. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. Monatshefte für Mathematik und Physik 38, 173–198 (1931)
11. Gordon, P.: Numerical cognition without words: Evidence from Amazonia. Science 306, 496–499 (2004)
12. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages and Computation, 1st edn. Addison-Wesley, Reading (1979)
13. Iudin, D.I.: Ya.D. Sergeyev, and M. Hayakawa. Interpretation of percolation in terms of infinity computations. Applied Mathematics and Computation 218(16), 8099–8111 (2012)
14. Kleene, S.C.: Introduction to metamathematics. D. Van Nostrand, New York (1952)
15. Kolmogorov, A.N.: On the concept of algorithm. Uspekhi Mat. Nauk 8(4), 175–176 (1953)
16. Kolmogorov, A.N., Uspensky, V.A.: On the definition of algorithm. Uspekhi Mat. Nauk 13(4), 3–28 (1958)
17. Leibniz, G.W., Child, J.M.: The Early Mathematical Manuscripts of Leibniz. Dover Publications, New York (2005)
18. Levi-Civita, T.: Sui numeri transfiniti. Rend. Acc. Lincei, Series 5a 113, 7–91 (1898)
19. Lolli, G.: Metamathematical investigations on the theory of Grossone. To appear in Applied Mathematics and Computation
20. Lolli, G.: Infinitesimals and infinites in the history of mathematics: A brief survey. Applied Mathematics and Computation 218(16), 7979–7988 (2012)
21. Margenstern, M.: Using Grossone to count the number of elements of infinite sets and the connection with bijections. p-Adic Numbers, Ultrametric Analysis and Applications 3(3), 196–204 (2011)
22. Margenstern, M.: An application of Grossone to the study of a family of tilings of the hyperbolic plane. Applied Mathematics and Computation 218(16), 8005–8018 (2012)

23. Markov Jr., A.A., Nagorny, N.M.: Theory of Algorithms, 2nd edn. FAZIS, Moscow (1996)
24. Mayberry, J.P.: The Foundations of Mathematics in the Theory of Sets. Cambridge University Press, Cambridge (2001)
25. Newton, I.: Method of Fluxions. 1671
26. Pica, P., Lemer, C., Izard, V., Dehaene, S.: Exact and approximate arithmetic in an amazonian indigene group. Science 306, 499–503 (2004)
27. Post, E.: Finite combinatory processes – formulation 1. Journal of Symbolic Logic 1, 103–105 (1936)
28. Robinson, A.: Non-standard Analysis. Princeton Univ. Press, Princeton (1996)
29. Rosinger, E.E.: Microscopes and telescopes for theoretical physics: How rich locally and large globally is the geometric straight line? Prespacetime Journal 2(4), 601–624 (2011)
30. Sergeyev, Y.D.: Arithmetic of Infinity. Edizioni Orizzonti Meridionali, CS (2003)
31. Sergeyev, Y.D.: Blinking fractals and their quantitative analysis using infinite and infinitesimal numbers. Chaos, Solitons & Fractals 33(1), 50–75 (2007)
32. Sergeyev, Y.D.: A new applied approach for executing computations with infinite and infinitesimal quantities. Informatica 19(4), 567–596 (2008)
33. Sergeyev, Y.D.: Evaluating the exact infinitesimal values of area of Sierpinski's carpet and volume of Menger's sponge. Chaos, Solitons & Fractals 42(5), 3042–3046 (2009)
34. Sergeyev, Y.D.: Numerical computations and mathematical modelling with infinite and infinitesimal numbers. Journal of Applied Mathematics and Computing 29, 177–195 (2009)
35. Sergeyev, Y.D.: Numerical point of view on Calculus for functions assuming finite, infinite, and infinitesimal values over finite, infinite, and infinitesimal domains. Nonlinear Analysis Series A: Theory, Methods & Applications 71(12), e1688–e1707 (2009)
36. Sergeyev, Y.D.: Counting systems and the First Hilbert problem. Nonlinear Analysis Series A: Theory, Methods & Applications 72(3-4), 1701–1708 (2010)
37. Sergeyev, Y.D.: Lagrange Lecture: Methodology of numerical computations with infinities and infinitesimals. Rendiconti del Seminario Matematico dell'Università e del Politecnico di Torino 68(2), 95–113 (2010)
38. Sergeyev, Y.D.: Higher order numerical differentiation on the infinity computer. Optimization Letters 5(4), 575–585 (2011)
39. Sergeyev, Y.D.: On accuracy of mathematical languages used to deal with the Riemann zeta function and the Dirichlet eta function. p-Adic Numbers, Ultrametric Analysis and Applications 3(2), 129–148 (2011)
40. Sergeyev, Y.D.: Using blinking fractals for mathematical modelling of processes of growth in biological systems. Informatica 22(4), 559–576 (2011)
41. Sergeyev, Y.D.: Solving ordinary differential equations by working with infinitesimals numerically on the infinity computer. Applied Mathematics and Computation 219(22), 10668–10681 (2013)
42. Sergeyev, Y.D., Garro, A.: Observability of Turing machines: A refinement of the theory of computation. Informatica 21(3), 425–454 (2010)
43. Sergeyev, Y.D., Garro, A.: Single-tape and Multi-tape Turing Machines through the lens of the Grossone methodology. The Journal of Supercomputing 65(2), 645–663 (2013)
44. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proceedings of London Mathematical Society, Series 2 42, 230–265 (1936-1937)
45. Vita, M.C., De Bartolo, S., Fallico, C., Veltri, M.: Usage of infinitesimals in the Menger's Sponge model of porosity. Applied Mathematics and Computation 218(16), 8187–8196 (2012)

46. Wallis, J.: Arithmetica infinitorum (1656)
47. Zhigljavsky, A.A.: Computing sums of conditionally convergent and divergent series using the concept of Grossone. Applied Mathematics and Computation 218(16), 8064–8076 (2012)
48. Žilinskas, A.: On strong homogeneity of two global optimization algorithms based on statistical models of multimodal objective functions. Applied Mathematics and Computation 218(16), 8131–8136 (2012)

Chapter 8
On Parallel Array P Systems

Linqiang Pan and Gheorghe Păun

Abstract. We further investigate the parallel array P systems recently introduced by K.G. Subramanian, P. Isawasan, I. Venkat, and L. Pan. We first make explicit several classes of parallel array P systems (with one or more axioms, with total or maximal parallelism, with rules of various types). In this context, some results from the paper by Subramanian et al. mentioned above are improved. Several open problems are formulated.

8.1 Introduction

The generality/versatility of membrane computing is already a well known fact, the computing framework abstracted from the cell structure and functioning can cover a large variety of processes, dealing – in particular – with a large variety of objects processed in the compartments of membrane structures. The arrays (in general, two-dimensional and three-dimensional figures of various types) are one of the types of objects considered already since 2001, see [3]. A direct extension from string objects to two-dimensional arrays was introduced in [1] and then investigated in a series of papers.

A recent contribution to this research area is [6], where a natural counterpart of the array P systems from [1] is considered: parallel rewriting of arrays, instead of the sequential rewriting from [1]. Actually, the kind of parallelism investigated in [6] is that suggested by Lindenmayer systems: all nonterminals of an array should be rewritten in each step. A possible alternative, closer to the style of membrane

Linqiang Pan
Key Laboratory of Image Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
e-mail: lqpan@mail.hust.edu.cn

Gheorghe Păun
Institute of Mathematics of the Romanian Academy, P.O. Box 1-764,
014700 Bucureşti, Romania
e-mail: gpaun@us.es

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_8

computing, is to consider the maximal parallelism: a multiset of rules is used which is maximal among the multisets of applicable rules in a given moment.

In the present paper, we explicitly consider these two kinds of parallelism, and we prove that most of the results from [6] hold true for both kinds of parallelism, also improving those results (less membranes are used in some of them, while the powerful priority relation is avoided in other results).

Several questions remain open; several topics for further research are formulated.

8.2 Definitions and Notations

It is useful for the reader to be familiar with basic elements of membrane computing, e.g., from [4] (with up-dated information available at [7]), and of array grammars, but the notions we use will be recalled below. Actually, in what concerns arrays, we will usually use the pictorial representation, hence we need a minimal formalism (otherwise, cumbersome if rigorously formulated).

The arrays we consider consist of finitely many symbols from a specified alphabet V placed in the points (we call them *pixels*) of $\mathbf{Z}^2$ (the plane); the points of the plane which are not marked with elements of V are supposed to be marked with the *blank symbol* $\# \notin V$. Given an array W over V, $supp(W)$ denotes the set of points in $\mathbf{Z}^2$ marked with symbols in V. In order to specify an array, it is usual to specify the pixels of the support, by giving their coordinates, together with their associated symbols from V, but, as we said above, we will pictorially represent the arrays, indicating their non-blank pixels. These pictures should be interpreted as arrays placed in any position of the plane (congruent, possibly to be superposed by means of a translation).

By V^{*2} we denote the set of all two-dimensional arrays of finite support over V, including the empty array, denoted by λ. Any subset of V^{*2} is called an *array language*.

We handle the arrays by means of rewriting rules. An *array rewriting rule* (over an alphabet V) is written as a usual string rewriting rule, in the form $W_1 \to W_2$, where W_1, W_2 are *isotonic* arrays over V: W_1 and W_2 cover the same pixels, no matter whether they are marked with symbols in V or with #. When graphically representing an array, usually we ignore the blank pixels, but, when representing rewriting rules, the pixels marked with # are also explicitly shown. A rule as above is used to rewrite an array W in the natural way: a position in W is identified where W_1 can be superposed, with all pixels matching, whether or not they are marked with symbols in V or with #, and then those pixels are replaced with W_2 (the fact that W_1 and W_2 are isotonic ensures the fact that this replacement is possible). If the result is the array W', we write $W \Longrightarrow W'$. The reflexive and transitive closure of the relation $\Longrightarrow$ is denoted by $\Longrightarrow^*$.

Similar to string rewriting rules, the array productions can be classified according to their form. Remember that all rules we work with are isotonic (the shapes of the left hand side and the right hand side are identical, only the marking, by blank or non-blank symbols, differs). Note that so far we have not distinguished between terminal and nonterminal symbols, like in Chomsky grammars. Yet in the following, we will work, like in [6], in a Chomsky framework, with terminal and nonterminal symbols, and consider only two types of rules, *context-free* and *regular* ones. A context-free rule is an isotonic one with only one non-blank pixel in its left-hand side and only non-blank pixels in its right-hand side. A regular rule over the alphabets T and N, N being the nonterminal one, is a rule of one of the following forms: $A\,\# \to a\,B$, $\#\,A \to B\,a$, $\begin{array}{c}\#\\ A\end{array} \to \begin{array}{c}B\\ a\end{array}$, $\begin{array}{c}A\\ \#\end{array} \to \begin{array}{c}a\\ B\end{array}$, $A \to B$, $A \to a$, where $A,B \in N$ and $a \in T$.

Before introducing the array P systems, we recall a notion useful below: *two-dimensional right-linear grammars*.

Such a grammar (e.g., see [2]) is a construct $G = (V_h, V_v, V_i, T, S, R_h, R_v)$, where V_h, V_v, V_i are the horizontal, vertical, and intermediate alphabets of nonterminals, $V_i \subseteq V_v$, T is the terminal alphabet, $S \in V_h$ is the axiom, R_h is the finite set of horizontal rules, of the forms $X \to AY, X \to A$, for $X,Y \in V_h, A \in V_i$, and R_v is the finite set of vertical rules, of the forms $A \to aB, A \to a$, for $A,B \in V_v, a \in T$.

A derivation in G has two phases, a horizontal one, which uses rules from R_h, and a vertical one, which uses rules from R_v. The horizontal derivation is as usual in a string grammar. In the vertical phase, the rules are used in parallel, downwards, with the restriction that the terminal rules are used simultaneously for all vertical nonterminals. Thus, in the end, a rectangle is obtained, filled with symbols in T. The set of all rectangles generated in this way by G is denoted by $L(G)$ and the family of all languages of this form is denoted by $2RLG$.

Two array languages which will be used below are L_R, of all hollow rectangles with the edges marked with a (one element of this language is shown in Fig. 8.1), and L_S, of all hollow squares with the edges marked with a.

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

Fig. 8.1 A hollow rectangle in L_R

8.3 Parallel Array P Systems

We now pass to define the *parallel array P systems*. Such a device (of degree $m \geq 1$) is a construct

$$\Pi = (V, T, \#, \mu, F_1, \dots, F_m, R_1, \dots, R_m, i_o),$$

where: V is the total alphabet, $T \subseteq V$ is the terminal alphabet, # is the blank symbol, μ is a membrane structure with m membranes labeled in a one-to-one way with $1, 2, \dots, m$, $F_1, \dots, F_m$ are finite sets of arrays over V associated with the m regions of μ, $R_1,\ldots,R_m$ are finite sets of array rewriting rules over V associated with the m regions of μ; the rules have attached targets *here, out, in* (in general, *here* is omitted), hence they are of the form $W_1 \rightarrow W_2(tar)$; finally, i_o is the label of a membrane of μ specifying the output region.

In what follows, we only consider array P systems with regular (REG) and context-free (CF) rules – with the symbols in $V - T$ considered as nonterminals.

A computation in an array P system is defined in the same way as in a symbol object P system, with the following details. Every array from a compartment of the system must be rewritten by the rules in that compartment. The rewriting is parallel, with two types of parallelism:

(1) *the total one*, indicated by *allP*, which means that all nonterminal symbols from the array are rewritten, and

(2) *the maximal one*, indicated by *maxP*, which means that a multiset of rules is applied which is maximal, no further rule can be added to it. For any two rules used simultaneously, no pixel of their left hand sides may overlap (i.e., cover the same pixel of the rewritten array).

An important point appears here in what concerns the target indications of the rules: in each compartment, in a step we apply a multiset of rules with the same target indication. This is a very strong restriction, because it refers to all arrays from the compartment. In this paper, we work under this restriction. A weaker and somewhat more natural condition, which remains to be investigated (e.g., are the results proved below valid also in this case?), is to impose the restriction to use rules with the same target separately for each rewritten array (thus, separate arrays may be rewritten by rules with different targets). Of course, the two variants coincide for systems with only one axiom in the initial configuration.

The arrays obtained by an *allP* or a *maxP* rewriting are placed in the region indicated by the target associated with the used rules, in the way usual in membrane computing. It is important to stress the fact that all arrays from a given compartment travel together during a computation.

A computation is successful only if it halts, which means that it reaches a configuration where no rule can be applied to the current arrays. The result of a halting computation consists of the arrays composed only of symbols from T placed inthe region with label i_o in the halting configuration. The set of all such arrays computed (we also say *generated*) by a system Π is denoted by $A(\Pi)$.

Note that a computation which produces a terminal array (hence, no rule can be applied to it), but still can rewrite another array, is not halting; if the rewriting of one array continues forever, no matter how many terminal arrays were produced, then no result is obtained.

By $PAP_m(ax_k, \alpha, \beta)$ we denote the family of all array languages $A(\Pi)$ generated by systems Π as above, with at most m membranes, at most k initial arrays in its compartments ($\Sigma_{i=1}^{m} card(F_i) \leq k$), with rules of type $\alpha \in \{REG, CF\}$, working in the mode $\beta \in \{allP, maxP\}$. When m or k is not bounded, then it is replaced with $*$.

The following results were proved in [6] (*pri* indicates the use of a priority relation on the rules):

Lemma 1 (Lemma 3 in [6]). $2RLG \subseteq PAP_3(ax_1, CF, allP)$.

Lemma 2 (Lemma 4 in [6]). $PAP_3(ax_1, CF, allP) - 2RLG \neq \emptyset$.

Lemma 3 (Theorem 3 in [6]). $L_R \in PAP_2(ax_1, REG, allP, pri)$.

Lemma 4 (Theorem 4 in [6]). $L_S \in PAP_3(ax_1, REG, allP, pri)$.

In what follows, we will improve the first two lemmas in terms of the number of membranes. Moreover, we can avoid the priority relation in the last two lemmas at the price of using more than one axiom or using the *maxP* way of applying the rules.

8.4 Results

We now improve the first two lemmas stated above.

Theorem 1. $2RLG \subseteq PAP_2(ax_1, CF, \beta), \beta \in \{allP, maxP\}$.

Proof. Let $G = (V_h, V_v, V_i, T, S, T_h, R_v)$ be a two-dimensional right-linear grammar. We construct the array P system Π indicated in Figure 8.2. The horizontal derivation is done in membrane 2. When the horizontal phase is completed, the array is moved to the skin membrane, where the vertical phase is performed. After the use of the terminal rules, the array is moved back into the inner membrane, where this terminal array is obtained as a result of the computation in G. If the horizontal phase does not stop, then the computation continues forever, also by means of the rules $A \to A, A \in V_i$. Moreover, these rules guarantee that the *maxP* computations are *allP* computations, too. The equality $L(G) = A(\Pi)$ is clear.

Theorem 2. $PAP_2(ax_1, CF, \beta) - 2RLG \neq \emptyset,\ \beta \in \{allP, maxP\}$.

Proof. Let us consider the array P system from Figure 8.3. From the axiom AB, we generate a string X^nY^n (A goes to the left, simultaneously with B going to the right), then, like in a two-dimensional right-linear grammar, in the skin region we go vertically, marking the pixels with a in the columns of X and with b in the columns of Y. Moving back into the central membrane, a rectangle with the same number of columns marked with a and with b is obtained. The set of all these rectangles is not in the family $2RLG$ (the same example was already used in [6], too). The system works identically in both modes *allP* and *maxP*.

Removing the priority from the other two results from [6] can be done, but only by making use of the possibility of having two axioms in the initial configuration of the systems. One of them will generate the desired arrays, the other one will generate “twin” arrays, which control the computation of the former arrays.

Proposition 1. $L_R \in PAP_2(ax_2, REG, \beta),\ \beta \in \{allP, maxP\}$.

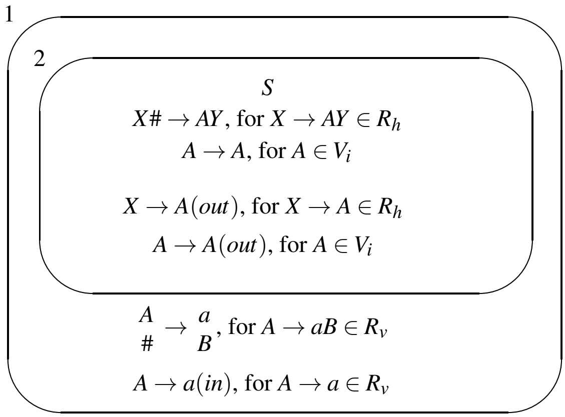

1
2
S
$X\# \to AY$, for $X \to AY \in R_h$
$A \to A$, for $A \in V_i$
$X \to A(out)$, for $X \to A \in R_h$
$A \to A(out)$, for $A \in V_i$
$\begin{array}{c}A\\ \#\end{array} \to \begin{array}{c}a\\ B\end{array}$, for $A \to aB \in R_v$
$A \to a(in)$, for $A \to a \in R_v$

Fig. 8.2 The array P system from the proof of Theorem 1

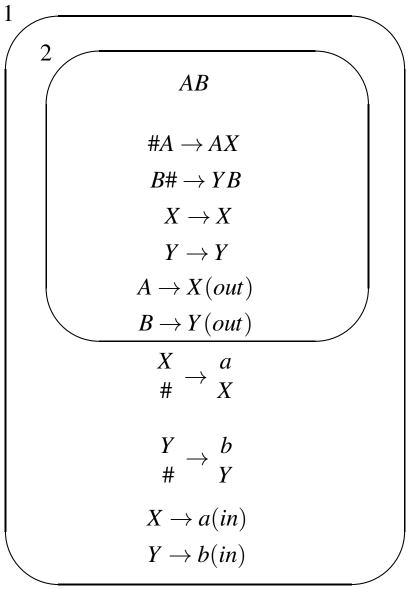

1
2
AB
$\#A \to AX$
$B\# \to YB$
$X \to X$
$Y \to Y$
$A \to X(out)$
$B \to Y(out)$
$\begin{array}{c}X\\ \#\end{array} \to \begin{array}{c}a\\ X\end{array}$
$\begin{array}{c}Y\\ \#\end{array} \to \begin{array}{c}b\\ Y\end{array}$
$X \to a(in)$
$Y \to b(in)$

Fig. 8.3 The array P system from the proof of Theorem 2

Proof. We consider the array P system from Figure 8.4. The axioms and the rules are written on two columns, in the left one those which lead to the desired arrays, in the right one those handling the "twin" (control) arrays. The rules are similar, with the ones in the right column dealing with primed nonterminal symbols.

The axiom in the left column is placed in the skin region, the one in the right column is placed in the inner membrane. In the first step, we move the inner axiom to the skin membrane, while removing the subscript 0 from all symbols A, B, A', B'. Now we start the generation of the hollow rectangles.

The bottom horizontal line of the arrays is generated in the skin membrane, by moving B and B' to the right. At a given step, we switch to moving vertically, with the arrays moved to membrane 2. We go upwards, synchronously, then the arrays are again moved to the skin membrane, with D and D' being the current nonterminals. Both D and D' go to the left and, at some moment, D is replaced with a (hence the "left" array is terminal (but we do not know whether the rectangle is completed). Moved again into membrane 2, the left array remains idle, while the right one can be rewritten – and, because of the parallelism, this must be done – if (and only if) D' has a non-marked pixel in its left. This happens if and only if the terminal array is not a complete hollow rectangle. The symbol D'' will go up without stopping, hence, the computation never halts. Thus, only the halting computations produce elements in the language L_R.

A similar result can be obtained for the language of hollow squares.

Proposition 2. $L_S \in PAP_2(ax_2, REG, \beta),\ \beta \in \{allP, maxP\}$.

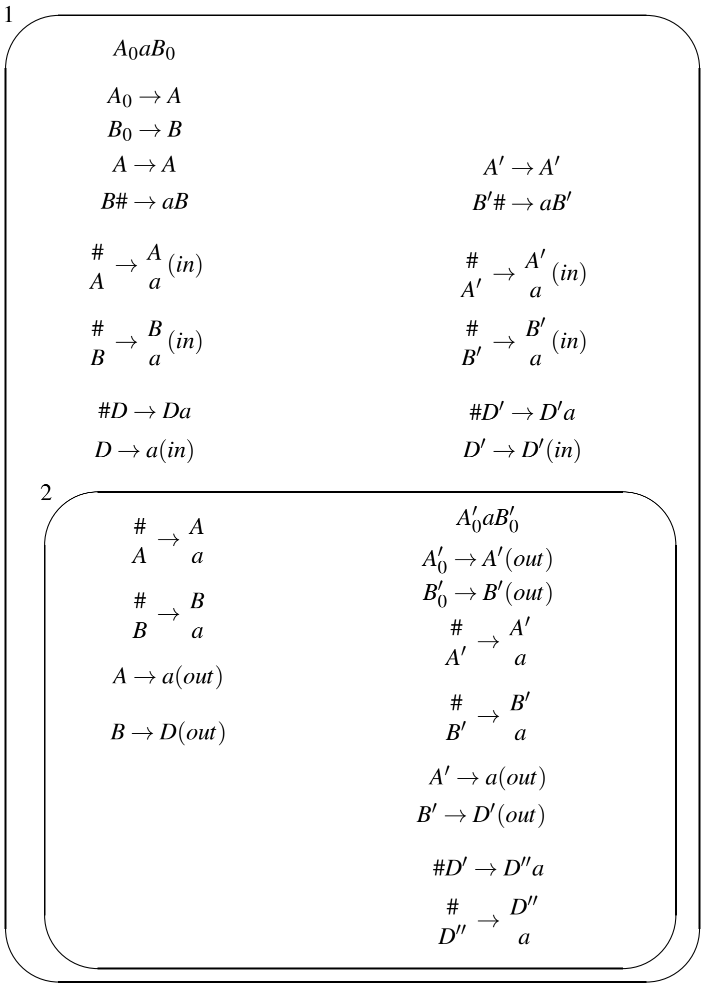
Proof. We consider the array P system from Figure 8.5, again with the axioms and the rules written on two columns, with the same significance as above. In the first step, we move the axiom from the skin region to the inner membrane, while also removing the subscript 0.

In the inner membrane we start constructing the square, from the left bottom corner, growing here the left and the bottom edges. At some moment, the two arrays are moved to the skin membrane, where the upper and the right edges are constructed. The work on the "left" array is terminated at the moment when the arrays are moved to the inner membrane. We check here whether or not the square is completed. It is not completed if and only if the nonterminal B'' has a non-marked pixel above it. If this is the case, the computation will continue forever, with B''' going to the right, hence the computation never halts. Checking all details remains as an exercise for the reader.

The *maxP* mode of using the rules is, intuitively speaking, able of "appearance checking", which is known to be a powerful feature of regulated grammars. This is confirmed also in our framework: we can generate the languages L_R, L_S by means of array P systems with only one axiom using the *maxP* mode.

Proposition 3. $L_S \in PAP_2(ax_1, REG, maxP)$.

Proof. We consider the array P system from Figure 8.6. In fact, this systems works like the previous one, but now generating only one array in which the checking for the square to have been completed is done directly in this array. The main difference is that the *maxP* mode allows us to let one symbol (B'') idle for one step. The rule $\bar{B}\# \to aB'''$ guarantees that the nonterminal symbol $\bar{B}$ cannot stay idle in the skin with the target *here* if A' does not change to $\bar{A}'$ at the same moment as B' changes to $\bar{B}'$.



1
A_0aB_0
$A_0 \to A$
$B_0 \to B$
$A \to A$
$B\# \to aB$
$\begin{array}{c}\#\\A\end{array} \to \begin{array}{c}A\\a\end{array}(in)$
$\begin{array}{c}\#\\B\end{array} \to \begin{array}{c}B\\a\end{array}(in)$
$\#D \to Da$
$D \to a(in)$
$A' \to A'$
$B'\# \to aB'$
$\begin{array}{c}\#\\A'\end{array} \to \begin{array}{c}A'\\a\end{array}(in)$
$\begin{array}{c}\#\\B'\end{array} \to \begin{array}{c}B'\\a\end{array}(in)$
$\#D' \to D'a$
$D' \to D'(in)$
2
$\begin{array}{c}\#\\A\end{array} \to \begin{array}{c}A\\a\end{array}$
$\begin{array}{c}\#\\B\end{array} \to \begin{array}{c}B\\a\end{array}$
$A \to a(out)$
$B \to D(out)$
$A'_0aB'_0$
$A'_0 \to A'(out)$
$B'_0 \to B'(out)$
$\begin{array}{c}\#\\A'\end{array} \to \begin{array}{c}A'\\a\end{array}$
$\begin{array}{c}\#\\B'\end{array} \to \begin{array}{c}B'\\a\end{array}$
$A' \to a(out)$
$B' \to D'(out)$
$\#D' \to D''a$
$\begin{array}{c}\#\\D''\end{array} \to \begin{array}{c}D''\\a\end{array}$

Fig. 8.4 The array P system from the proof of Proposition 1

It remains as an exercise for the reader to use similar ideas in order to prove that $L_R \in PAP_2(ax_1, REG, maxP\}$. On the other hand, we see no way to replace $maxP$ with $allP$ in these two results: $L_R \in PAP_2(ax_1, REG, maxP)$ and $L_S \in PAP_2(ax_1, REG, maxP)$.

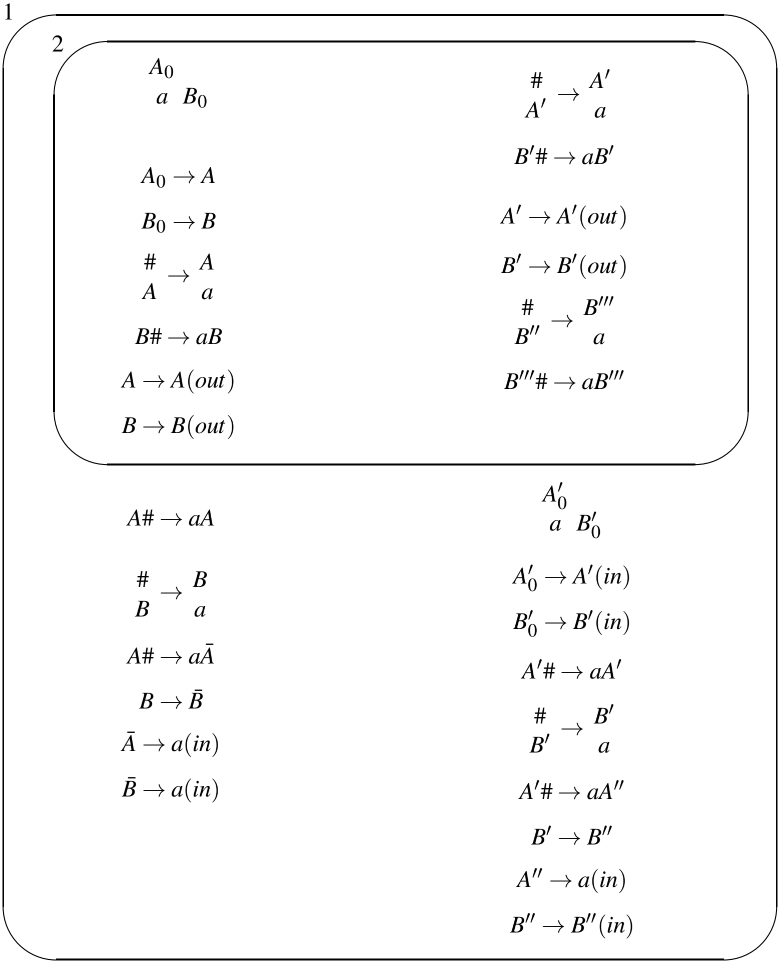

1
2
$\begin{matrix} A_0 & \\ a & B_0 \end{matrix}$
$A_0 \rightarrow A$
$B_0 \rightarrow B$
$\begin{matrix} \# \\ A \end{matrix} \rightarrow \begin{matrix} A \\ a \end{matrix}$
$B\# \rightarrow aB$
$A \rightarrow A(out)$
$B \rightarrow B(out)$
$\begin{matrix} \# \\ A' \end{matrix} \rightarrow \begin{matrix} A' \\ a \end{matrix}$
$B'\# \rightarrow aB'$
$A' \rightarrow A'(out)$
$B' \rightarrow B'(out)$
$\begin{matrix} \# \\ B'' \end{matrix} \rightarrow \begin{matrix} B''' \\ a \end{matrix}$
$B'''\# \rightarrow aB'''$
$A\# \rightarrow aA$
$\begin{matrix} \# \\ B \end{matrix} \rightarrow \begin{matrix} B \\ a \end{matrix}$
$A\# \rightarrow a\bar{A}$
$B \rightarrow \bar{B}$
$\bar{A} \rightarrow a(in)$
$\bar{B} \rightarrow a(in)$
$\begin{matrix} A'_0 & \\ a & B'_0 \end{matrix}$
$A'_0 \rightarrow A'(in)$
$B'_0 \rightarrow B'(in)$
$A'\# \rightarrow aA'$
$\begin{matrix} \# \\ B' \end{matrix} \rightarrow \begin{matrix} B' \\ a \end{matrix}$
$A'\# \rightarrow aA''$
$B' \rightarrow B''$
$A'' \rightarrow a(in)$
$B'' \rightarrow B''(in)$

Fig. 8.5 The array P system from the proof of Proposition 2

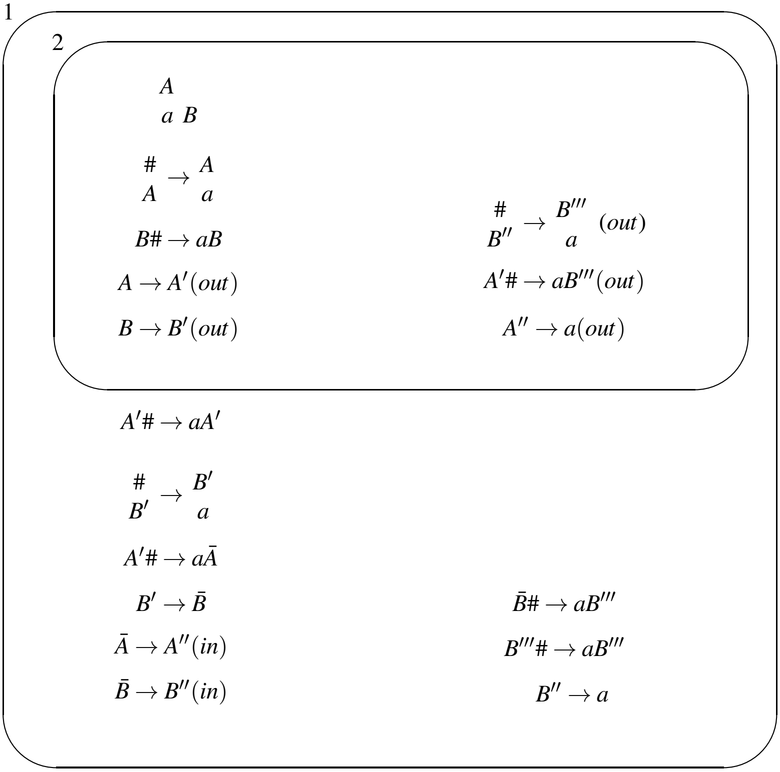

1
2
$\begin{matrix}A\\a\ B\end{matrix}$
$\begin{matrix}\#\\A\end{matrix} \to \begin{matrix}A\\a\end{matrix}$
$B\# \to aB$
$A \to A'(out)$
$B \to B'(out)$
$\begin{matrix}\#\\B''\end{matrix} \to \begin{matrix}B'''\\a\end{matrix}\ (out)$
$A'\# \to aB'''(out)$
$A'' \to a(out)$
$A'\# \to aA'$
$\begin{matrix}\#\\B'\end{matrix} \to \begin{matrix}B'\\a\end{matrix}$
$A'\# \to a\bar{A}$
$B' \to \bar{B}$
$\bar{A} \to A''(in)$
$\bar{B} \to B''(in)$
$\bar{B}\# \to aB'''$
$B'''\# \to aB'''$
$B'' \to a$

Fig. 8.6 The array P system from the proof of Proposition 3

8.5 Concluding Remarks

The goal of this note was, on the one hand, to make explicit the features involved in an array P system, especially, the number of axioms and the two types of parallelism. We made use of them in order to improve some of the results shown in [6], and, moreover, to replace the powerful ingredient priority relation by using more than one axiom or the more powerful variant of parallelism *maxP*. Further research efforts are necessary to clarify the computing power of parallel array P systems. For instance, we have the families $PAP_m(ax_k,\alpha,\beta)$, for $m \geq 1, k = 1$, $\alpha \in \{CF, REG\}, \beta \in \{allP, maxP\}$. What are the relations between them? Do the parameters m and k induce infinite hierarchies? Besides CF and REG, one can also consider variants of context-free array rules, for example erasing in contrast to non-erasing ones or allowing the right-hand side to contain blank pixels.

A natural question is whether or not parallel array P systems are computationally complete as the sequential ones considered in [1] are.

Rather natural is the possibility, already mentioned, to impose the use of rules with the same target for each array, separately, thus making possible that two arrays from a given region can go to different places after rewriting. A related issue is to consider other ways to control the communication, such as the t-mode from the grammar systems area, already investigated, for the sequential case, for array P systems in [5].

Finally, we mention the problem of considering Lindenmayer-like array P systems, either pure (without nonterminals) or extended (with all symbols to be rewritten, but accepting only terminal arrays).

Acknowledgements. The work of the first author was supported by the National Natural Science Foundation of China (61033003, 91130034 and 61320106005). Many thanks to Rudi Freund for carefully reading the initial version of the paper and for improving Proposition 3.

References

1. Ceterchi, R., Mutyam, M., Păun, Gh., Subramanian, K.G.: Array-rewriting P systems. Natural Computing 2, 229–249 (2003)
2. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 215–267. Springer, Berlin (1997)
3. Krishna, S.N., Krithivasan, K., Rama, R.: P systems with picture objects. Acta Cybernetica 15(1), 53–74 (2001)
4. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
5. Subramanian, K.G., Ali, R.M., Nagar, A.K., Margenstern, M.: Array P systems and t communication. Fundamenta Informaticae 91(1), 145–159 (2009)
6. Subramanian, K.G., Isawasan, P., Venkat, I., Pan, L.: Parallel array-rewriting P systems. Romanian Journal of Information Science and Technology (to appear)
7. The P Systems Website, `http://ppage.psystems.eu`.

Chapter 9
Small P Systems Defining Non-semilinear Sets

Artiom Alhazov and Rudolf Freund

Abstract. We present a number of tiny P systems generating or accepting non-semilinear sets of (vectors of) natural numbers with very small numbers of rules, even for 1, 2, 3, 4, and 5 rules, depending on the particular model and the additional features used in these systems. Among the models we consider are P systems with target agreement and target selection, P systems with promoters and inhibitors, catalytic and purely catalytic P systems with normal catalysts or bi-stable catalysts. We then improve the results for catalytic (purely catalytic) P systems: 14 rules for generating a non-semilinear vector set and 29 rules for generating a non-semilinear number set are sufficient when allowing only the minimal number of two (three) catalysts; only 23 rules are needed if we allow more catalysts, i.e., five (seven) catalysts. Moreover, we introduce the new concept of toxic objects, objects that must not stay idle as otherwise the computation is abandoned without yielding a result. P systems of various kinds using toxic objects allow for smaller descriptional complexity, especially for smaller numbers of rules, as trap rules can be avoided.

9.1 Introduction

Membrane systems were introduced by Gheorghe Păun in [13] (and therefore called *P systems* since then). Motivated by the biological functioning of molecules in cells, from a mathematical point of view a P system can be viewed as a parallel multiset

Artiom Alhazov
Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova
Academiei 5, Chişinău, MD-2028, Moldova
e-mail: artiom@math.md

Rudolf Freund
Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
e-mail: rudi@emcc.at

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_9

rewriting system. When using non-cooperative rules without any additional control, it has the behaviour of an $E0L$ system; yet when only taking the results when the system halts means that the objects evolve in a context-free manner, generating a set in $PsCF$, which is known (by Parikh's theorem) to coincide with $PsREG$, i.e., with the family of semilinear sets. In the accepting setup, P systems using non-cooperative rules without any additional control are even weaker, accepting all multisets over some subalphabet, or nothing, see [1].

In this paper we therefore are interested in different variants of P systems able to generate or accept non-semilinear sets of natural numbers or at least sets of vectors of natural numbers, yet with as small ingredients as possible. We assume that in "non-trivial" computations there has to be some form of interaction between the elements of the system (each element only containing finite information). Our main focus is on the descriptional complexity of these P systems, i.e., on how small the total number of rules may be, depending on the specific features of the particular models of P systems we look at. As most of the models of P systems can be shown to be computationally complete based on a simulation of the actions of a register machine, even simple examples often turn out to be somehow more complicated than the general proof.

We start with the strongest models, at least seen with respect to the number of rules needed to accept or generate a non-semilinear set. We conclude the list of results by P systems with two catalysts, thereby improving the result established in [19]. Ideas how to find good examples of catalytic P systems generating a non-semilinear set of numbers were discussed intensively during the Fourteenth International Conference on Membrane Computing (CMC 2013) in Chişinău, especially by Petr Sosík and Rudolf Freund, based on a draft by Petr Sosík, with most of them finally being included in his paper [19]. Some new observations just found recently allowed us to reduce the number of rules again in a considerable way, see Section 9.6.5.

In the following sections of this paper, we first provide a lot of examples for P systems generating or accepting a non-semilinear set of natural numbers or of vectors of natural numbers with only a few rules, especially for the maximally parallel derivation mode. Then we introduce the new concept of *toxic objects* which allows us to "kill" a computation branch if we cannot find a multiset of rules covering all occurrences of toxic objects which then somehow become "lethal" by killing such a computation. For all the proof techniques using a trap symbol # to "kill" a computation by introducing the trap symbol # with a non-cooperative rule $a \to \#$, the concept of toxic objects allows us to save most of the trap rules or even all of them, thus improving the descriptional complexity of the underlying P systems.

The rest of the paper is organized as follows: We first recall the basic definitions from formal language theory as well as the definitions for the most important variants of P systems for which we will give examples generating or accepting non-semilinear sets of (vectors of) natural numbers with only a few rules. Then we present examples with a small number of rules for variants of accepting P systems and then for variants of generating P systems, ending up with the catalytic and purely catalytic P systems, for which we improve previous results established in [19]. For a lot of variants, the concept of toxic objects allows us to give examples with a smaller number of rules than that we obtained with the original variant. We conclude the paper with a table summarizing our examples.

9.2 Definitions

In this section we first recall the basic notions from formal language theory needed in this paper and then the definitions of the basic variants of P systems considered in the following sections. For more details in formal language theory we refer the reader to the standard monographs and textbooks as [17] and for the area of regulated rewriting to [7]. All the main definitions and results for P systems can be found in [14] and [15]; only specific notations and models not yet to be found there will be explained in more detail in this paper, especially the new idea of *"toxic objects"*, which will be explained and studied in Section 9.6. For actual informations and new developments in the area of membrane computing we refer to the P systems webpage [20].

9.2.1 Prerequisites

The set of non-negative integers (*natural numbers*) is denoted by $\mathbb{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the elements of V^* are called strings, and the *empty string* is denoted by λ; $V^* \setminus \{\lambda\}$ is denoted by V^+. Let $\{a_1,\cdots,a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in a string x is denoted by $|x|_{a_i}$; the *Parikh vector* associated with x with respect to $a_1,\cdots,a_n$ is $\left(|x|_{a_1},\cdots,|x|_{a_n}\right)$. The *Parikh image* of a language L over $\{a_1,\cdots,a_n\}$ is the set of all Parikh vectors of strings in L, and we denote it by $Ps(L)$. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$; for families of languages over a one-letter alphabet, the corresponding sets of non-negative integers are denoted by NFL; for an alphabet V containing exactly d objects, the corresponding sets of Parikh vectors with d components is denoted by N^dFL, i.e., we replace Ps by N^d.

A (finite) *multiset* over the (finite) alphabet V, $V = \{a_1,\cdots,a_n\}$, is a mapping $f : V \longrightarrow \mathbb{N}$ and represented by $\langle f(a_1),a_1\rangle\cdots\langle f(a_n),a_n\rangle$ or by any string x the Parikh vector of which with respect to $a_1,\cdots,a_n$ is $(f(a_1),\cdots,f(a_n))$. In the following we will not distinguish between a vector $(m_1,\cdots,m_n)$, its representation by a multiset $\langle m_1,a_1\rangle\cdots\langle m_n,a_n\rangle$ or its representation by a string x having the Parikh vector $\left(|x|_{a_1},\cdots,|x|_{a_n}\right) = (m_1,\cdots,m_n)$. Fixing the sequence of symbols $a_1,\cdots,a_n$ in the alphabet V in advance, the representation of the multiset $\langle m_1,a_1\rangle\cdots\langle m_n,a_n\rangle$ by the string $a_1^{m_1}\cdots a_n^{m_n}$ is unique. The family of regular, context-free, and recursively enumerable string languages is denoted by REG, CF, and RE, respectively.

9.2.1.1 $ET0L$ Systems

An $ET0L$ system is a construct $G = (V,T,w,P_1,\cdots,P_m)$ where $m \geq 1$, V is an alphabet, $T \subseteq V$ is the terminal alphabet, $w \in V^*$ is the *axiom*, and the P_i, $1 \leq i \leq m$, are finite sets (*tables*) of non-cooperative rules over V. In a derivation step in G, all the symbols present in the current sentential form are rewritten using one table. The language generated by G, denoted by $L(G)$, consists of all terminal strings $w \in T^*$ which can be generated by a derivation in G starting from the axiom w. The family of languages generated by $ET0L$ systems and by $ET0L$ systems with at most k tables is denoted by $ET0L$ and ET_k0L, respectively.

Remark 1. According to Theorem 1.3 in chapter V of [16], for each language $L \in ET0L$ there exists an $ET0L$ system with only two tables $G' = (V',T,w,P_1',P_2')$ such that $L(G) = L$, i.e., we have $ET0L = ET_k0L$ for all $k \geq 2$. The main idea of the construction is the following: Given an $ET0L$ system $G = (V,T,w,P_1,\cdots,P_n)$ with $n > 2$, we use additional symbols $[a,i]$ for each $a \in V$ and $1 \leq i \leq n$. Then we define

$$\begin{aligned}
V' &= V \cup \{[a,i] \mid a \in V, 1 \leq i \leq n\},\\
P_1' &= \{a \to [a,1], [a,n] \to a \mid a \in V\} \cup \{[a,i] \to [a,i+1] \mid a \in V, 1 \leq i < n\},\\
P_2' &= \{[a,i] \to v \mid a \to v \in P_i, 1 \leq i \leq n\} \cup \{a \to a \mid a \in V\}.
\end{aligned}$$

The component P_1' “colors” the symbols $a \in V$ as $[a,i]$ for all of them being available in P_2' for a simulation of table P_i. After the use of P_2', we again have to use P_1' several times to get all symbols a colored as $[a,j]$ for all of them being available in P_2' for a simulation of a table P_j etc.

9.2.1.2 Register Machines

A *register machine* is a tuple $M = (m,B,l_0,l_h,P)$, where m is the number of registers, P is the set of instructions bijectively labeled by elements of B, $l_0 \in B$ is the initial label, and $l_h \in B$ is the final label. The instructions of M can be of the following forms:

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  Increase the value of register j by one, and non-deterministically jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : (SUB(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$.
  If the value of register j is zero then jump to instruction l_3, otherwise decrease the value of register j by one and jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stop the execution of the register machine.

A *configuration* of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed. Computations start by executing the first instruction of P (labeled with l_0), and terminate with reaching the $HALT$-instruction.

Register machines provide a simple universal computational model, for example, see [12]. In the following, we shall call a specific model of P systems *computationally complete* or *universal* if and only if for any register machine M we can effectively construct an equivalent P system Π of that type simulating M and yielding the same results.

9.2.1.3 Non-semilinear Sets of Numbers and Vectors of Numbers

In most of the examples described in the following sections, we will use (variants) of the set of natural numbers

$$\{2^n \mid n \geq 0\} = N\left(\left\{a^{2^n} \mid n \geq 0\right\}\right)$$

and the set of (two-dimensional) vectors of natural numbers

$$\{(n,m) \mid n \geq 1, n \leq m \leq 2^n\} = Ps(\{(a^n b^m) \mid n \geq 1, n \leq m \leq 2^n\}),$$

which both are known to not be semilinear.

9.2.2 P Systems

The ingredients of the basic variants of (cell-like) P systems are the membrane structure, the objects placed in the membrane regions, and the evolution rules. The *membrane structure* is a hierarchical arrangement of membranes. Each membrane defines a *region/compartment*, the space between the membrane and the immediately inner membranes; the outermost membrane is called the *skin membrane*, the region outside is the *environment*, also indicated by (the label) 0. Each membrane can be labeled, and the label (from a set *Lab*) will identify both the membrane and its region. The membrane structure can be represented by a rooted tree (with the label of a membrane in each node and the skin in the root), but also by an expression of correctly nested labeled parentheses. The *objects* (multisets) are placed in the compartments of the membrane structure and usually represented by strings, with the multiplicity of a symbol corresponding to the number of occurrences of that symbol in the string. The basic *evolution rules* are multiset rewriting rules of the form $u \to v$, where u is a multiset of objects from a given set O and $v = (b_1, tar_1) \ldots (b_k, tar_k)$ with $b_i \in O$ and $tar_i \in \{here, out, in\}$ or $tar_i \in \{here, out\} \cup \{in_j \mid j \in Lab\}$, $1 \leq i \leq k$. Using such a rule means "consuming" the objects of u and "producing" the objects $b_1, \ldots, b_k$ of v; the *target indications here*, *out*, and *in* mean that an object with the target *here* remains in the same region where the rule is applied, an object with the target *out* is sent out of the respective membrane (in this way, objects can also be sent to the environment, when the rule is applied in the skin region), while an object with the target *in* is sent to one of the immediately inner membranes, nondeterministically chosen, whereas with in_j this inner membrane can be specified directly. In general, we may omit the target indication *here*.

Yet there are a lot of other variants of rules we shall consider later; for example, if on the right-hand side of a rule we add the symbol δ, the surrounding membrane is dissolved whenever at least one such rule is applied, at the same moment all objects inside this membrane (the objects of this membrane region together with the whole inner membrane structure) are released to the surrounding membrane, and the rules assigned to the dissolved membrane region get lost. Another option is to add *promoters* $p_1,\dots,p_m \in O^+$ and *inhibitors* $q_1,\dots,q_n \in O^+$ to a rule and write $u \to v|_{p_1,\dots,p_m,\neg q_1,\dots,\neg q_n}$, which rule then is only applicable if the current contents of the membrane region includes each of the promoter multisets, but none of the inhibitor multisets; in most cases promoters and inhibitors are rather taken to be singleton objects than multisets. Further variants of P systems will be defined later.

Formally, a (cell-like) *P system* is a construct

$$\Pi = (O,\mu,w_1,\dots,w_m,R_1,\dots,R_m,f)$$

where O is the alphabet of *objects*, μ is the *membrane structure* (with m membranes), $w_1,\dots,w_m$ are multisets of objects present in the m regions of μ at the beginning of a computation, $R_1,\dots,R_m$ are finite sets of *evolution rules*, associated with the membrane regions of μ, and f is the label of the membrane region from which the outputs are taken/the inputs are put in ($f = 0$ indicates that the output/input is taken from the environment).

If a rule $u \to v$ has at least two objects in u, then it is called *cooperative*, otherwise it is called *non-cooperative*. In *catalytic P systems* we use non-cooperative as well as *catalytic rules* which are of the form $ca \to cv$, where c is a special object which never evolves and never passes through a membrane (both these restrictions can be relaxed), but it just assists object a to evolve to the multiset v. In a *purely catalytic P system* we only allow catalytic rules. For a catalytic as well as for a purely catalytic P system Π, in the description of Π we replace "O" by "O,C" in order to specify those objects from O which are the catalysts in the set C. As already explained above, cooperative and non-cooperative as well as catalytic rules can be extended by adding promoters and/or inhibitors, thus yielding rules of the form $u \to v|_{p_1,\dots,p_m,\neg q_1,\dots,\neg q_n}$.

All the rules defined so far can be used in different derivation modes: in the *sequential* mode (*sequ*), we apply exactly one rule in every derivation step; in the *asynchronous* mode (*asyn*), an arbitrary number of rules is applied in parallel; in the *maximally parallel* (*maxpar*) derivation mode, at any computation step of Π we choose a multiset of rules from the sets $R_1,\dots,R_m$ in a non-deterministic way such that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the membrane regions $1,\dots,m$.

The membranes and the objects present in the compartments of a system at a given time form a *configuration*; starting from a given *initial configuration* and using the rules as explained above, we get *transitions* among configurations; a sequence of transitions forms a *computation* (we often also say *derivation*). A computation is *halting* if and only if it reaches a configuration where no rule can be applied any more. With a halting computation we associate a *result generated* by this computation, in the form of the number of objects present in membrane f in the halting configuration. The set of multisets obtained as results of halting computations in Π working in the derivation mode $\delta \in \{sequ, asyn, maxpar\}$ is denoted by $mL_{gen,\delta}(\Pi)$, the set of natural numbers obtained by just counting the number of objects in the multisets of $mL_{gen,\delta}(\Pi)$ by $N_{gen,\delta}(\Pi)$, and the set of (Parikh) vectors obtained from the multisets in $mL_{gen,\delta}(\Pi)$ by $Ps_{gen,\delta}(\Pi)$.

Yet we may also start with some additional input multiset w_{input} over an *input alphabet* Σ in membrane f, i.e., in total we there have $w_f w_{input}$ in the initial configuration, and *accept* this input w_{input} if and only if there exists a halting computation with this input; the set of multisets accepted by halting computations in

$$\Pi = (O, \Sigma, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$$

working in the derivation mode δ is denoted by $mL_{acc,\delta}(\Pi)$, the corresponding sets of natural numbers and of (Parikh) vectors are denoted by $N_{acc,\delta}(\Pi)$ and $Ps_{acc,\delta}(\Pi)$, respectively.

The family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\gamma \in \{gen, acc\}$ computed by P systems with at most m membranes working in the derivation mode δ and with rules of type X is denoted by $Y_{\gamma,\delta}OP_m(X)$.

For example, it is well known (for example, see [13]) that for any $m \geq 1$, for the types of non-cooperative (*ncoo*) and cooperative (*coo*) rules we have

$$NREG = N_{gen,maxpar}OP_m(ncoo) \subset N_{gen,maxpar}OP_m(coo) = NRE.$$

For $\gamma \in \{gen, acc\}$ and $\delta \in \{sequ, asyn, maxpar\}$, the family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, computed by catalytic and purely catalytic P systems with at most m membranes and at most k catalysts is denoted by $Y_{\gamma,\delta}OP_m(cat_k)$ and $Y_{\gamma,\delta}OP_m(pcat_k)$, respectively; from [9] we know that, with the results being sent to the environment (which means taking $f = 0$), we have

$$Y_{gen,maxpar}OP_1(pcat_2) = Y_{gen,maxpar}OP_1(pcat_3) = YRE.$$

If we allow a catalyst c to switch between two different states c and $\bar{c}$ we call c a bi-stable catalyst; in that way we obtain *P systems with bi-stable catalysts*. For $\gamma \in \{gen, acc\}$ and $\delta \in \{sequ, asyn, maxpar\}$, the family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, computed by catalytic and purely catalytic P systems with bi-stable catalysts with at most m membranes and at most k catalysts is denoted by $Y_{\gamma,\delta}OP_m(2cat_k)$ and $Y_{\gamma,\delta}OP_m(p2cat_k)$, respectively. We note that in the generative case we do not count the catalysts in the output membrane region.

For all the variants of P systems of type X, we may consider to label all the rules in the sets $R_1, \ldots, R_m$ in a one-to-one manner by labels from a set H and to take a set W containing subsets of H. Then a *P system with label selection* is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, H, W, f)$$

where $\Pi' = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$ is a P system as defined above, H is a set of labels for the rules in the sets $R_1, \ldots, R_m$, and $W \subseteq 2^H$. In any transition step in Π we first select a set of labels $U \in W$ and then apply a non-empty multiset R of rules such that all the labels of these rules in R are in U (and in the case of maximal parallelism, the set R cannot be extended by any further rule with a label from U so that the obtained multiset of rules would still be applicable to the existing objects in the membrane regions $1,\dots,m$). The families of sets $Y(\Pi)$, $Y \in \{N,Ps\}$, computed by P systems with label selection with at most m membranes and rules of type X as well as $card(W) \leq k$ are denoted by $Y_{\gamma,\delta}OP_m(X,ls_k)$, for any $\gamma \in \{gen,acc\}$ and $\delta \in \{sequ,asyn,maxpar\}$.

For all variants of P systems using rules of some type X, we may consider systems containing only rules of the form $u \to v$ where $u \in O$ and $v = (b_1,tar)\dots(b_k,tar)$ with $b_i \in O$ and $tar \in \{here,out,in\}$ or $tar \in \{here,out\} \cup \{in_j \mid j \in H\}$, $1 \leq i \leq k$, i.e., in each rule there is only one target for all objects b_i; if catalytic rules are considered, then we request the rules to be of the form $ca \to c(b_1,tar)\dots(b_k,tar)$; in both cases, for a rule $u \to (b_1,tar)\dots(b_k,tar)$ we then write $u \to (b_1,\dots,b_k;tar)$.

A *P system with target selection* contains only these forms of rules; moreover, in each computation step, for each membrane region i we choose a multiset of rules from R_i having the same target indication tar; for different membrane regions these targets may be different; moreover, the total multiset obtained in this way must not be empty. The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N,Ps\}$, computed by P systems with target selection with at most m membranes and rules of type X are denoted by $Y_{\gamma,\delta}OP_m(X,ts)$, for any $\gamma \in \{gen,acc\}$ and $\delta \in \{sequ,asyn,maxpar\}$.

Remark 2. P systems with target selection were first defined in [10], but there the chosen multiset of rules for each R_i had to be non-empty if possible. In this paper, we only require the total multiset of rules, obtained by choosing multisets of rules in each R_i with the results going to a chosen target membrane, to be non-empty. Yet as in [10] we assume that when choosing the target *in* all objects are sent to just one selected inner membrane.

We may extend rules of the form $u \to (b_1,\dots,b_k;tar)$ to rules of the form $u \to (b_1,\dots,b_k;Tar)$ where Tar is a finite set of targets, thus obtaining *P systems with target agreement*. In each computation step, for each membrane we first choose a target tar and then a multiset of rules of the form $u \to (b_1,\dots,b_k;Tar)$ with $tar \in Tar$ – again, for different membranes these targets may be different. The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N,Ps\}$, computed by P systems with target agreement with at most m membranes and rules of type X are denoted by $Y_{\gamma,\delta}OP_m(X,ta)$, for any $\gamma \in \{gen,acc\}$ and $\delta \in \{sequ,asyn,maxpar\}$.

P systems with target agreement have the same computational power as P systems with target selection, as proved in the following theorem, yet they allow for a more compact description of rules as we will see in Subsection 9.4.1.

Theorem 1. *For all types of rules X, any $\gamma \in \{gen,acc\}$, any derivation mode $\delta \in \{sequ,asyn,maxpar\}$, any $Y \in \{N,Ps\}$, and any $m \in \mathbb{N}$, we have*

$$Y_{\gamma,\delta}OP_m(X,ta) = Y_{\gamma,\delta}OP_m(X,ts).$$

Proof. Given a P system with *target selection*

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$$

we can also interpret Π as a P system

$$\Pi' = (O, \mu, w_1, \ldots, w_m, R'_1, \ldots, R'_m, f)$$

with *target agreement* by replacing each rule $u \to (b_1, \ldots, b_k; tar)$ in any of the sets R_i, $1 \leq i \leq m$, by the corresponding rule $u \to (b_1, \ldots, b_k; \{tar\})$ in R'_i. Obviously, $Y_{\gamma,\delta}(\Pi) = Y_{\gamma,\delta}(\Pi')$.

On the other hand, given a P system

$$\Pi' = (O, \mu, w_1, \ldots, w_m, R'_1, \ldots, R'_m, f)$$

with *target agreement* we immediately get the corresponding P system with *target selection*

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$$

such that $Y_{\gamma,\delta}(\Pi) = Y_{\gamma,\delta}(\Pi')$: for each rule $u \to (b_1, \ldots, b_k; Tar) \in R'_i$ we take all the rules $u \to (b_1, \ldots, b_k; tar)$ with $tar \in Tar$ into R_i. □

Whereas for most of the other variants considered in this paper the so-called *flattening* procedure (for more details see [10]) allows for finding equivalent systems with only one membrane, for P systems with target selection or target agreement the membrane structure usually plays an essential rôle.

The basic idea of membrane systems with objects moving between membrane regions most nicely is captured by the model of symport/antiport P systems: in the inner membranes objects can neither be generated nor deleted, but only move from one membrane region to another one by passing through membranes; new objects can only be taken from the environment which provides an unlimited source for specific objects.

Formally, a *symport/antiport P system* is a construct

$$\Pi = (O, E, \mu, w_0, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$$

where O is the alphabet of objects, $E \subseteq O$ is the set of objects being available in the environment in an unbounded number, μ is the membrane structure (with m membranes), w_0 is the finite multiset of objects over $O \setminus E$ present in the environment at the beginning of a computation, $w_1, \ldots, w_m$ are the multisets of objects present in the m regions of μ at the beginning of a computation, $R_1, \ldots, R_m$ are finite sets of symport and/or antiport rules, associated with the membranes of μ, and f is the label of the membrane region from which the outputs are taken/the inputs are put in ($f = 0$ indicates that the output/input is taken from the environment). Every rule is of the form $(u, out; v, in)$ with $u, v \in O^*$ and $uv \neq \lambda$; if $u = \lambda$ or $v = \lambda$ then this rule is called a *symport rule*, otherwise it is called an *antiport rule*. The application of a rule $(u, out; v, in) \in R_i$ means sending out u from membrane region i and taking

v into it from the surrounding membrane region. For $(u, out; v, in)$, $\max\{|u|, |v|\}$ is called its *weight* and $|uv|$ is called its *size*; obviously, for symport rules weight and size are the same. The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\gamma \in \{gen, acc\}$, and $\delta \in \{sequ, asyn, maxpar\}$, computed by symport/antiport P systems with at most m membranes, symport rules with maximal weight r as well as antiport rules with maximal weight w and maximal size s are denoted by $Y_{\gamma,\delta}OP_m(sym_r, anti_{w,s})$.

As we will only give one very small example from the area of P systems with active membranes, we only define a *P system with active membranes and polarizations* as a construct

$$\Pi = (O, \mu, H, P, w_1, \ldots, w_m, R, f)$$

where O is the alphabet of objects, μ is the membrane structure consisting of m membranes, labeled (not necessarily in a one-to-one manner) with elements of H; P is a set of polarizations (mainly) taken to be 0, 1, and -1; $w_1, \ldots, w_m$ are strings over O, describing the multisets of objects in the m regions of μ; f is the label of an input/output membrane; R is a finite set of *developmental rules*, of the following forms:

(1)(*a*) $[\, a \to v \,]_h^e$,
for $h \in H$, $e \in P$, $a \in O$, $v \in O^*$
(*object evolution rules*; associated with membranes and depending on the label and the polarization of the membranes, but not directly involving the membranes, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);

(*b*) $a[\ \ ]_h^{e_1} \to [\, b \,]_h^{e_2}$,
for $h \in H$, $e_1, e_2 \in P$, $a, b \in O$
(*communication rules*; an object is sent in through the membrane, possibly modified by this process; also the polarization of the membrane may be modified, but not its label);

(*c*) $[\, a \ ]_h^{e_1} \to [\ \ ]_h^{e_2} b$,
for $h \in H$, $e_1, e_2 \in P$, $a, b \in O$
(*communication rules*; an object is sent out through the membrane, possibly modified by this process; also the polarization of the membrane may be modified, but not its label);

(*d*) $[\, a \ ]_h^e \to b$,
for $h \in H$, $e \in P$, $a, b \in O$
(*dissolving rules*; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(*e*) $[\, a \ ]_h^{e_1} \to [\, b \ ]_h^{e_2} [\, c \ ]_h^{e_3}$,
for $h \in H$, $e_1, e_2, e_3 \in P$, $a, b, c \in O$
(*division rules for elementary membranes*; in reaction with an object, the membrane is divided into two membranes with the same label, possibly of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects, and the remaining objects are duplicated).

In general, we use the maximally parallel derivation mode, i.e., all objects which can be used by a rule have to be used, but whereas the rules of type (a) are applied in parallel, the rules of types (b), (c), (d), and (e) are used sequentially in the sense that one membrane can be used by at most one rule of these types at a time. The families of sets $Y_\gamma(\Pi)$, $Y \in \{N, Ps\}$, $\gamma \in \{gen, acc\}$, computed by P systems with active membranes with at most m membranes, p polarizations, and rules of type X are denoted by $Y_\gamma OP_m^p(X)$.

Some further specific variants of P system will be explained directly in the subsections of the following sections.

For any of the families of (vectors of) natural numbers $Y_{\gamma,\delta}OP_m(X)$ we will add subscript k at the end to indicate that only systems with at most k rules are considered, i.e., we write $Y_{\gamma,\delta}OP_m(X)_k$. If any of the finite parameters like m and k is unbounded, we replace it by $*$.

9.3 Accepting P Systems

The constructions in this section accept the non-semilinear set of natural numbers $\{2^n \mid n \geq 0\} \cup \{0\}$, i.e., for each of the following P systems Π we have $mL_{acc,maxpar}(\Pi) = \{a^{2^n} \mid n \geq 0\} \cup \{\lambda\}$. They are all based on iterated halving of the number of input objects.

9.3.1 Using a Special Accepting and Halting Condition

In this subsection we step away from the standard halting condition and from the standard definition of accepting inputs, which allows us to show how much these definitions may influence the result of minimizing the total number of rules. In a more general way, we will follow the idea of stopping a derivation when specific objects remain idle in Subsection 9.6.2.

We consider the P system

$$\Pi_1 = (O = \{a\}, \Sigma = \{a\}, \mu = [\ ]_1, w_1 = \lambda, R_1 = \{aa \to a\}, f = 1),$$

which, taken with the usual halting and accepting conditions simply accepts everything, i.e., $N_{acc,maxpar}(\Pi_1) = \{a^n \mid n \geq 0\} = \{a\}^*$. We would like to mention that in the description of the P systems exhibited in the following we will use the extended description as for the P system Π_1 given above, not only writing

$$\Pi_1 = (\{a\}, \{a\}, [\ ]_1, \lambda, \{aa \to a\}, 1),$$

in order to make clear which meaning all the parameters in the description have.

In fact, the main idea of the P system Π_1 is to halve the number of objects in each derivation step, trying to reach 1. Hence, we now define a computation in Π_1 to immediately halt whenever some object a remains idle, but to only accept the input if the number of objects in the final configuration is not more than 1. Using the notation $fc1$ for this special variant of halting (we halt whenever no multiset of rules exists fully covering all objects in the underlying configuration) and acceptance (in the final configuration, at most 1 object is present), we get the desired result $mL_{acc-fc1,maxpar}(\Pi_1) = \{a^{2^n} \mid n \geq 0\} \cup \{\lambda\}$, i.e.,

$$\{2^n \mid n \geq 0\} \cup \{0\} \in N_{acc-fc1,maxpar}OP_1\,(coo)_1\,.$$

This accepting P system has only one cooperative rule, but uses a special non-standard halting condition and a special non-standard accepting condition. Moreover, it obviously is deterministic.

It is well-known that P systems with cooperative rules are computationally complete, even with standard halting and accepting definitions. The smallest known universal one has 23 rules, see [5].

For the further examples of this section, we now return to the original definitions of halting and acceptance.

9.3.2 P Systems with Input Alphabet

In many examples presented in this paper there is no need to distinguish between the total alphabet of symbols and a subset of input symbols, which we are going to do when considering the P system

$$\begin{aligned}\Pi_2 &= (O=\{a,s\},\Sigma=\{a\},\mu=[\ ]_1,w_1=\lambda,R_1,f=1) \text{ where}\\ R_1 &= \{aa \rightarrow a,\ a \rightarrow s,\ ss \rightarrow ss\}.\end{aligned}$$

In each derivation step, Π_2 halves (part of) the objects a, renaming the rest of objects a into s. If more than one s is produced, an infinite computation is forced due to the rule $ss \rightarrow ss$. The only way to produce not more than one s is to always have even multiplicity of objects a until it reaches the last one; hence, we conclude $mL_{acc,maxpar}(\Pi_2) = \{a^{2^n} \mid n \geq 0\} \cup \{\lambda\}$.

This accepting P system Π_2 has 3 cooperative rules, i.e., we have

$$\{2^n \mid n \geq 0\} \cup \{0\} \in N_{acc,maxpar}OP_1\,(coo)_3\,.$$

It is well-known that P systems with cooperative rules are computationally complete. The smallest known universal one has 23 rules, see [5].

9.3.3 P Systems with Symport/Antiport

The following symport/antiport P system accepting $\{a^{2^n} \mid n \geq 0\} \cup \{\lambda\}$ has only 3 rules and its construction is based on an idea from [11]:

$$\begin{aligned}\Pi_3 &= (O=\{a\},\Sigma=O,E=O,\mu=[\,[\,[\;]_3\,]_2\,]_1,\\ &\qquad w_1=\lambda,w_2=\lambda,w_3=aa,R_1,R_2,R_3,f=1),\\ R_1 &= \{(aa,out;a,in)\},\\ R_2 &= \{(a,in)\},\\ R_3 &= \{(aa,out;aa,in)\}.\end{aligned}$$

The construction works as follows: The only way to accept the input is to halve the number of objects in the skin membrane until this number reaches one (and therefore rule $(a,in) \in R_2$ need not be used more than once); if at least two objects come to region 2 from the skin membrane, i.e., if the rule $(a,in) \in R_2$ has been used at least twice, then the computation never halts, because rule $(aa,out;aa,in) \in R_3$ can be used forever.

This accepting P system has 3 symport/antiport rules of weight at most 2 and size at most 4; the size of the antiport rules may be decreased to 3 at a cost of one more rule, i.e., by taking the following symport/antiport P system:

$$\begin{aligned}\Pi_4 &= (O=\{a\},\Sigma=O,E=O,\mu=[\,[\,[\;]_3\,]_2\,]_1,\\ &\qquad w_1=\lambda,w_2=\lambda,w_3=a,R_1,R_2,R_3,f=1),\\ \mu &= [\,[\,[\;]_3\,]_2\,]_1,\\ R_1 &= \{(aa,out;a,in)\},\\ R_2 &= \{(a,in)\},\\ R_3 &= \{(aa,out;a,in),(a,out;aa,in)\}.\end{aligned}$$

In Π_4, an infinite loop is entered with the rules from R_3 as soon as the rule (a,in) from R_2 in total has been used at least twice.

Hence, in sum we have

$$\begin{aligned}\{2^n \mid n \geq 0\} \cup \{0\} \in\ & N_{acc,maxpar}OP_3\,(sym_1,anti_{2,4})_3\\ & \cap N_{acc,maxpar}OP_3\,(sym_1,anti_{2,3})_4\,.\end{aligned}$$

In general, symport/antiport P systems are well-known to be computationally complete with only one membrane. The smallest known number of rules is again 23, see [5].

9.4 Generating by Doubling in the Maximally Parallel Mode

This section contains models of P systems with "mass influence", where something can simultaneously affect (directly or indirectly) an unbounded number of copies of specific objects (e.g., target agreement, target selection, label selection, tables, membrane dissolution, promoters, inhibitors, active membranes with polarizations). In that way we obtain tiny P system using massive parallelism for repeated doubling, and using one of the effects mentioned above to halt.

9.4.1 P Systems with Target Agreement Or Target Selection

We first consider the case of *target agreement* which allows for a smaller descriptional complexity with only one rule:

$$\begin{aligned} \Pi_6 &= (O=\{a\},\mu=[\ [\ ]_2\ ]_1,w_1=a,w_2=\lambda,R_1,R_2=\emptyset,f=2) \text{ with} \\ R_1 &= \{a \rightarrow (aa,\{here,in\})\}. \end{aligned}$$

Π_6 doubles the number of objects each turn that the objects stay in the skin membrane choosing the target *here*. At some moment, all objects agree in the target destination *in* thus moving into the inner membrane where no rule can be applied any more. At any moment of the computation, all simultaneously produced objects must agree in the same destination, effectively choosing between continuing the doubling or halting.

If we resolve the rule $a \rightarrow (aa,\{here,in\})$ into its two corresponding rules $a \rightarrow (aa,here)$ and $a \rightarrow (aa,in)$, we immediately get the P system with *target selection*

$$\begin{aligned} \Pi_7 &= (O=\{a\},\mu=[\ [\ ]_2\ ]_1,w_1=a,w_2=\lambda,R_1,R_2=\emptyset,f=2) \text{ with} \\ R_1 &= \{a \rightarrow (aa,here),a \rightarrow (aa,in)\}. \end{aligned}$$

If in both cases we allow the output to be collected in the environment, we even get the following P systems with target agreement/selection having only one membrane:

$$\begin{aligned} \Pi_8 &= (O=\{a\},\mu=[\ ]_1,w_1=a,R_1,f=0) \text{ with} \\ R_1 &= \{a \rightarrow (aa,\{here,out\})\}. \end{aligned}$$

and

$$\begin{aligned} \Pi_9 &= (O=\{a\},\mu=[\ ]_1,w_1=a,R_1,f=0) \text{ with} \\ R_1 &= \{a \rightarrow (aa,here),a \rightarrow (aa,out)\}. \end{aligned}$$

In sum, we therefore infer

$$\{2^n \mid n \geq 1\} \in N_{gen,maxpar}OP_1\,(ncoo,ta)_1 \cap N_{gen,maxpar}OP_1\,(ncoo,ts)_2\,.$$

9.4.2 P systems with Label Selection

Instead of selecting different targets, in the P system with label selection

$$\begin{aligned} \Pi_{10} &= (O=\{a\},\mu=[\ ]_1,w_1=a,R_1,H=\{1,2\},W=\{\{1\},\{2\}\},f=0) \text{ with} \\ R_1 &= \{1: a \rightarrow (aa,here),2: a \rightarrow (aa,out)\}. \end{aligned}$$

we select different labels to be able to choose when to send out all objects from the skin membrane to the environment; hence, we have

$$\{2^n \mid n \geq 1\} \in N_{gen,maxpar}OP_1\,(ncoo,ls_2)_2\,.$$

9.4.3 P Systems with Tables

The following P system with tables of rules (*tabled P system*) needs only 2 rules and is closely related to the P system with target selection described in Subsection 9.4.1.

$$\begin{aligned} \Pi_{11} &= (O=\{a\},\mu=[\ ]_1,w_1=b,R_1,f=0) \text{ with} \\ R_1 &= \{T_1=\{a\to aa,here\},T_2=\{a\to a,out\}\}\,. \end{aligned}$$

Π_{11} doubles the multiplicity of objects a each step as long as using table T_1. At any time, if after $n \geq 0$ such steps the second table T_2 is chosen, all objects a are sent out to the environment and the computation halts, having generated a^{2^n}.

We also note that the second table (and thus the table feature) is not needed under a specific variant of halting called *unconditional halting* which resembles the L systems (Lindenmayer systems) mode of taking the result; P systems using non-cooperative rules and taking the results after each computation step (i.e., with unconditional halting) were considered in [6] and shown to characterize $PsET0L$.

In sum, we have obtained

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}OP_1\,(ncoo,table_2)_2 \cap N_{gen-u,maxpar}OP_1\,(ncoo)_1\,,$$

where $gen-u$ indicates that we take the results generated in the output membrane after every computation step (i.e., with unconditional halting) and $table_2$ indicates that we are using 2 tables.

It is rather obvious that the two concepts of using tables and using labels are pretty much the same – any possibility of choosing a specific table for each membrane region corresponds with choosing a specific subset of labels for the rules in the membrane regions; hence, we immediately infer the following results:

Theorem 2. *For any $k \geq 2$ and $n \geq 1$,*

$$\begin{aligned} PsET0L &= PsET_k0L \\ &= Ps_{gen,maxpar}OP_n\,(ncoo,table_k) \\ &= Ps_{gen,maxpar}OP_n\,(ncoo,ls_k)\,. \end{aligned}$$

Proof. We now first show that

$$PsET0L \subseteq Ps_{gen,maxpar}OP_1\,(ncoo,table_2)\,.$$

Given an $ET0L$ system $G=(V,T,w,P_1,\ldots,P_n)$, we adapt the construction given in Remark 1 for proving $PsET0L = PsET_20L$ in order to construct a P system with 2 tables and one membrane generating the elements of $Ps(L)$ in the environment. The main idea is to work on "colored" variants of the symbols from V, i.e., we define the renamings $h_i : V \to V \times \{i\}$ by $h_i(a) = [a,i]$ for all $a \in V$ and $1 \leq i \leq n$, as well as

$$\begin{aligned}
\Pi &= (O,\mu = [\ ]_1, w_1 = h_1(w), R_1 = \{T_1,T_2\}, f = 0),\\
O &= \left(\cup_{i=1}^{n} h_i(V)\right) \cup T \cup \{\#\},\\
T_1 &= \{h_i(a) \to (h_{i+1}(a), here) \mid a \in V, 1 \le i < n\}\\
&\quad \cup \{h_n(a) \to (a, out) \mid a \in T\}\\
&\quad \cup \{h_n(a) \to (\#, here) \mid a \in V \setminus T\} \cup \{\# \to (\#, here)\},\\
T_2 &= \{h_i(a) \to (h_1(w), here) \mid a \to w \in P_i, 1 \le i \le n\}.
\end{aligned}$$

By using the rules $h_i(a) \to (h_{i+1}(a), here)$ in table T_1, we can get to the right "color" i for simulating the application of table P_i in G by the corresponding rules $h_i(a) \to (h_1(w), here)$ from table T_2 in Π. Whenever we have reached "color" n we have the choice to either apply T_2 or the final rules $h_n(a) \to (a, out)$. If G has generated a terminal string w, then the corresponding multiset is sent out to the environment by applying the rules $h_n(a) \to (a, out)$ from table T_1 and the computation in Π halts, whereas if any of the rules $h_n(a) \to (\#, here) \in T_1$ for some $a \in V \setminus T$ has to be applied, we inevitably enter an infinite loop in Π. In sum, we conclude $Ps(L) = Ps_{gen,maxpar}(\Pi)$.

Obviously, for any $k \ge 1$ we have

$$Ps_{gen,maxpar}OP_1(ncoo, table_k) \subseteq Ps_{gen,maxpar}OP_1(ncoo, ls_k),$$

as given a P system with k tables

$$\Pi = (O, \mu = [\ ]_1, w_1, R_1 = \{T_1, \ldots, T_k\}, f = 0),$$

we immediately get the P system with label selection

$$\begin{aligned}
\Pi' &= (O, \mu = [\ ]_1, w_1, R_1, H, W, f = 0) \text{ where}\\
R_1 &= T_1' \cup \cdots \cup T_k',\\
T_i' &= \{i : p \mid p \in T_i\},\ 1 \le i \le k,\\
H &= \{1, \ldots, k\},\\
W &= \{\{1\}, \ldots, \{k\}\}.
\end{aligned}$$

Obviously, $Ps_{gen,maxpar}(\Pi') = Ps_{gen,maxpar}(\Pi)$.

By the usual "flattening" procedure, see [10], we can get an equivalent P system with label selection with only one membrane for any P system with label selection with n membranes; hence, it only remains to show that, for any $m \ge 1$,

$$Ps_{gen,maxpar}OP_1(ncoo, ls_m) \subseteq PsET0L.$$

For a given P system with label selection

$$\Pi = (O, \mu = [\ ]_1, w_1, R_1, H = \{1, \ldots, k\}, W = \{U_1, \ldots, U_m\}, f = 0)$$

we construct an equivalent $ET0L$ system G as follows:

$$\begin{aligned}
G &= \left(O \cup O' \cup \tilde{O} \cup \{\#\}, O, h(w), P_0, P_1, \dots, P_m\right),\\
P_i &= \{X' \to h'(b_1, tar_1) \dots h'(b_k, tar_k) \mid\\
&\qquad l : X \to (b_1, tar_1) \dots (b_k, tar_k) \in R_1, l \in U_i\}\\
&\quad \cup \{X' \to X' \mid X \in O, \neg \exists l \in U_i (l : X \to (b_1, tar_1) \dots (b_k, tar_k) \in R_1)\}\\
&\quad \cup \{\tilde{X} \to \tilde{X} \mid X \in O\} \cup \{X \to X \mid X \in O\} \cup \{\# \to \#\}, 1 \le i \le m,\\
P_0 &= \{\tilde{X} \to X \mid X \in O\} \cup \{X \to X \mid X \in O\} \cup \{\# \to \#\}\\
&\quad \cup \{X' \to \lambda \mid X \in O,\\
&\qquad \neg \exists l \in H (l : X \to (b_1, tar_1) \dots (b_k, tar_k) \in R_1 \wedge \exists j (l \in U_j))\}\\
&\quad \cup \{X' \to \# \mid X \in O,\\
&\qquad \exists l \in H (l : X \to (b_1, tar_1) \dots (b_k, tar_k) \in R_1 \wedge \exists j (l \in U_j))\}.
\end{aligned}$$

The renaming h is defined by $h : O \to O'$ with $h(X) = X'$ for $X \in O$; the morphism h' is defined by $h' : O \times \{here, out\} \to O' \cup \tilde{O}$ with $h'((X, here)) = X'$ and $h'((X, out)) = \tilde{X}$ for $X \in O$. With the table R_i we can simulate the application of rules with labels from U_i. In order to obtain the final multiset we have to apply P_0; objects $\tilde{X}$ for $X \in O$ represent objects X sent out to the environment, i.e., from the desired result, yet we may only take them as the result in G, too, if for all the remaining symbols Y represented as Y' there exists no rule with which the computation in Π could be continued, as then the nonterminal # is generated. Therefore we conclude $Ps(L(G)) = Ps_{gen,maxpar}(\Pi)$. □

It is easy to see that with *P systems with target agreement* and *P systems with target selection* we also get a characterization of $PsET0L$ when only working with non-cooperative rules:

Theorem 3. *For any $k \ge 2$,*

$$\begin{aligned}
Ps(ET0L) &= Ps(ET_k0L)\\
&= Ps_{gen,maxpar}OP_k(ncoo, ts)\\
&= Ps_{gen,maxpar}OP_k(ncoo, ta).
\end{aligned}$$

Proof. We first show that $Ps(ET0L) \subseteq Ps_{gen,maxpar}OP_2(ncoo, ts)$, i.e., we again start with an $ET0L$ system $G = (V, T, w, P_1, \dots, P_n)$ and adapt the construction given in Remark 1 in order to construct a P system Π with target selection and only 2 membranes generating the elements of $Ps(L)$ in the innermost membrane. We now define the renamings $h_i : V \to V \times \{i\}$ by $h_i(a) = [a, i]$ for all $a \in V$ and $0 \le i \le n$. Then we construct the following P system with target selection:

$$\begin{aligned}
\Pi &= (O, \mu = [\,[\;]_2\,]_1, w_1 = h_0(w), w_2 = \lambda, R_1, R_2, f = 2),\\
O &= \left(\cup_{i=0}^{n} h_i(V)\right) \cup T \cup \{\#\},\\
R_1 &= \{h_i(a) \to (h_{i+1}(a), tar) \mid a \in V, 0 \le i < n, tar \in \{here, in\}\}\\
&\quad \cup \{h_n(a) \to (a, in) \mid a \in T\}\\
&\quad \cup \{h_n(a) \to (\#, in) \mid a \in V \setminus T\},\\
R_2 &= \{h_i(a) \to (h_0(w), out) \mid a \to w \in P_i, 1 \le i \le n\} \cup \{\# \to \#\}.
\end{aligned}$$

In a non-deterministic way, with the targets *here* and *in* in the rules of R_1 we choose whether to continue with R_1 or to send all objects into the inner membrane, where one application of P_i is simulated. Only terminal multisets can be sent into the innermost membrane without causing the system Π to enter an infinite loop there with the rule # → #. Hence, we infer $Ps_{gen,maxpar}(\Pi) = Ps(L(G))$.

Due to the proof of Theorem 1, for all $n \geq 1$ we have

$$Ps_{gen,maxpar}OP_n(ncoo,ta) = Ps_{gen,maxpar}OP_n(coo,ts).$$

Finally, it remains to prove that

$$Ps_{gen,maxpar}OP_n(ncoo,ts) \subseteq Ps(ET0L) \text{ for all } n \geq 1.$$

We only sketch the main ideas of the proof: Given a P system with target selection Π, for each membrane i, in the simulating $ET0L$ system G we use the symbol $[a,i]$ instead of the symbol a; the tables of the $ET0L$ system then are constructed by taking every possible variant of choosing a target for each membrane and then simulating the rules of Π in G on the corresponding symbols $[a,i]$. At the end, with a special final table of rules T_f the contents of the final membrane f may be recovered as a result of a derivation in G, too, by projecting all objects $[a,f]$ to a and at the same time projecting all other symbols $[a,i]$ with $i \neq f$ to λ if there is no rule for a in the membrane region i of Π or else to the trap symbol #; an additional rule # → # guarantees that we enter an infinite loop if we tried to apply this final table T_f at the wrong moment when in Π the computation carried out so far had not yet halted. The technical details of the proof are left to the interested reader. □

9.4.4 P Systems with Membrane Dissolution

When using only non-cooperative rules we cannot obtain computational completeness with the additional feature of membrane dissolution; yet, in [8] an infinite hierarchy with respect to the number of membranes was established using membrane dissolution in a linear membrane structure. Now consider

$$\Pi_{12} = (O = \{a\}, \mu = [\ [\ ]_2\ ]_1, w_1 = \lambda, w_2 = a, R_1 = \emptyset, R_2, f = 1)$$

where $R_2 = \{a \rightarrow aa,\ a \rightarrow aa\delta\}$.

Π_{12} doubles the number of objects each turn when only using the rule $a \rightarrow aa$ until at some moment the inner membrane may be dissolved by at least for one a using the dissolution rule $a \rightarrow aa\delta$, thus stopping the computation.

This generative system has 2 non-cooperative rules and membrane dissolution and computes $mL_{gen,maxpar}(\Pi_{12}) = \{a^{2^n} \mid n \geq 0\}$, i.e.,

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}OP_2(ncoo,\delta)_2.$$

9.4.5 P Systems with Active Membranes Using Two Polarizations

We here use a very restricted variant of a P system with active membranes, i.e., we have only one membrane (the skin membrane) which may take one of the polarizations 0 and 1:

$$\begin{aligned}\Pi_{13} = (O = \{a\}, \mu = [\ ]_1, H = \{1\}, P = \{0,1\}, w_1 = a, R, f = 1) \text{ with}\\ R = \{[\, a \to aa\,]_1^0,\ [\, a\,]_1^0 \to [\ ]_1^1 a\}.\end{aligned}$$

Π_{13} doubles the multiplicity of objects a in each step using only the rule $[\, a \to aa\,]_1^0$ until at some moment, after $n \geq 0$ such doubling steps, (exactly) one object a is sent out by the rule $[\, a\,]_1^0 \to [\ ]_1^1 a$, changing polarization to 1; as with polarization 1 no rule can be applied any more in the skin membrane, the system halts with having generated a^{2^n-1}. In contrast to the previous example, now exactly one symbol a is used to stop the derivation undergoing the application of the rule $[\, a\,]_1^0 \to [\ ]_1^1 a$, whereas in the previous example the dissolution rule $a \to aa\delta$ could have been applied to an arbitrary non-zero number of copies of a.

Π_{13} is a P system with one active membrane, 2 polarizations and 2 rules, i.e., we have

$$\{2^n - 1 \mid n \geq 0\} \in N_{gen,maxpar}OP_1^2\,((\text{a}),(\text{c}))_2\,.$$

P systems with active membranes using non-cooperative rules are known to be computationally complete even with 2 polarizations and only rules of types (a) and (c), see [3]. They are computationally complete even without polarizations, but with creation of an unbounded number of membranes, allowing dissolution of membranes and rules for bringing objects inside a membrane, see [2] and [4].

9.4.6 P Systems with Inhibitors

The following P system uses atomic inhibitors in only two rules:

$$\Pi_{14} = (O = \{a,b\}, \mu = [\ ]_1, w_1 = a, R_1 = \{a \to aa|_{\neg b},\ a \to bb|_{\neg b}\}, f = 1).$$

Π_{14} doubles the multiplicity of objects a in each step using the rule $a \to aa|_{\neg b}$. At any time, some objects a may change to bb instead of aa when being evolved by the rule $a \to bb|_{\neg b}$, which causes the system to halt, having generated $a^{2m}b^{2n}$ for some m,n with $m+n = 2^k,\ k \geq 0,\ n > 0$, i.e., we obtain

$$mL_{gen,maxpar}(\Pi_{14}) = \{a^{2m}b^{2n} \mid m+n = 2^k,\ k \geq 0,\ n > 0\}$$

and therefore

$$\{(2m,2n) \mid m+n = 2^k,\ k \geq 0,\ n > 0\} \in Ps_{gen,maxpar}OP_1\,(ncoo, inh_1)_2\,;$$

inh_1 indicates that we only need atomic inhibitors, i.e., inhibitors of weight 1.

P systems with non-cooperative rules and atomic inhibitors characterize $PsET0L$, see [18].

9.4.7 P Systems with Promoters

The following P system uses one atomic promoter in only one rule:

$$\Pi_{15} = (O = \{a,b\}, \mu = [\ ]_1, w_1 = ab, R_1 = \{a \to aa|_b,\ b \to b,\ b \to \lambda\}, f = 1).$$

Π_{15} doubles the multiplicity of objects a in each step using the rule $a \to aa|_b$. At any time, object b, instead of being kept by the rule $b \to b$, may be erased by the rule $b \to \lambda$, thus causing the system to halt, having generated a^{2^n} for some $n \geq 1$. Π_{15} is a P system with 3 rules and using one atomic promoter; hence, we have

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}OP_1\,(ncoo, pro_1)_3$$

with pro_1 indicating that we only need atomic promoters, i.e., promoters of weight 1.

P systems with non-cooperative rules and atomic promoters characterize $PsET0L$, see [18].

9.5 Asynchronous and Sequential P systems

The constructions in this section do not rely on maximal parallelism as those given in the previous section. Hence, an action like doubling has to be done sequentially, and therefore it must be controlled in some specific way. The resulting systems elaborated in the following subsections fulfill their purpose equally well in the asynchronous, in the sequential mode, and even in the maximally parallel mode.

9.5.1 P Systems with One Bi-stable Catalyst

The purely catalytic P system with only one bi-stable catalyst (having the two states c and $\bar{c}$)

$$\begin{aligned} \Pi_{16} &= (O = \{a,b,b',c,\bar{c}\}, C = \{c,\bar{c}\}, \mu = [\ ]_1, w_1 = cab', R_1, f = 1) \text{ with} \\ R_1 &= \{cb' \to cbb,\ cb' \to \bar{c}bb,\ \bar{c}b \to \bar{c}b',\ \bar{c}a \to caa\} \end{aligned}$$

generates $mL_{gen,\delta}(\Pi_{16}) = \{a^n b^m \mid n \leq m \leq 2^n, n \geq 1, m \geq 2\}$, i.e., $Ps_{gen,\delta}(\Pi_{16})$ is a non-linear set of vectors of natural numbers, and this even holds for any derivation mode $\delta \in \{sequ, asyn, maxpar\}$.

With the catalyst being in state c, Π_{16} at most doubles the joint population of objects b and b', and may change the state to $\bar{c}$. With the catalyst being in state $\bar{c}$, some b may be renamed back to b', and the state is reset to c with adding one object a. The computation halts if and only if with c no object b' is present.

Π_{16} is a purely catalytic P system with only one bi-stable catalyst and 4 catalytic rules. Clearly, the construction works equally well in the maximally parallel, the asynchronous, and the sequential mode. In sum, we conclude

$$\{(n,m) \mid n \leq m \leq 2^n, n \geq 1, m \geq 2\} \in Ps_{gen,\delta}OP_1\,(p2cat_1)_4$$

for any $\delta \in \{sequ, asyn, maxpar\}$. In general, it is not difficult to see that purely catalytic P system with only one bi-stable catalyst have sequential behavior even under maximal parallelism. Hence, their computational power does not exceed that of partially blind register machines (i.e., we stay within *PsMAT*).

9.5.2 Asynchronous and Sequential P Systems with Promoters and Inhibitors

The P system using rules with promoters and inhibitors

$$\begin{aligned} \Pi_{17} &= (O=\{a,b,s,t\}, \mu=[\ ]_1, w_1=ta, R_1, f=1) \text{ with} \\ R_1 &= \{a \to bb|_s,\ s \to t|_{\neg a},\ b \to a|_t,\ t \to s|_{\neg b},\ t \to \lambda|_{\neg b}\} \end{aligned}$$

generates the well-known nonlinear set of natural numbers $\{2^n \mid n \geq 0\}$, as we have $mL_{gen,\delta}(\Pi_{17}) = \{a^{2^n} \mid n \geq 0\}$ for any $\delta \in \{sequ, asyn, maxpar\}$.

Π_{17} toggles between state s (converting objects a in pairs of objects b) and t (renaming b to a). The state may be changed when none of its corresponding reactants remains. State t may be erased instead of being reset to s, which leads to halting.

This generative P system Π_{17} has 5 non-cooperative rules with atomic promoters and inhibitors. We note that Π_{17} works equally well in the sequential, the asynchronous, and the maximally parallel mode. Hence, we infer

$$\{2^n \mid n \geq 0\} \in N_{gen,\delta}OP_1\,(ncoo, pro_1, inh_1)_5$$

for any $\delta \in \{sequ, asyn, maxpar\}$. Moreover, Π_{17} may take advantage of available parallelism. Recall that smaller solutions have been presented in the previous section for the maximally parallel derivation mode, even using either promoters or inhibitors only, but not both.

In general, sequential (and asynchronous) P systems with non-cooperative rules and promoters and inhibitors are computationally complete, e.g., see [1].

9.6 P Systems with Catalysts

P systems with catalysts were already considered in the originating papers for membrane systems, see [13]. In [9], two catalysts (three catalysts) were shown to be sufficient for getting computational completeness with catalytic (purely catalytic) P systems. Whether or not one two catalysts (respectively three catalysts) might already be enough to obtain computational completeness, is still one of the most challenging open problems in the area of P systems. We only know that purely catalytic P systems (working in the maximally parallel mode) with only one catalyst simply correspond with sequential P systems with only one membrane, hence, to multiset rewriting systems with context-free rules, and therefore can only generate linear sets.

Using additional control mechanisms as, for example, priorities or promoters/inhibitors, P systems with only one catalyst can be shown to be computationally complete, e.g., see Chapter 4 of [15]. On the other hand, additional features for the catalyst may be taken into account; for example, we may use bi-stable catalysts (catalysts switching between two different states) as already considered in Subsection 9.5.1.

The first system presented in this section continues the work using one bi-stable catalyst: allowing non-cooperative rules, a non-semilinear set of *numbers* can be obtained. Then we introduce a new specific variant of halting avoiding the trap symbol, which allows us to save a lot of rules, i.e., rules involving the trap symbol. The rest of the section deals with catalysts without states. All results of this section have one feature in common: even under maximal parallelism, all *halting* computations have a small universal bound (typically, the number of catalytic objects plus one) on the degree of actual parallelism.

The task of generating a non-semilinear set by catalytic P systems is rather complicated. Not only that we have no mass influence like we had in Section 9.4, but we also cannot control the evolution by states like in the previous constructions. Although catalytic P systems are known to be universal, a direct translation of a register machine generating powers of 2 yields a rather big number of rules. Below we present two optimized constructions. The first one generates a vector set; since the underlying register machine is partially blind (it does not have zero-test instructions), it allows us to further simplify the simulation. The latter generates a non-semilinear set of numbers, which relies on the simulation of two zero-test instructions in addition to the decrement instructions of the underlying register machine.

9.6.1 Generating a Non-semilinear Number Set with One Bi-stable Catalyst

Whereas in Subsection 9.5.1 an accepting P systems with one bi-stable catalyst has been established, we now turn our attention to the more difficult generative case, using the maximally parallel mode.

$$\begin{aligned}
\Pi_{18} &= (O, C = \{c,\bar{c}\}, \mu = [\ ]_1, w_1 = \bar{c}pa, R_1, f = 1) \text{ where} \\
O &= \{c,\bar{c}\} \cup \{a,b\} \cup \{p,q,s,t\} \cup \{\#\}, \\
R_1 &= \{ca \to \bar{c}bb,\ s \to t,\ \bar{c}t \to cs,\ ct \to \bar{c}p,\ t \to \#,\ \# \to \#, \\
&\qquad \bar{c}b \to ca,\ p \to q,\ cq \to \bar{c}p,\ \bar{c}q \to cs,\ \bar{c}p \to \bar{c},\ q \to \#\}.
\end{aligned}$$

Π_{18} works in two phases. In phase 1, in addition to the bi-stable catalyst toggling between c and $\bar{c}$, there is a state object present, toggling between s and t. Every two steps, a is replaced by bb with c changing to $\bar{c}$ while s changes to t; then $\bar{c}t$ are reset to cs. If objects a are no longer present, c is idle for one step, and then ct change to $\bar{c}p$, entering phase 2.

In phase 2, the state object toggles between p and q. Every two steps, b is renamed into a with $\bar{c}$ changing to c while p changes to q; then cq are reset to $\bar{c}p$. If objects b

are no longer present, either $\bar{c}$ is idle for one step, and then $\bar{c}p$ change to cs, returning to phase 1, or $\bar{c}$ erases p thus causing the system to halt.

In both phases, if the bi-stable catalyst "chooses" a "wrong"rule, then either t or q are left to themselves, forcing the system to enter an infinite computation, so this computation does not lead to a result any more; hence, in sum we obtain $mL_{gen,maxpar}(\Pi_{18}) = \{a^{2^n} \mid n \geq 0\}$.

Π_{18} is a purely catalytic P system with only one bi-stable catalyst and 12 rules, hence, we have

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}OP_1\left(2cat_1\right)_{12}.$$

The computational completeness of P systems with one bi-stable catalyst working in the maximally parallel mode was established in [2].

9.6.2 A Special Variant of Halting Avoiding the Trap Symbol

Throughout this section, a main part of the constructions (which is inevitable for proving computational completeness, too) is the introduction of the trap symbol # in case the derivation goes the wrong way and by the rule $\# \to \#$ (or $c\# \to c\#$ with a catalyst c) guaranteeing the derivation never to halt. Yet all these rules can be avoided if we apply a specific *new halting strategy* allowing the system to halt without yielding a result. This somehow corresponds to the case of a Turing machine halting its computation in a non-final state, and as for Turing machines, where each such computation halting in a non-final state can be transformed into an infinite computation, in P systems usually the trap rules perform this task to yield an infinite computation.

This specific *new halting strategy* allowing the system to halt without yielding a result can be defined in various ways, e.g., with a special symbol (like the trap symbol) appearing in the configuration. As our main goal in this paper is to save as many rules as possible, we want to stop one step earlier, i.e., before the trap symbol would be introduced. Hence, we specify a specific subset of *toxic* objects O_{tox}; the P system is only allowed to continue a computation from a configuration C by using an applicable multiset of rules covering all copies of objects from O_{tox} occurring in C; moreover, if there exists no multiset of applicable rules covering all toxic objects, the whole computation having yielded the configuration C is abandoned, i.e., no results can be obtained from this computation.

This idea somehow resembles the strategy of using priorities with having the rules involving the objects from O_{tox} having priority over other rules. Yet this strategy is both weaker and stronger compared with the strategy of priority on rules: on one hand, we can only specify priorities with respect to the objects from O_{tox}; on the other hand, if not all copies of *toxic* objects from O_{tox} can be covered by any multiset of rules, we stop without getting a result, whereas in the case of priorities any multiset of rules respecting the priority relation can be used to continue the computation.

For any variant of P systems, we add the set of *toxic* objects O_{tox} and in the specification of the families of sets of (vectors of) numbers generated/accepted by P systems with toxic objects using rules of type X we add the subscript tox to O, thus obtaining the families $Y_{\gamma,maxpar}O_{tox}P_m\,(X)$, for any $\gamma \in \{gen, acc\}$ and $m \geq 1$.

Hence, for the P system with one bi-stable catalyst Π_{18} elaborated in the preceding subsection, we obtain the corresponding P system Π_{19} with toxic objects

$$\begin{aligned} \Pi_{19} &= (O, C = \{c, \bar{c}\}, O_{tox}, \mu = [\ ]_1, w_1 = \bar{c}pa, R_1, f = 1) \text{ where} \\ O &= \{c, \bar{c}\} \cup \{a, b\} \cup \{p, q, s, t\}, \\ O_{tox} &= \{q, t\}, \\ R_1 &= \{ca \to \bar{c}bb,\ s \to t,\ \bar{c}t \to cs,\ ct \to \bar{c}p, \\ &\qquad \bar{c}b \to ca,\ p \to q,\ cq \to \bar{c}p,\ \bar{c}q \to cs,\ \bar{c}p \to \bar{c}\}. \end{aligned}$$

According to the arguments established in the preceding subsection, we immediately infer $mL_{gen,maxpar}(\Pi_{19}) = \{a^{2^n} \mid n \geq 0\}$. This system now does not need more than 9 rules, and therefore we have

$$\{2^n \mid n \geq 0\} \in N_{gen,maxpar}O_{tox}P_1\,(2cat_1)_9\,.$$

9.6.3 General Results for [Purely] Catalytic P Systems with Toxic Objects

Looking closer into the computational completeness proofs for catalytic P systems given in [9], we see that the only non-cooperative rules used in the proofs given there are rules involving the trap symbol. When going to purely catalytic P systems, we realize that all rules involving the trap symbol can be assigned to one additional catalyst; for example, to generate any recursively enumerable set of natural numbers we need two catalysts for catalytic P systems and three catalysts for purely catalytic P systems.

As the proof of the basic result

$$PsRE = Ps_{gen,maxpar}OP_1(cat_2) = Ps_{gen,maxpar}OP_1(pcat_3)$$

also is the basis of the construction of the [purely] catalytic P system elaborated in Subsection 9.6.5, we first recall the main ideas of the simulations for ADD- and SUB-instructions of a register machine $M = (d, B, l_0, h, R)$.

For every instruction label $j \in B$ of the register machine to be simulated two symbols are used to keep the two main catalysts c_1, c_2 busy, starting with $p_j\tilde{p}_j$. At the end, for the halting label h we use the two rules $c_1p_h \to c_1$ and $c_2\tilde{p}_h \to c_2$ to stop the derivation in case the simulation has succeeded to be correct. The number n_a stored in register a is represented by n_a copies of the symbol o_a.

Each ADD-instruction $j : (ADD(a), k, l)$, for $a \in \{1, 2, \ldots, d\}$, can easily be simulated by the rules $c_1p_j \to c_1o_ap_k\tilde{p}_k$, $c_1p_j \to c_1o_ap_l\tilde{p}_l$, and $c_2\tilde{p}_j \to c_2$.

Each SUB-instruction $j : (SUB(a), k, l)$, only necessary to be considered for $a \in 1, 2$, is simulated in four steps as shown in the table listed below:

Simulation of the SUB-instruction $j : (SUB(a), k, l)$ if

register a is not empty	register a is empty
$c_a p_j \to c_a \hat{p}_j \hat{p}'_j$	$c_a p_j \to c_a \bar{p}_j \bar{p}'_j \bar{p}''_j$
$c_{3-a} \tilde{p}_j \to c_{3-a}$	$c_{3-a} \tilde{p}_j \to c_{3-a}$
$c_a o_a \to c_a c'_a$	$c_a \bar{p}_j \to c_a$
$c_{3-a} \hat{p}_j \to c_{3-a}$	$c_{3-a} \bar{p}''_j \to c_{3-a} p''_j$
$c_a c'_a \to c_a c''_a$	
$c_{3-a} \hat{p}'_j \to c_{3-a} \hat{p}''_j$	$c_{3-a} p''_j \to c_{3-a} p'_j$
$c_a \hat{p}''_j \to c_a p_k \tilde{p}_k$	$c_a p'_j \to c_a p_l \tilde{p}_l$
$c_{3-a} c''_a \to c_{3-a}$	$c_{3-a} \bar{p}'_j \to c_{3-a}$

In addition, trap rules guarantee that in case the guess whether the contents of register a is empty or not was wrong, the derivation enters an infinite loop with the rule $\# \to \#$ in the catalytic case or $c_3\# \to c_3\#$ in the purely catalytic case. These objects x for which we have such trap rules $x \to \#$ in the catalytic case or $c_3 x \to c_3\#$ in the purely catalytic case, are c'_1, c''_1, c'_2, c''_2 and, for every label j of a SUB-instruction $j : (SUB(a), k, l)$, the objects $p_j, p'_j, p''_j, \hat{p}_j, \hat{p}''_j, \tilde{p}_j, \bar{p}_j, \bar{p}''_j$, and for every label j of an ADD-instruction $j : (ADD(a), k, l)$ the objects $p_j, \tilde{p}_j$ as well. We should like to mention that these trap rules for the objects $p_j, \tilde{p}_j$ coming from the ADD-instructions $j : (ADD(a), k, l)$ were forgotten to be included in the proof of the special Corollary 8 in [9], whereas in the more general Theorem 4, due to writing down the range of the trap rules in a different way, these trap rules for the symbols indicating an ADD-instruction were included correctly.

The construction shown above strictly follows the proof elaborated in [9]; yet we observe that many symbols are just needed to keep one of the two catalysts busy for one step with being erased or else having to evolve to the trap symbol. Hence, every symbol x for which we only have a rule $c_i x \to c_i \lambda$, $i \in \{1, 2\}$, can be replaced by just one symbol d_i. In addition, for p_j we now always use c_1 and for $\tilde{p}_j$ (which in fact now is replaced by d_2) we now always use c_2. Finally, we can also replace all symbols $\bar{p}'_j$ by just one variable $\hat{d}_{3-a}$.

In that way, we obtain the following tables of rules for the simulation of ADD-instructions and SUB-instructions:

Simulation of the ADD-instruction $j : (ADD(a), k, l)$

$c_1 p_j \to c_1 o_a p_k d_2,\ \ c_1 p_j \to c_1 o_a p_k d_2;\ \ c_2 d_2 \to c_2.$

The table for a SUB-instruction now contains several identical entries:

Simulation of the SUB-instruction $j : (SUB(a),k,l)$ if

register a is not empty	register a is empty
$c_1 p_j \to c_1 d_{3-a} \hat{p}'_j$	$c_1 p_j \to c_1 d_a \hat{d}_{3-a} \bar{p}''_j$
$c_2 d_2 \to c_2$	$c_2 d_2 \to c_2$
$c_a o_a \to c_a c'_a$	$c_a d_a \to c_a$
$c_{3-a} d_{3-a} \to c_{3-a}$	$c_{3-a} \bar{p}''_j \to c_{3-a} p''_j$
$c_a c'_a \to c_a d_{3-a}$	
$c_{3-a} \hat{p}'_j \to c_{3-a} \hat{p}''_j$	$c_{3-a} p''_j \to c_{3-a} p'_j$
$c_a \hat{p}''_j \to c_a p_k d_2$	$c_a p'_j \to c_a p_l d_2$
$c_{3-a} d_{3-a} \to c_{3-a}$	$c_{3-a} \hat{d}_{3-a} \to c_{3-a}$

The zero-test case now can be reduced considerably to two steps:

$c_1 p_j \to c_1 \bar{p}_j$
$c_2 d_2 \to c_2$
c_a remains idle
$c_{3-a} \bar{p}_j \to c_{3-a} p_l d_2$

For x being an element of the following set, we have to add the trap rules $c_3 x \to c_3 \#$ in the purely catalytic case and the corresponding rules $x \to \#$ in the catalytic case:

$$\{\#, d_1, d_2, c'_1, c'_2\} \cup \{p_j \mid j : (ADD(a),k,l)\} \cup \{p_j, \hat{p}''_j, \bar{p}_j \mid j : (SUB(a),k,l)\}$$

In the case of catalytic P systems, the only non-cooperative rules are these trap rules, and in the case of purely catalytic P systems, the trap rules, and only those, are associated with the third catalyst c_3. If we take exactly those objects for which such a trap rule exists as *toxic objects* and omit all trap rules, then we immediately infer the following computational completeness result, where O_{tox} indicates that we are using toxic objects:

Theorem 4. $PsRE = PsO_{tox}P_{gen,maxpar}(cat_2) = PsO_{tox}P_{gen,maxpar}(pcat_2)$.

In general, for all the results elaborated in [9] we obtain similar results: when using toxic objects, then the construction for obtaining the results for catalytic and purely catalytic P systems coincide, as for example in the preceding theorem, where in both cases we only need two catalysts (which number currently is assumed to be the minimal one). Moreover, the simulation becomes somehow deterministic, as only the correct simulation paths survive; in that sense, deterministic register machines can be simulated by *deterministic* [purely] catalytic P systems with toxic objects.

9.6.4 Generating a Non-Semilinear Vector Set with a [Purely] Catalytic P System

We now construct [purely] catalytic P systems generating the non-semilinear set of pairs of natural numbers $\{(n,m) \mid n \leq m \leq 2^n\}$. First we define the purely catalytic P system Π_{20} generating $\{a_3^n a_4^m \mid n \leq m \leq 2^n\}$.

$$\begin{aligned}
\Pi_{20} &= (O, C = \{c_1,c_2,c_3\}, \mu = [\ ]_1, w_1 = c_1c_2c_3a_1p_1d_1d_2, R_1, f = 1) \text{ where} \\
O &= \{c_1,c_2,c_3\} \cup \{a_1,a_2,a_3,a_4,d_1,d_2,p_1,p_2,\#\}, \\
R_1 &= \{c_1a_1 \to c_1,\ c_1p_2 \to c_1a_1a_4p_2,\ c_1p_2 \to c_1a_1a_3a_4p_1,\ c_1p_2 \to c_1a_3a_4, \\
&\qquad c_2a_2 \to c_2,\ c_2p_1 \to c_2a_2a_2p_1,\ c_2p_1 \to c_2a_2a_2p_2, \\
&\qquad c_1d_2 \to c_1,\ c_2d_1 \to c_2,\ c_1d_1 \to c_1\#,\ c_2d_2 \to c_2\#, \\
&\qquad c_3p_1 \to c_3\#,\ c_3p_2 \to c_3\#,\ c_3\# \to c_3\#\}.
\end{aligned}$$

This purely catalytic system has 14 rules, and with just omitting the third catalyst we obtain the corresponding catalytic P system having 14 rules, too. But with using toxic objects we can save some of (but in this case not all!) the trap rules, i.e., we obtain the purely catalytic P system with toxic objects Π_{21} with only 11 rules:

$$\begin{aligned}
\Pi_{21} &= (O, C = \{c_1,c_2\}, O_{tox}, \mu = [\ ]_1, w_1 = c_1c_2a_1p_1d_1d_2, R_1, f = 1) \text{ where} \\
O &= \{c_1,c_2\} \cup \{a_1,a_2,a_3,a_4,d_1,d_2,p_1,p_2,\#\}, \\
O_{tox} &= \{p_1,p_2,\#\}, \\
R_1 &= \{c_1a_1 \to c_1,\ c_1p_2 \to c_1a_1a_4p_2,\ c_1p_2 \to c_1a_1a_3a_4p_1,\ c_1p_2 \to c_1a_3a_4, \\
&\qquad c_2a_2 \to c_2,\ c_2p_1 \to c_2a_2a_2p_1,\ c_2p_1 \to c_2a_2a_2p_2, \\
&\qquad c_1d_2 \to c_1,\ c_2d_1 \to c_2,\ c_1d_1 \to c_1\#,\ c_2d_2 \to c_2\#\}.
\end{aligned}$$

In all cases, the derivations work as follows: the objects p_1, p_2 work as states, and the objects d_i, $i \in \{1,2\}$, are used to check that the corresponding catalyst c_i is busy at some stage, otherwise these objects d_i force the system to enter an infinite loop with the trap rules or to *kill* the derivation.

In state p_1, we decrement (the number of objects representing) register 1 by the rule $c_1a_1 \to c_1$ and double their number by applying the rule $c_2p_1 \to c_2a_2a_2p_1$ in parallel. By using the rule $c_2p_1 \to c_2a_2a_2p_2$ instead, we change to state p_2, yet without checking whether register 1 is already empty. In case the latter rule is used too late, i.e., if no object a_1 is present any more, then we have to use the trap rule $c_1d_1 \to c_1\#$.

In state p_2, we decrement (the number of objects representing) register 2 by the rule $c_2a_2 \to c_2$ and copy (the contents of) this register to register 1, at the same time adding this number to register 4 by using the rule $c_1p_2 \to c_1a_1a_4p_2$ in parallel. If this rule is used until no object a_2 is present any more, then we have to use the trap rule $c_2d_2 \to c_2\#$. If instead we use the rule $c_1p_2 \to c_1a_1a_3a_4p_1$, we switch back to state 1, at the same moment incrementing register 3, yet again without checking register a_2 for being zero. By using the rule $c_1p_2 \to c_1a_3a_4$, we end this cycling between states p_1 and p_2, i.e., now both p_1 and p_2 are not present any more, and the two objects d_1 and d_2 have to be eliminated, which can be achieved by using the two rules $c_1d_2 \to c_1$ and $c_2d_1 \to c_2$ in parallel.

At the end of a computation, even if the two objects d_1 and d_2 have already been deleted, the catalysts c_1 and c_2 will still be active to delete all the remaining objects a_1 and a_2, hence, at the end only copies of objects a_3 and a_4 are present any more. In sum, we conclude that $Ps(\{a_3^n a_4^m \mid n \leq m \leq 2^n\})$ belongs to

$$Ps_{gen,maxpar}OP_1(pcat_3)_{14} \cap Ps_{gen,maxpar}O_{tox}P_1(pcat_2)_{11}.$$

9.6.5 Generating Number Sets with [Purely] Catalytic P Systems

We now are going to improve the result from [19], where a catalytic P system with 54 rules was elaborated, generating the non-semilinear set of natural numbers $\{2^n - 2n \mid n \geq 2\}$. In the following, we construct a [purely] catalytic P system with 29 rules, generating the (standard) non-semilinear set of natural numbers $\{2^n \mid n \geq 1\}$. Yet we will show even more, i.e., our construction works for any set of natural numbers representing a function $g : \mathbb{N} \to \mathbb{N}$ which is computed in r_3 by the following function program (starting with $r_1 = b_0$ and $r_2 = r_3 = 0$), with r_1, r_2, r_3 representing three registers of a register machine, and with the parameters b_0, b_1, b_2, b_3, b_4 being natural numbers, $b_0 \geq 1$:

```
function $g(r_3)$:
1: if $r_1 > 0$
        then begin DEC($r_1$); ADD($1,r_2$); goto 1 end
        else goto 2
        orelse begin ADD($b_3,r_3$); goto 3 end
2: if $r_2 > 0$
        then begin DEC($r_2$); ADD($b_2,r_1$); ADD($b_1,r_3$);
        goto 2 end
        else begin ADD($b_4,r_1$); goto 1 end;
3: HALT
endfunction
```

The idea of this program is that in label 1 we copy (register) r_1 to r_2; the notation DEC(r) means decrementing (register) r by one, whereas ADD(k,r) means adding k to (register) r. As soon as register r_1 is empty, we switch to label 2 or halt after having added b_3 to (the result register) r_3 before. In label 2, we copy back the value of register r_2 to r_1, but take it b_2 times, at the same time adding b_1 times the value of register r_2 to r_3, and at the end, when r_2 is empty, we add b_4 to r_1.

The structures

```
i: if $r_i > 0$
        then begin DEC($r_i$); ADD($1,r_{3-i}$); goto i end
        else goto j
```

in this function program correspond with the following instructions in a register machine program:

$i : (SUB(i), i', j)$
$i' : (ADD(3-i), i, i)$

We now describe the functions computed by specific values of the parameters b_0, b_1, b_2, b_3, b_4; thereby let $f_i(n)$, $i \in \{1,2,3\}$, denote the value of register i after n times, $n \geq 0$, having gone through the loops 1 and 2, and $g(n)$ the final value of the function when going through the loops 1 and 2 for n times and then performing loop 1 once more, yet exiting at the end of loop 1 to halt.

In general, we get $f_2(0) = f_2(n) = 0$ for all $n \geq 0$ as well as the system of linear recursions

$$\begin{aligned} f_1(n+1) &= b_2 f_1(n) + b_4, \\ f_3(n+1) &= f_3(n) + b_1 f_1(n) \end{aligned}$$

with $f_1(0) = b_0$ and $f_3(0) = 0$ as well as the final result $g(n) = f_3(n) + b_3$.

Case 1. $b_2 = 1$:

In this case, we get the system of linear recursions

$$\begin{aligned} f_1(n+1) &= f_1(n) + b_4, \\ f_3(n+1) &= f_3(n) + b_1 f_1(n). \end{aligned}$$

Solving these recursions yields $f_1(n) = b_0 + b_4 n$ and, for $n \geq 0$,

$$\begin{aligned} f_3(n) &= f_3(0) + \sum_{i=0}^{n-1} b_1 f_1(i) = b_1 \sum_{i=0}^{n-1} (b_0 + b_4 i) \\ &= b_1 b_0 n + b_1 b_4 n(n-1)/2 \end{aligned}$$

hence, $g(0) = b_3$ and, for $n \geq 0$,

$$g(n) = f_3(n) + b_3 = (b_1 b_4/2)n^2 + (b_1 b_0 - b_1 b_4/2)n + b_3,$$

i.e., a quadratic function provided $b_1 \neq 0$ and $b_4 \neq 0$.

As a specific example, for $(b_0, b_1, b_2, b_3, b_4) = (1,1,1,0,2)$ we obtain $g(n) = n^2$.

Case 2. $b_2 > 1, b_1 = 1, b_4 = 0$:

In this case, we get the linear recursions

$$\begin{aligned} f_1(n+1) &= b_2 f_1(n), \\ f_3(n+1) &= f_3(n) + f_1(n) \end{aligned}$$

with $f_1(0) = b_0$ and $f_2(0) = f_3(0) = 0$ as well as the final result $g(n) = f_3(n) + b_3$, i.e., for $n \geq 0$ we obtain $f_1(n) = b_0(b_2)^n$ and

$$f_3(n) = f_3(0) + \sum_{i=0}^{n-1} b_0 (b_2)^i = b_0 \sum_{i=1}^{n-1} (b_2)^i = b_0(((b_2)^n - 1)/(b_2 - 1))$$

as well as

$$g(n) = f_3(n) + b_3 = b_0(((b_2)^n - 1)/(b_2 - 1)) + b_3.$$

As a specific example, for $(b_0, b_1, b_2, b_3, b_4) = (1, 1, 2, 1, 0)$ we therefore obtain $g(n) = 2^n$.

We now start from the constructions for simulating SUB-instructions as already exhibited in Subsection 9.6.3. In order to get even more efficient simulations, we save the first steps in a specific way; moreover, every ADD-instruction can be incorporated into the rules of the last steps of the simulations, in a similar way as this was already done in the construction elaborated in [19]. Thus, for the function g with the parameters b_0, b_1, b_2, b_3, b_4 we construct the catalytic P system

$$\begin{aligned}\Pi_{22}(b_0,b_1,b_2,b_3,b_4) &= (O, C = \{c_1, c_2\}, \mu = [\ ]_1, w_1, R_1, f = 1) \text{ where}\\ O &= \{a_1, a_2, a_3, c_1', c_2', d, \#\}\\ &\cup \{p_j, p_j', p_j'', \bar{p}_j \mid j \in \{1,2\}\},\\ w_1 &= c_1 c_2 (a_1)^{b_0} p_1,\\ R_1 &= R_{1,c} \cup R_{1,\#},\end{aligned}$$

and $R_{1,c}$ consists of the catalytic rules contained in the following two tables:

Simulation of the instructions related with label 1 if

register 1 is not empty	register 1 is empty
$c_1 a_1 \to c_1 c_1'$	c_1 remains idle
$c_2 p_1 \to c_2 p_1'$	$c_2 \bar{p}_1 \to c_2 p_2$
$c_1 c_1' \to c_1 d$	
$c_2 p_1' \to c_2 p_1''$	
$c_1 p_1'' \to c_1 p_1 a_2$ or	
$c_1 p_1'' \to c_1 \bar{p}_1 a_2$	
$c_2 d \to c_2$	
halting:	$c_2 \bar{p}_1 \to c_2 (a_3)^3$

Simulation of the instructions related with label 2 if

register 2 is not empty	register 1 is empty
$c_2 a_2 \to c_2 c_2'$	c_2 remains idle
$c_1 p_2 \to c_1 p_2'$	$c_1 \bar{p}_2 \to c_1 p_1 (a_1)^{b_4}$
$c_2 c_2' \to c_2 d$	
$c_1 p_2' \to c_1 p_2''$	
$c_2 p_2'' \to c_2 p_2 (a_1)^{b_2} (a_3)^{b_1}$ or	
$c_2 p_2'' \to c_2 \bar{p}_2 (a_1)^{b_2} (a_3)^{b_1}$	
$c_1 d \to c_1$	

In addition, we have to add trap rules to guarantee that in case of wrong guesses, the derivation enters an infinite loop with the rule $\# \to \#$ in the catalytic case (or $c_3\# \to c_3\#$ in the purely catalytic case). The objects x for which we have such trap rules $x \to \#$ in the catalytic case (or $c_3 x \to c_3\#$ in the purely catalytic case) are # and d as well as, for $j \in \{1,2\}$, the objects $c'_j, p_j, p'_j, p''_j, \bar{p}_j$, i.e.,

$$R_{1,\#} = \{x \to \# \mid x \in \{\#, d\} \cup \{c'_j, p_j, p'_j, p''_j, \bar{p}_j \mid j \in \{1,2\}\}\}.$$

In total this yields 17 catalytic rules in $R_{1,c}$ and 12 trap rules in $R_{1,\#}$, i.e., 29 rules in R_1. Obviously, the same number of rules is obtained for the corresponding purely catalytic P system $\Pi'_{22}(b_0,b_1,b_2,b_3,b_4)$ where we simply have to add the third catalyst c_3 and replace the context-free trap rules $x \to \#$ by the corresponding catalytic trap rules $c_3 x \to c_3\#$.

We can omit the trap rules when using toxic objects, i.e., if we take

$$\begin{aligned}
\Pi''_{22}(b_0,b_1,b_2,b_3,b_4) &= (O'', C = \{c_1,c_2\}, O_{tox}, \mu = [\ ]_1, w_1, R_{1,c}, f = 1) \text{ where} \\
O'' &= \{a_1,a_2,a_3,c'_1,c'_2,d\} \\
&\cup \{p_j, p'_j, p''_j, \bar{p}_j \mid j \in \{1,2\}\}, \\
O_{tox} &= \{c'_1,c'_2,d\} \cup \{p_j, p'_j, p''_j, \bar{p}_j \mid j \in \{1,2\}\}, \\
w_1 &= c_1 c_2 (a_1)^{b_0} p_1,
\end{aligned}$$

and $R_{1,c}$ contains the catalytic rules as listed above. This system now only contains 17 rules.

How to argue that the catalytic P system $\Pi_{22}(b_0,b_1,b_2,b_3,b_4)$ and the corresponding catalytic P systems $\Pi'_{22}(b_0,b_1,b_2,b_3,b_4)$ work correctly was exhibited in detail in [9] as well as in [19]. Yet as we have reduced the number of rules in a considerable way, we have to argue for all possible cases of decrementing or zero-test register a:

If decrementing of register a is possible, all steps have to be performed exactly as described in the table. If decrementing fails, then in the last (third) step c_{3-a} must be used with the rule $c_{3-a}a_{3-a} \to c_{3-a}c'_{3-a}$ as not both registers can be empty during a computation in the register machine. Yet in the next step the catalyst c_{3-a} is busy with the program symbol p_a or $\bar{p}_a$, hence, with c'_{3-a} and one of these program symbols competing for the same catalyst, one of these symbols will be trapped.

A successful simulation of testing register a for zero is performed in one step leaving catalyst c_a idle. In case the register is not empty, c'_a has to be generated, and this symbol in the next step will compete for the catalyst c_a with the program symbol p_{3-a} and thus one of these symbols will be trapped.

As explained in [19], we here also mention that when using $c_2\bar{p}_2 \to c_2(a_3)^{b_3}$ instead of $c_2\bar{p}_1 \to c_2 p_2$ in order to reach a halting configuration, the system does not immediately halt, but instead, if having chosen the rule when register 1 is empty, uses the sequences of rules $c_2 a_2 \to c_2 c'_2$, $c_2 c'_2 \to c_2 d$, and $c_2 d \to c_2$ (or $c_1 d \to c_1$), to clear register 2, so that in the end only the objects a_3 remain besides the catalysts. If $c_2\bar{p}_2 \to c_2(a_3)^{b_3}$ is chosen too early, then both registers may be cleared by using the corresponding rules. The result expressed by the number of symbols a_3 is not affected, if we make such a wrong choice at the end only.

As specific examples, we therefore obtain

$$N_{gen,maxpar}(\Pi_{22}(1,1,1,0,2)) = \{n^2 \mid n \geq 0\}$$

and

$$N_{gen,maxpar}(\Pi_{22}(1,1,2,1,0)) = \{2^n \mid n \geq 0\}.$$

We observe that with respect to the complexity of the systems, especially concerning the number of rules, there is no difference at all between the sets of natural numbers growing in a quadratic and in an exponential way, respectively.

In sum, we conclude that all these non-linear sets of natural numbers as described above are contained in

$$N_{gen,maxpar}OP_1(cat_2)_{29} \cap N_{gen,maxpar}OP_1(pcat_3)_{29}$$

as well as in

$$N_{gen,maxpar}O_{tox}P_1(cat_2)_{17} \cap N_{gen,maxpar}O_{tox}P_1(pcat_2)_{17}.$$

If we do not limit ourselves with the number of catalysts, a better solution with respect to the number of rules is possible, i.e., for the function g with the parameters b_0, b_1, b_2, b_3, b_4 we construct the purely catalytic P system

$$\begin{aligned} \Pi_{23}(b_0,b_1,b_2,b_3,b_4) &= (O, C, \mu = [\ ]_1, w_1, R_1, f = 1) \text{ where} \\ O &= C \cup \{a_1,a_2,a_3,p_1,p_2,p_h,d_1,d_2,d_1',d_2',d,d',\#\} \\ C &= \{c_{r,Decr}, c_{r,0Test} \mid r \in \{1,2\}\} \cup \{c_d, c_p, c_\#\} \\ w_1 &= c_{1,Decr}c_{1,0Test}c_{2,Decr}c_{2,0Test}c_dc_pc_\#(a_1)^{b_0}p_1d_2d_1'd_2'dd', \end{aligned}$$

and R_1 consists of the catalytic rules described in the following.

The main idea of this new construction is to use two catalysts for each register – one for the decrement ($c_{r,Decr}$) and one for the zero-test ($c_{r,0Test}$). Moreover, each SUB-instruction is simulated by two rules, one for the decrement and one for the zero-test, just allowing the corresponding catalyst to do its work, whereas all other catalysts are kept busy having introduced d_r for $c_{r,Decr}$ and d_r' for $c_{r,0Test}$. The catalyst c_d for the special symbol d is kept busy by d'; the symbol d is used for trapping in case an intended decrement fails and can only be allowed to vanish in the last step. The catalyst c_p is used with the instruction labels p_1, p_2, and p_h. The catalyst $c_\#$ is only needed for handling the trap symbol #.

For each of the two registers $r \in \{1,2\}$, the following rules perform the decrement and the zero-test, respectively, in case this operation is initiated by omitting d_r or d_r', respectively, in the step before.

decrement $c_{r,Decr}a_r \to c_{r,Decr}$: if register r is not empty, it is decremented;
$c_{r,Decr}d \to c_{r,Decr}\#$: if register r is empty, the catalyst $c_{r,Decr}$ has to be used with the symbol d thereby introducing the trap symbol #;
$c_{r,Decr}d_r \to c_{r,Decr}$: d_r keeps $c_{r,Decr}$ busy if another instruction is to be simulated;
$c_\#d_r \to c_\#\#$: d_r is a "toxic" object which must not stay idle;
$c_dd' \to c_d$: d' keeps c_d busy until the end;
$c_\#d' \to c_\#\#$: d' is a "toxic" object which must not stay idle.

zero-test $c_{r,0Test}a_r \to c_{r,0Test}\#$: if register r is not empty, a trap symbol # is generated;

$c_{r,0Test}d_r' \to c_{r,0Test}$: d_r' keeps $c_{r,0Test}$ busy if another instruction is to be simulated; for the symbol d_r' we do not need an additional trap rule as the only alternative is already a trap rule.

The following rules initiate the decrement or the zero-test on register 1 or 2 and simulate the program for the function g:

(1) $c_pp_1 \to c_pa_2p_1d_2d_1'd_2'd'$: decrement register 1;
$c_pp_1 \to c_pp_2d_1d_2d_2'd'$: zero-test register 1;
(2) $c_pp_2 \to c_p(a_1)^{b_2}(a_3)^{b_1}p_2d_1d_1'd_2'd'$: decrement register 2;
$c_pp_2 \to c_p(a_1)^{b_4}p_1d_1d_2d_1'd'$: zero-test register 2;
$c_pp_1 \to c_p(a_3)^{b_3}p_hd_1d_2d_2'd'$: zero-test register 1 and go to halting.
(3) Whereas register 1 is already empty, now also register 2 has to be cleaned using the instruction label p_h:
$c_pp_h \to c_pp_hd_1d_1'd_2'd'$: decrement register 2;
$c_pp_h \to c_pd_1d_2d_1'$: zero-test register 2 and eliminate p_h;
$c_dd \to c_d$: finally c_d is allowed to eliminate d;
$c_\#\# \to c_\#\#$: in case something goes wrong during a simulation of an instruction, this rule keeps the P system in an infinite loop.

In sum, this purley catalytic P system $\Pi_{23}(b_0,b_1,b_2,b_3,b_4)$ contains only 23 rules. We can save two catalysts by using non-cooperative rules instead of the catalytic rules assingned to the catalysts c_p and $c_\#$, thus obtaining the catalytic P system $\Pi_{23}'(b_0,b_1,b_2,b_3,b_4)$. Hence, all the non-linear sets of natural numbers described above are contained in

$$N_{gen,maxpar}OP_1(cat_5)_{23} \cap N_{gen,maxpar}OP_1(pcat_7)_{23}.$$

Besides the trap rule $c_\#\# \to c_\#\#$, only the rules $c_\#d_r \to c_\#\#$, $r \in \{1,2\}$, and $c_\#d' \to c_\#\#$ can be omitted when considering a (purely) catalytic P system with "toxic" objects, yet this result with 19 rules is even weaker than the previous one where we also used less catalysts.

9.7 Conclusions

In this paper we have illustrated that many models of P systems need (rather) small numbers n of rules to define, i.e., to accept or to generate, some specific non-semilinear sets of vectors of natural numbers or non-semilinear sets of natural numbers; all the specific results obtained in this paper are summarized in Table 1.

For the catalytic P systems/purely catalytic P systems it is still one of the most challenging questions in the area of P systems whether we really need two/three catalysts to get computational completeness or at least to accept or generate a non-semilinear set of (vectors of) natural numbers. Moreover, whereas some (very small) numbers in the table above are already optimal, especially the bigger numbers might allow for (considerable) improvements, but at the moment are the best solutions known according to our knowledge.

Table 1 Examples for using n rules in specific models of P systems

n	models
1	– deterministic cooperative rules, accepting numbers with a special (non-standard) halting and accepting condition – non-cooperative rules with target agreement, generating numbers
2	– non-cooperative rules with target selection, generating numbers – non-cooperative rules with label selection, generating numbers – non-cooperative rules with tables, generating numbers – non-cooperative rules with membrane dissolution, generating numbers – non-cooperative rules in active membranes with two polarizations, generating numbers – non-cooperative rules with one inhibitor, generating numbers
3	– non-deterministic cooperative rules, accepting numbers – symport/antiport rules of weight ≤ 2 (and size ≤ 4), accepting numbers – non-cooperative rules with one promoter, generating numbers
4	– purely catalytic rules with one bi-stable catalyst, generating vectors – symport/antiport rules of weight ≤ 2 (and size ≤ 3), accepting numbers
5	– non-cooperative rules with promoters/inhibitors, generating numbers
9	– [purely] catalytic rules with one bi-stable catalyst and toxic objects, generating numbers
11	– (purely) catalytic rules with two (three) catalysts and toxic objects, generating vectors
12	– [purely] catalytic rules with one bi-stable catalyst, generating numbers
14	– (purely) catalytic rules with two (three) catalysts, generating vectors
17	– (purely) catalytic rules with two (three) catalysts and toxic objects, generating numbers
23	– (purely) catalytic rules with five (seven) catalysts, generating numbers
29	– (purely) catalytic rules with two (three), generating numbers

Acknowledgements. The first author acknowledges project STCU-5384 *Models of high performance computations based on biological and quantum approaches* awarded by the Science and Technology Center in the Ukraine. Both authors are very grateful to Petr Sosík for pointing out the mistake in the proof of Corollary 8 in [9].

References

1. Alhazov, A., Freund, R.: Asynchronous and Maximally Parallel Deterministic Controlled Non-cooperative P Systems Characterize *NFIN* and *coNFIN*. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, Gy. (eds.) CMC 2012. LNCS, vol. 7762, pp. 101–111. Springer, Heidelberg (2013)
2. Alhazov, A.: P Systems without Multiplicities of Symbol-Objects. Information Processing Letters 100(3), 124–129 (2006)

3. Alhazov, A., Freund, R., Păun, Gh.: Computational Completeness of P Systems with Active Membranes and Two Polarizations. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 82–92. Springer, Heidelberg (2005)
4. Alhazov, A., Freund, R., Riscos-Núñez, A.: One and Two Polarizations, Membrane Creation and Objects Complexity in P Systems. In: Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2005, pp. 385–394. IEEE Computer Society (2005)
5. Alhazov, A., Verlan, S.: Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. Theoretical Computer Science 412(17), 1581–1591 (2011)
6. Beyreder, M., Freund, R.: Membrane Systems Using Noncooperative Rules with Unconditional Halting. In: Corne, D.W., Frisco, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 129–136. Springer, Heidelberg (2009)
7. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer (1989)
8. Freund, R.: Special Variants of P Systems Inducing an Infinite Hierarchy with Respect to the Number of Membranes. Bulletin of the EATCS 75, 209–219 (2001)
9. Freund, R., Kari, L., Oswald, M., Sosík, P.: Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient. Theoretical Computer Science 330(2), 251–266 (2005)
10. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (Tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu., Rozenberg, G., Salomaa, A. (eds.) CMC 2013. LNCS, vol. 8340, pp. 173–188. Springer, Heidelberg (2014)
11. Ibarra, O.H., Woodworth, S.: On Symport/Antiport P Systems with a Small Number of Objects. International Journal of Computer Mathematics 83(7), 613–629, 137–152 (2006)
12. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs (1967)
13. Păun, Gh.: Computing with Membranes. J. Comput. Syst. Sci. 61, 108–143 (2000); also see TUCS Report 208 (1998), `www.tucs.fi`
14. Păun, Gh.: Membrane Computing. An Introduction. Springer (2002)
15. Păun, Gh., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing, pp. 118–143. Oxford University Press (2010)
16. Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press, New York (1980)
17. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, 3 vols. Springer (1997)
18. Sburlan, D.: Further Results on P Systems with Promoters/Inhibitors. International Journal of Foundations of Computer Science 17(1), 205–221 (2006)
19. Sosík, P.: A Catalytic P System with Two Catalysts Generating a Non-Semilinear Set. Romanian Journal of Information Science and Technology 16(1), 3–9 (2013)
20. The P systems webpage, `http://ppage.psystems.eu`

Chapter 10
Generalized Communicating P Automata

Erzsébet Csuhaj-Varjú and György Vaszil

Abstract. In this paper we introduce and study generalized communicating P automata, computing devices that combine properties of classical automata and generalized communicating P systems. We show that certain variants of these constructs are able to accept any recursively enumerable language even with a small number of cells which interact with each other using only one type of very simple communication rules, but there are resticted types which recognize only the class of regular languages.

10.1 Introduction

P automata are computing devices which combine properties of classical automata and P systems. The notion of a P system or a membrane system, a computational framework inspired by functioning and architecture of living cells, was introduced by Gheorghe Păun in 1998, his seminal paper appeared in 2000, [7]. The concept of a P automaton was introduced in [2]. Since 2000, membrane computing has become a well-established, active scientific area. For basic information on P systems the reader is referred to [8], for details on P automata consult [1].

The notion of a generalized communicating P system was defined in [11], in order to provide a common generalization of various purely communicating models in P systems theory. A generalized communicating P system can be considered as a hypergraph where each node is represented by a cell and each edge is represented

Erzsébet Csuhaj-Varjú
Department of Algorithms and Their Applications, Faculty of Informatics,
Eötvös Loránd University, Pázmány Péter sétány 1/c, 1117 Budapest, Hungary
e-mail: csuhaj@inf.elte.hu

György Vaszil
Department of Computer Science, Faculty of Informatics,
University of Debrecen, Kassai út 26, 4028 Debrecen, Hungary
e-mail: vaszil.gyorgy@inf.unideb.hu

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_10

by a rule. Every cell contains a multiset of objects which — by communication rules — can be communicated between the cells. The form of a communication rule is $(a,i)(b,j) \to (a,k)(b,l)$ where a and b are objects and i,j,k,l are labels identifying the input and the output cells. Such a rule means that an object a from cell i and an object b from cell j move synchronously to cell k and cell l, respectively. Often, communication rules can also be considered as interaction rules.

Depending on their form, several restrictions on communication rules (modulo symmetry) were introduced; see [11] for these variants. When a generalized communicating P system has only one type of these restricted rules, then we speak of a generalized communicating P system with minimal interaction or a minimal interaction P system (with the given type of rules).

Due to the simplicity of rules, the generative power of minimal interaction P systems is of particular interest and it has been studied in details. In [11] and [4] it was proved that eight of the possible nine restricted variants (with respect to the form of rules) are able to generate any recursively enumerable set of numbers; in the ninth case only finite sets of singletons can be obtained. Furthermore, these systems even with relatively small numbers of cells and simple underlying (hypergraph) architectures are able to achieve this generative power. The necessary number cells to obtain this power was significantly reduced in [6], and in [3] it was shown that minimal interaction P systems where the alphabet of objects is a singleton given with any of the so-called parallel-shift, presence-move, chain, and join rules are able to generate every recursively enumerable set of natural numbers.

Cells of generalized communicating P systems communicate with their environment. In this paper, we describe the behavior of generalized communicating P systems by the sequences of multisets of symbols entering some designated membranes from the environment. That is, we consider the generalized communicating P system as an automaton that changes its state according to the obtained input. As the behaviour of a conventional automaton can be represented by the set of input sequences it accepts, the behaviour of generalized communicating P automaton can be described by the set of words that are obtained from its accepted multiset sequences by certain mapping. The concept of generalized communicating P automaton is a variant of P automata, purely communicating P systems functioning similarly to conventional automata.

We show that most of these constructs are computationally complete computing devices, even using only rules of type split, or symport2, or join, or conditional-uniport-in, are able to recognize any recursively enumerable language, while in the case of antiport1 rules they recognize the elements of the class of regular languages.

10.2 Preliminaries and Definitions

We assume that the reader is familiar with the basics of formal language theory and membrane computing; for more information, we refer to the monograph [10], and the handbooks [9] and [8].

In the following, we briefly review the notions and the notation we will use. An alphabet is a finite non-empty set of symbols. Given an alphabet V, we denote the set of strings over V by V^*. If the empty string, ε, is not included, then we use the notation V^+.

A finite multiset over an alphabet V is a mapping $M : V \to \mathbb{N}$ where $\mathbb{N}$ denotes the set of non-negative integers, and $M(a)$ for $a \in V$ is said to be the multiplicity of a in M. The multiset M can also be represented by any permutation of a string $w = a_1^{M(a_1)} a_2^{M(a_2)} \ldots a_n^{M(a_n)} \in V^*$, where if $M(x) \neq 0$, then there exists j, $1 \leq j \leq n$, such that $x = a_j$. The set of all finite multisets over an alphabet V is denoted by V^*, the empty multiset is denoted by $\emptyset$ as in the same way as the empty set. We define the union and the difference of two multisets $M_3 = M_1 \cup M_2$ and $M_4 = M_1 \setminus M_2$ as $M_3(a) = M_1(a) + M_2(a)$ and $M_4(a) = \max(M_1(a) - M_2(a), 0)$, for all $a \in V$, respectively. Sometimes we also identify a set S with the multiset where all elements of S are present with multiplicity one, so we write expressions like $M_2 = M_1 \cup S$ or $M_2 = M_1 \setminus S$ where M_i, $1 \leq i \leq 2$, are multisets over some alphabet V and $S \subseteq V$ is a set.

A generalized communicating P automaton is a P system with generalized communication which is placed in an environment of objects. During its functioning, similarly to "ordinary" P automata, it consumes multisets from the environment. The accepted sequences of multisets are those which enter one of some designated input membranes during a computation which ends in one of the previously defined accepting configurations.

The formal definition of a generalized communicating P automaton is as follows.

Definition 1. *A* generalized communicating P automaton *with $n \geq 1$ membranes is a construct $\Pi = (V, E, c_0, P, I_{in}, \mathcal{F})$ where*

- *V is an alphabet of objects and $E \subseteq V$ is the set of objects present in an arbitrary number of copies in the environment;*
- *$c_0 = (w_0, w_1, \ldots, w_n)$, is the initial configuration of Π where $w_i \in V^*$, $1 \leq i \leq n$, corresponds to the multiset of objects contained initially by cell i, and $w_0 \in (V - E)^*$ corresponds to the multiset of objects from $V - E$ initially in the environment;*
- *$P \subseteq (V \times \mathbb{N})^2 \times (V \times \mathbb{N})^2$ is a finite set of interaction rules of the form $(a,i)(b,j) \to (a,k)(b,l)$ where $a, b \in V$, $0 \leq i, j, k, l \leq n$. Moreover, if $i = j = 0$, then $\{a,b\} \cap (V - E) \neq \emptyset$, that is, $a \notin E$ or $b \notin E$;*
- *$I_{in} \subseteq \{1, \ldots n\}$ is the set of indices of the input cells; and*
- *$\mathcal{F}$ is a finite set of n-tuples $(v_0, \ldots, v_n)$ where $v_i \in V^*$ or $v_i = \#$, $0 \leq i \leq n$, denoting the set of accepting configurations of Π.*

The system consists of n cells containing multisets of objects over V, and there is an additional region, numbered by 0, called the environment. The environment contains the objects of $E \subseteq V$ in an infinite number of copies. The cells interact with each other by means of the rules in P. Such an interaction rule of the form $(a,i)(b,j) \to (a,k)(b,l)$ may be applied if there is an object a in cell i and an object b in cell j. As a result, the object a moves from cell i to cell k and b moves from cell j to cell l.

Now we define the transition relation of a generalized communicating P automaton.

Definition 2. *Let $\Pi = (V, E, c_0, P, I_{in}, \mathcal{F})$ be a generalized communicating P automaton. The* transition mapping *of Π, $\delta : V^* \times (V^*)^{n+1} \to 2^{(V^*)^{n+1}}$ is defined as follows:*

For two configurations $c, c' \in (V^*)^{n+1}$, $c' \in \delta(u, c)$ if the multiset of symbols $u \in V^*$ enters any of the input cells $i_{in} \in I_{in}$ from the environment while the configuration of Π changes from c to c' as a result of applying a maximal subset of its rules in parallel.

By the maximal parallel application of the rules, we assign objects to rules nondeterministically in such a way that after the assignment no further rule can be applied to the remaining objects. A rule can be applied in the same step as many times as we would like, only the number of copies of objects matters. Those objects which are left unassigned, remain in the cell where they are and are passed to the next configuration unchanged.

Note that if both objects a, b come from the environment of the system, then at least one of them must not be an element of E, that is, at least one of them must not appear in the environment in an infinite number of copies. This requirement is necessary since otherwise the maximal parallel rule application could bring an infinite number of objects into the system in one step, which we would like to avoid.

The language accepted by the generalized communicating P automaton is obtained by considering the multiset sequences accepted by the system.

Definition 3. *The set of* accepted input sequences *of a generalized communicating P automaton $\Pi = (V, E, c_0, P, I_{in}, \mathcal{F})$ is defined by*

$$A(\Pi) = \{v_1, \ldots, v_s \mid v_i \in V^*, \text{ there are } c_0, c_1, \ldots c_s \text{ such that } c_i \in \delta(v_i, c_{i-1}),\ 1 \leq i \leq s, \text{ and } c_s \in \mathcal{F}\}.$$

The final configurations are given as $(v_0, \ldots, v_n) \in \mathcal{F}$ where v_i, $0 \leq i \leq n$, are either multisets over V, or the symbol #. A configuration $(u_0, \ldots, u_n) \in (V^*)^{n+1}$ is a final configuration, if there is a $(v_0, \ldots, v_n) \in \mathcal{F}$, such that, for all i, $0 \leq i \leq n$, where $v_i \neq \#$, $u_i = v_i$. (For all such i, where $v_i = \#$, the multiset $u_i \in V^*$ can be arbitrary).

The words of the accepted language are obtained by applying a mapping to the accepted multiset sequences. In order to restrict the power of generalized communicating P automata, we consider mappings which are "nonerasing" in the sense that they do not decrease the length of the sequences they are applied to.

Definition 4. *Let $\Pi = (V, E, c_0, P, I_{in}, \mathcal{F})$ be a generalized communicating P automaton, Σ be a finite alphabet, and $f : V^* \to \Sigma^*$ be a mapping from the set of finite multisets over V to the set of strings over Σ such that $f(u) = \varepsilon$ if and only if $u = \emptyset$. The* language *accepted by Π with respect to f is defined as*

$$L(\Pi, f) = \{f(v_1) \ldots f(v_s) \in \Sigma^* \mid v_1, \ldots, v_s \in A(\Pi)\}.$$

Following [11], we list the possible types of interaction rules (modulo symmetry) that can be used in a generalized communicating P automaton.

Let V be an alphabet and let us consider an interaction rule $(a,i)(b,j) \to (a,k)(b,l)$ with $a,b \in V$, $i,j,k,l \geq 0$. There are several cases:

(1) $i = j = k \neq l$: the rule is of type *conditional-uniport-out* (*uout* in short), it sends b to cell l provided that a and b are in cell i.
(2) $i = k = l \neq j$: the rule is of type *conditional-uniport-in* (*uin* in short), it brings b to cell i provided that a is also in cell i.
(3) $i = j, k = l, i \neq k$: the rule is of type *symport2* (*sym*2 in short), it moves a and b together from cell i to k.
(4) $i = l, j = k, i \neq j$: the rules is of type *antiport1* (*anti*1 in short), as the result of its application, a and b are exchanged in cells i and k.
(5) $i = k, i \neq j, i \neq l, j \neq l$: the rule is of type *presence-move* (*presence* in short), it moves the object b from cell j to l, provided that there is an object a in cell i and i, j, l are pairwise different cells.
(6) $i = j, i \neq k, i \neq l, k \neq l$: the rule is of type *split*, it sends a and b from cell i to cells k and l, respectively.
(7) $k = l, i \neq j, k \neq i, k \neq j$: the rule is of type *join*, it brings a from cell i and b from cell j together in cell k.
(8) $i = l, i \neq j, i \neq k\ and j \neq k$: the rule is of type *chain*, it moves a from cell i to cell k while b is moved from cell j to cell i, i.e., to the cell where a was previously.
(9) i, j, k, l are pairwise different: the rule is of type *parallel-shift* (*shift* in short), it moves a and b from two different cells to other two different cells.

Let $\mathcal{L}(PA_n, X)$ denote the class of languages accepted by generalized communicating P automata having n cells where $n \geq 1$, and using only rules of type $X \in \{uout, uin, sym2, anti1, presence, split, join, chain, shift\}$.

In the following we will also need the notion of a k-counter machine from [5].

Definition 5. *A k-*counter machine *for some $k \in \mathbb{N}$, is given by $M = (k, Q, \Sigma, q_0, P, F)$ where Q is a set of internal states with initial state $q_0 \in Q$ and set of accepting states $F \subseteq Q$; Σ is the finite input alphabet; and P is a set of instructions of the form $(x, p, (i_1, c_{i_1}), \ldots, (i_l, c_{i_l}); q, e_{i_1}, \ldots, e_{i_l})$ where $x \in \Sigma \cup \{\varepsilon\}$, $p, q \in Q$, $i_j \in \{1, \ldots, k\}$, and $c_{i_j} \in \{> 0, = 0\}$, $e_{i_j} \in \{+1, -1, 0\}$, $1 \leq j \leq l$.*

By a rule of the above form, M being in the state p may enter state $q \in Q$, if the value stored in the i_jth counter corresponds to $c_{i_j} \in \{> 0, = 0\}$, $1 \leq j \leq l$ (that is, the stored value is zero if c_{i_j} is "$= 0$" or nonzero if c_{i_j} is "≥ 0"). In case of the application of the rule, the counters are modified according to $e_{i_1}, \ldots, e_{i_l} \in \{+1, -1, 0\}$ (denoting increment, decrement, or leaving the value unchanged). If $x \in \Sigma$, then the machine scanned x on the input tape, and the head moves one cell to the right; if $x = \varepsilon$, then the machine performs the transition irrespectively of the scanned input symbol, and the reading head does not move.

A configuration of the k-counter machine M can be denoted by a $(k+2)$-tuple $(w, q, j_1, \ldots, j_k)$ where $w \in \Sigma^*$ is the unread part of the input word written on the input tape, $q \in Q$ is the state of the machine, and $j_i \in \mathbb{N}$, $1 \leq i \leq k$, is the value stored in the ith counter of M. For two configurations, C and C', we write $C \Rightarrow C'$ if M is capable of changing the configuration $C = (xw, q, j_1, \dots, j_k)$ to $C' = (w, q', j'_1, \dots, j'_n)$ by reading a symbol $x \in \Sigma \cup \{\varepsilon\}$ from the input tape by applying one of its transition rules.

A word $w \in \Sigma^*$ is accepted by the machine if starting in the initial configuration $(w, q_0, 0, \dots, 0)$, the input head eventually reaches and reads the rightmost non-blank symbol on the input tape, and M is in an accepting state $q_F \in F$, that is, it reaches a configuration $(\varepsilon, q_f, j_1, \dots, j_k)$ for some $j_i \in \mathbb{N}$, $1 \leq i \leq k$.

Definition 6. *The* language *accepted by the k-counter machine M is defined as*

$$\begin{aligned} L(M) = \{w \in \Sigma^* \mid &\mathit{where}\ (w, q_0, 0, \dots, 0) \Rightarrow^* (\varepsilon, q_f, c_1, \dots, c_k) \\ &\mathit{for\ some}\ q_f \in F\} \end{aligned}$$

where $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

In the following, we will use k-counter machines in two special forms which we will call in this paper the first and the second normal forms.

A k-counter machine $M = (k, Q, \Sigma, q_0, P, F)$ is in the *first normal form*, if the instructions of P can be grouped into pairs of the following form

- $(x, p, (i, > 0); q, -1)$ and $(x, p, (i, = 0); r, 0)$ for some $x \in \Sigma \cup \{\varepsilon\}$, $p, q, r \in Q$, $i \in \{1, \dots, k\}$, or
- $(\varepsilon, p, (i, > 0); q, +1)$ and $(\varepsilon, p, (i, = 0); q, +1)$ for some $p, q \in Q$, $i \in \{1, \dots, k\}$.

These instruction pairs describe two types of changes the machine can make in its configuration in one step: First, it can check whether the value stored in a counter is zero or not, and based on the result of the check, it can either switch to an internal state and decrement the counter, or switch to another internal state and leave the counter unchanged. Second, it can increment a counter irrespective of its contents while not reading anything from the input tape. Thus, for the sake of easier readability, we will denote the above defined instruction pairs as single instructions

- $(x, p; C(i)-, q, r)$, and
- $(\varepsilon, p; C(i)+, q)$,

where $x \in \Sigma \cup \{\varepsilon\}$, $p, q, r \in Q$, $i \in \{1, \dots, k\}$.

The instructions are interpreted as before, thus, a configuration $d = (w, q, c_1, \dots, c_k)$ can be changed to the configuration $d' = (w', q', c'_1, \dots, c'_k)$, if

- there is an instruction $(x, p; C(i)-, r, s) \in P$ such that $w' = xw$, $q = p$, and either $c_i > 0$, $q' = r$, and $c'_i = c_i - 1$, or $c_i = 0$, $q' = s$, and $c'_i = c_i$;
- there is an instruction $(\varepsilon, p; C(i)+, r) \in P$ such that $q = p$, $q' = r$, and $c'_i = c_i + 1$. In this case $w' = w$.

To see that these types of machines are equivalent to the ones given in the usual way by Definition 5, we present the following lemma.

Lemma 1. *For any k-counter machine M, there is a k-counter machine M' in the first normal form, such that $L(M') = L(M)$.*

Proof. Let $M = (k, Q, \Sigma, q_0, P, F)$ be a k-counter machine. First we show how to convert the instruction set in such a way that each instruction works with one of the counters only. Let $M'' = (k, Q'', \Sigma, q_0, P'', F)$ be defined as $Q'' = Q \cup \{p_{t,1}, \dots, p_{t,k-1} \mid p \in Q,\ t \in P\}$, and let P'' be defined as follows.

$$\begin{aligned} P'' = {} & \{(x,p,(i_1,c_{i_1});p_{t,1},e_{i_1}) \mid t:(x,p,(i_1,c_{i_1}),\dots,(i_l,c_{i_l});q,e_{i_1},\dots,e_{i_l}) \in P\} \cup \\ & \{(\varepsilon,p_{t,j-1},(i_j,c_{i_j});p_{t,j},e_{i_j}) \mid t:(x,p,(i_1,c_{i_1}),\dots,(i_l,c_{i_l});q,e_{i_1},\dots,e_{i_l}) \in P, \\ & \quad 2 \le j \le k-1\} \cup \\ & \{(\varepsilon,p_{t,l-1},(i_l,c_{i_l});q,e_{i_l}) \mid t:(x,p,(i_1,c_{i_1}),\dots,(i_l,c_{i_l});q,e_{i_1},\dots,e_{i_l}) \in P\}. \end{aligned}$$

The instruction set P'' contains for each instruction $t \in P$ which works with $l \le k$ counters, a set of l instructions which imply the same change in the configuration of the machine, but in l steps instead of the one step of M.

Now from M'', we construct $M' = (k, Q', \Sigma, q_0, P', F)$ in the first normal form, as follows. Ler $Q' = Q'' \cup \{q_T\} \cup \{p_{t,1}, p_{t,2} \mid p \in Q, t \in P''\}$, and for each instruction in P'', we introduce one or two instructions in P' as follows.

- For $t:(x,p,(i,>0);q,-1) \in P''$, we add $(x,p;C(i)-,q,q_T)$ to P';
- for $t:(x,p,(i,>0);q,0) \in P''$, we add $(x,p;C(i)-,p_{t,1},q_T)$ and $(\varepsilon,p_{t,1};C(i)+,q)$ to P';
- for $(x,p,(i,>0);q,+1) \in P''$, we add $(x,p;C(i)-,p_{t,1},q_T)$, $(\varepsilon,p_{t,1};C(i)+,p_{t,2})$, and $(\varepsilon,p_{t,2};C(i)+,q)$ to P';
- for $(x,p,(i,=0);q,0) \in P''$, we add $(x,p;C(i)-,q_T,q)$ to P'; and
- for $(x,p,(i,=0);q,+1) \in P''$, we add $(x,p;C(i)-,q_T,p_{t,1})$ and $(\varepsilon,p_{t,1};C(i)+,q)$ to P'.

It is not difficult to see why the machine M' being in normal form is equivalent to M. □

We say that a k-counter machine $M = (k, Q, \Sigma, q_0, P, F)$ is in the *second normal form*, if P only contains instructions of the form

- $(\varepsilon,p,(i,>0);q,-1)$ or $(\varepsilon,p,(i,=0);r,0)$ for some $p,q,r \in Q$, $i \in \{1,\dots,k\}$,
- $(x,p,(i,>0);q,+1)$ or $(x,p,(i,=0);q,+1)$ for some $x \in \Sigma \cup \{\varepsilon\}$, $p,q \in Q$, $i \in \{1,\dots,k\}$.

Lemma 2. *For any k-counter machine M, there is a k-counter machine M′ in the second normal form, such that* $L(M') = L(M)$.

Proof. Let $M = (k, Q, \Sigma, q_0, P, F)$ be a k-counter machine. First, we construct $M'' = (k, Q'', \Sigma, q_0, P'', F)$ where each instruction works with one of the counters only. This is done in the same way as in the proof of Lemma 1.

Now from M'', we construct $M' = (k, Q', \Sigma, q_0, P', F)$ in the second normal form, as follows. Let $Q' = Q'' \cup \{p_{t,1}, p_{t,2} \mid p \in Q'', t \in P''\}$, and for each instruction in P'' which does not conform to the normal form definition above, we introduce new instructions in P' as follows.

- For $t:(x,p,(i,>0);q,-1)\in P''$ with $x\neq\varepsilon$, we add $(x,p,(i,>0);q_{t,1},+1)$, $(\varepsilon,q_{t,1},(i,>0);q_{t,2},-1)$ and $(\varepsilon,q_{t,2},(i,>0);q,-1)$ to P';
- for $t:(x,p,(i,=0);q,0)\in P''$ with $x\neq\varepsilon$, we add $(x,p,(i,=0);q_{t,1},+1)$, and $(\varepsilon,q_{t,1},(i,>0);q,-1)$ to P'.

It is not difficult to see that the machine M' is in the second normal form, and it is equivalent to M. □

10.3 The Power of P Automata with Generalized Communication

First we examine the power of systems with split rules.

Theorem 3. $\mathcal{L}(PA_9,split)=RE$.

Proof. Let $L\subseteq\Sigma^*$ and let $M=(k,Q,\Sigma,q_0,P,F)$ be a k-counter machine in the first normal form accepting L. We construct a generalized communicating P automaton $\Pi=(V,E,c_0,P',\{4\},\mathcal{F})$ with nine membranes which accepts the language $L(M)$.

Let $V=E\cup\{q,q_r,\bar{q}_r\mid q\in Q,r\in P\}\cup\{X,Y,Z\}$ where $E=\Sigma\cup\{C_i\mid 1\leq i\leq k\}$, and let $c_0=(w_0,w_1,\ldots,w_9)$ with $w_0=w_3=w_5=w_7=\emptyset$, $w_1=\{q_0,X\}$, $w_2=\{q_r\mid q\in Q,r\in P\}$, $w_4=Q\setminus\{q_0\}$, $w_6=\{Y\}$, $w_8=\{\bar{q}_r\mid q\in Q,r\in P\}$, and $w_9=\{Z\}$.

The application of a rule of M is simulated by several steps of Π. The symbol representing the current state of the k-counter machine is in cell 1, while the values of the counters are stored in cell 5 in the form of symbols C_i where i refers to the ith counter, $1\leq i\leq k$.

The rule set P' of Π is defined as follows. For any $r:(\varepsilon,p,C(i)+;q)\in P$ we have in P'

$$\begin{array}{ll} r_1:(p,1)(X,1)\rightarrow(p,2)(X,3), & r_2:(p,2)(p_r,2)\rightarrow(p,4)(p_r,0),\\ r_3:(p_r,0)(C_i,0)\rightarrow(p_r,3)(C_i,5), & r_4:(p_r,3)(X,3)\rightarrow(p_r,4)(X,1),\\ r_5:(p_r,4)(q,4)\rightarrow(p_r,2)(q,1). & \end{array}$$

For any $r:(a,p,C(i)-;q,s)\in P,\ a\neq\varepsilon$, we have in P'

$$\begin{array}{ll} r_6:(p,1)(X,1)\rightarrow(p,2)(X,7), & r_7:(p,2)(p_r,2)\rightarrow(p,7)(p_r,0),\\ r_8:(p_r,0)(a,0)\rightarrow(p_r,5)(a,4), & r_9:(p,7)(X,7)\rightarrow(p,6)(X,3),\\ r_{10}:(p_r,5)(C_i,5)\rightarrow(p_r,3)(C_i,0), & r_{11}:(p_r,3)(X,3)\rightarrow(p_r,7)(X,5),\\ r_{12}:(p_r,4)(q,4)\rightarrow(p_r,2)(q,1), & r_{13}:(p,5)(X,5)\rightarrow(p,4)(X,1),\\ r_{14}:(p,6)(Y,6)\rightarrow(p,5)(Y,7), & r_{15}:(p_r,7)(Y,7)\rightarrow(p_r,4)(Y,6),\\ r_{16}:(p,5)(p_r,5)\rightarrow(p,8)(p_r,2), & r_{17}:(p,8)(\bar{p}_r,8)\rightarrow(p,4)(\bar{p}_r,9),\\ r_{18}:(\bar{p}_r,9)(Z,9)\rightarrow(\bar{p}_r,4)(Z,7), & r_{19}:(\bar{p}_r,4)(s,4)\rightarrow(\bar{p}_r,3)(s,1),\\ r_{20}:(\bar{p}_r,3)(X,3)\rightarrow(\bar{p}_r,8)(X,1), & r_{21}:(Z,7)(Y,7)\rightarrow(Z,9)(Y,6). \end{array}$$

For any $r:(\varepsilon,p,C(i)-;q,s)\in P$, we have in P' a similar rule set as above. Instead of r_6-r_9 we have

$$r_{22}:(p,1)(X,1)\rightarrow(p,2)(X,3),\ r_{23}:(p,2)(p_r,2)\rightarrow(p,6)(p_r,5),$$

and the rest of the rules, $r_{10}-r_{21}$, are the same as above.

We show that Π simulates M.

The simulation of a rule $r : (\varepsilon, p, C(i)+; q) \in P$ is done by using rules of the form r_1-r_5, as follows. First from cell 1 symbol p moves to cell 2 and auxiliary symbol X moves to cell 3. In the next step, p continues its way to cell 4 and p_r from cell 2 is sent to the environment. Note that in any moment of time there is at most one element of type p_r, $p \in Q$, $r \in R$ is in the environment and the choice of p_r determines the rule to be simulated. Then, p_r and a copy of C_i enter the system, p_r moves to cell 3 and C_i to cell 5. In the next two steps p_r moves to cell 4 and then to cell 2 , X to cell 1 and symbol q from cell 4 to cell 1. Thus, the obtained configuration corresponds to the new configuration of M, i.e., it changed its state to q without reading any symbol and the number stored in counter i is increased by one. Due to the construction of the rule set of P' no other rules can be applied during this phase of generation and thus the simulation is correct.

The simulation of $r : (a, p, C(i)-; q, s) \in P$, $a \neq \varepsilon$ in Π is done by rules of the form r_6-r_{21}, as follows: At the first step symbol p moves from cell 1 to cell 2 and symbol X to cell 7 (rule r_6) . Then, p continues to move to cell 7 and p_r is sent to the environment (rule r_7). After then p_r brings a letter a from the environment that moves to cell 4 and meantime p_r enters cell 5 (rule r_3). In parallel, symbol p and X leave cell 7 and move to cells 6 and 3, respectively (rule r_9). By this phase of the generation the reading of letter a is simulated.

Then, there are two cases: cell 5 contains at least one C_i (counter i is not empty) or there is no C_i in cell 5 (counter i is empty). In the first case, p_r leaves cell 5, moves to cell 3 and one occurrence of C_i is sent out to the environment (rule r_{10}). Then, p_r continues its way from cell 3 to cell 7 and X from cell 3 to cell 5 (rule r_{11}), and in parallel, symbol p and symbol Y move from cell 6 to cells 5 and 7, respectively (rule r_{14}). Then, symbol p_r is forwarded to cell 4 and symbol Y returns to cell 6 (rule r_{15}), and in parallel, from cell 5, p moves to to cell 4 and X to cell 1 (rule r_{13}). Finally, symbols p_r and q leave cell 4 and move to cells 2 and 1, respectively (rule r_{12}). Thus, the simulation of configuration change, when M in state p reads letter a, changes its state to q, and decrements the value of counter i by one is finished. It is easy to see that during this phase of the generation no other rule is applicable and the application of no other rule of M is simulated.

If no C_i can be found in cell 5, then the next applicable rule of P' moves p and Y from cell 6 to cells 5 and 7, respectively (rule r_{14}). After then p leaves cell 5 to cell 8 and p_r leaves cell 5 to cell 2 (rule r_{16}). In the next step, p leaves to cell 4 and $\bar{p}_r$ from cell 8 moves to cell 9 (rule r_{17}). Then $\bar{p}_r$ and Z leave cell 9 and $\bar{p}_r$ moves to cell 4 and Z moves to cell 7 (rule r_{18}). Then, $\bar{p}_r$ and s leave cell 4, $\bar{p}_r$ enters cell 3 and s cell 1 (rule r_{19}). In parallel, Z and Y from cell 7 move to cells 8 and 6, respectively (rule r_{21}). At the last step, $\bar{p}_r$ and X leave cell 3 and move to cells 8 and cell 1, respectively (rule r_{20}). This generation phase simulates the configuration change of M when it changes its state p to s, due to the fact that the value of counter i is zero, i.e., it cannot be decremented. As in the previous case, it can be seen that the rules can be applied only in the above order, and during this phase of the generation, the application of no other rule of M can be simulated.

Finally, we show how the simulation of a rule $r : (\varepsilon, p, C(i)-; q, s) \in P$ is done by rules of Π. The procedure is very similar to the previous one, the only difference is that instead of reading some letter a, M does not read any symbol. Thus, instead of using rules r_6-r_9, we use rules r_{22} and r_{23}, which result in moving p from cell 1 via cell 2 to cell 6, symbol X form cell 1 to cell 3 and symbol p_r from cell 2 to cell 5. Then, we use rules of the form r_{10}-r_{21} and the same reasoning as above.

Due to the construction of Π, only the instructions that M has and is able to perform can be simulated. Thus, if we have $\mathcal{F} = \{(\emptyset, qX, \#, \ldots, \#) \mid q \in F\}$ as the set of accepting configurations, and f is the identity function (we define $f : V^* \to V^*$ as $f(a) = a$ for all $a \in V$), then $L(\Pi, f)$ contains the same words as $L(M)$. □

According to the previous statement, generalized communicating P automata can accept any recursively enumerable language with split rules. Since split rules can be simulated by sym2 rules, Theorem 3 also implies the computational completeness of systems with sym2 rules only. This is shown in the next theorem.

Theorem 4. $\mathcal{L}(PA_*, sym2) = RE$.

Proof. Let $L \subseteq \Sigma^*$ be an arbitrary language and let Π be the generalized communicating P automaton from the proof of Theorem 3 using split rules and the mapping f, accepting the language $L(\Pi, f) = L$.

Notice that Π is constructed in such a way that no rule is used two or more times in parallel, and that no rule is used in two consecutive computational steps. Under these conditions, we can simulate split rules with symport2 rules, and thus, construct a system Π' with $L(\Pi) = L(\Pi')$ having only symport2 rules.

To prove the statement, we recall a construction from [4] that demonstrates how a split rule $l : (a,i)(b,i) \to (a,j)(b,k)$ where a, b are objects, i, j, k are numbers of cells, and l is the label of the rule, can be simulated by symport2 rules.

Let Π' contain for any split rule l of Π a cell 1_l (for different rules different cells), and initially let this cell contain objects A_l and B_l, and let cells j and k contain objects A'_l and B'_l, respectively (objects A_l, B_l, A'_l, B'_l are pairwise different and different from the objects of Π). Suppose furthermore that E contains one more object D, different from all of the previous ones, including the objects of Π.

By [4], the following rules simulate rule l:

$$\begin{array}{ll}
l_1 : (a,i)(b,i) \to (a,1_l)(b,1_l), & \\
l_2 : (a,1_l)(A_l,1_l) \to (a,j)(A_l,j), & l_3 : (b,1_l)(B_l,1_l) \to (b,k)(B_l,k), \\
l_4 : (A_l,j)(A'_l,j) \to (A_l,0)(A'_l,0), & l_5 : (A_l,0)(D,0) \to (A_l,1_l)(D,1_l), \\
l_6 : (B_l,k)(B'_l,k)) \to (B_l,0)(B'_l,0), & l_7 : (B_l,0)(D,0) \to (B_l,1_l)(D,1_l), \\
l_8 : (A'_l,0)(D,0) \to (A'_l,j)(D,j), & l_9 : (B'_l,0)(D,0) \to (B'_l,k)(D,k).
\end{array}$$

It can easily be seen that rule l_1 sends a and b to cell 1_l. From there, a moves to cell j and b moves to cell k, accompanied by symbols A_l and B_l, respectively. After then, by rules l_4 -l_9 the accompanying symbols return to cell 1_l. Notice that during this phase of functioning simulation of no other split rules can start.

Let $\Pi = (V, E, c_0, P, I_{in}, \mathcal{F})$ be the system from the proof of Theorem 3. Now Π' is obtained from Π by adding for any rule l of Π a cell 1_l, as described above, such that initially this cell contain objects A_l and B_l and let add to the initial contents of cells j and k objects A'_l and B'_l, respectively. Let E have one more symbol D, see above, and let us add for each rule l one more cell 2_l.

Now P' is obtained by replacing any split rule l with the set of rules above, with the exception of l_1, instead of which, the new rules

$$l'_1 : (a,i)(b,i) \to (a,2_l)(b,2_l),\ l''_1 : (a,2_l)(b,2_l) \to (a,1_l)(b,1_l),$$

should be used.

If we set $I_{in} = \{2_l \mid l \in P\}$, and we have $f' : V^* \to V^*$ with $f'(ua) = (a)$ for any multiset ua where $u \in V^*,\ a \in \Sigma$, then it is not difficult to see that $L(\Pi', f') = L(\Pi, f)$ holds. □

Now we turn to systems with rules of type antiport1. As shown in the next theorem, they only characterize the class of regular languages.

Theorem 5. $\mathcal{L}(PA_*, anti1) = REG.$

Proof. We first show that $\mathcal{L}(PA_*, anti1) \subseteq REG$. Let $\Pi = (V, E, c_0, P, I_{in}, \mathcal{F})$ be an arbitrary generalized communicating P automaton, Σ be a finite alphabet, and $f : V^* \to \Sigma^*$ be a mapping which satisfies conditions of Definition 4. Since the application of any rule of Π means exchange of symbols between two cells (including the environment), therefore the total number of objects inside the regions of Π remains unchanged under its functioning. This implies that the number of configurations of Π and thus the number of multisets entering the input cell (and thus the number of their f-images) are bounded by some constants. Then, starting from the transition mapping δ of Π, we can construct a nondeterministic finite automaton $M = (Q, \Sigma, \delta', q_0, F)$ such that $L(M) = L(\Pi, f)$ holds. (For M, Q denotes the finite nonempty set of states, Σ is the input alphabet, δ' is the transition mapping, q_0 is the initial state, and F is the set of accepting states; for details on deterministic and non-deterministic finite automata consult [9].) Let $C = \{c_0, \ldots, c_n\}$ be the set of configurations of Π and $\bar{V}$ be the set of multisets of objects that enter the input cells under functioning of Π. For any transition $c_i \in \delta(v, c_j)$ in Π, where $0 \le i, j \le n$, $v \in \bar{V}$, we construct a series of transitions in δ' as follows: if $f(v) = x_1 \ldots x_m$, where $x_1, \ldots, x_m \in \Sigma$, then $[c_{j_1}] \in \delta'(x_1, [c_j])$, $[c_{j_k}] = \delta'(x_k, [c_{j_{k-1}}])$ for $2 \le k \le m-1$, and $[c_i] = \delta'(x_m, [c_{j_{m-1}}])$. Let us define Q as the set of all symbols $[c_{j_k}]$, where $c_j \in C$, and $1 \le k \le max$, where max is the length of the longest word $f(v)$ entering the input cell from the environment. Let $F = \{[c_l] \mid c_l \in \mathcal{F}\}$ and $q_0 = [c_0]$. By the construction, it is easy to see that $L(M) = L(\Pi, f)$, which implies that $\mathcal{L}(PA_*, anti1) \subseteq REG$.

Next we show that the reverse inclusion also holds. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite deterministic automaton. We construct a generalized communicating P automaton $\Pi = (V, E, c_0, P, \{1\}, \mathcal{F})$ and give a mapping f such that $L(\Pi, f) = L(M)$ holds. Let Π have only one cell and let $V = \{q_{ini}\} \cup \{[p,a,q] \mid p, q \in Q, a \in \Sigma, \delta(p,a) = q\}$. Let $E = \{[p,a,q] \mid p, q \in Q, a \in \Sigma, \delta(p,a) = q\}$, $w_0 = q_{ini}$, $\mathcal{F} = \{[p,a,q] \mid p \in Q, q \in F, a \in \Sigma, \delta(p,a) = q\}$.

Let P consist of the following rules:

- $(p_{ini},1)([q_0,a,p],0) \to (p_{ini},0)([q_0,a,p],1)$, for any transition $\delta(q_0,a) = p$ in M, where $a \in \Sigma$,
- $([p,a,q],1)([q,b,r],0) \to ([p,a,q],0)([q,b,r],1)$, where $\delta(p,a) = q$ and $\delta(q,b) = r$ are transitions following each other in M.

Let $\mathcal{F} = \{[p,a,q] \mid p,q \in Q, a \in \Sigma, q \in F, \delta(p,a) = q\}$.

It is easy to see that Π simulates M. If we define $f[p,a,q] = a$ for each $p,q \in Q$ $a \in \Sigma$ where $\delta(p,a) = q$ in M, then it comes immediately that $L(\Pi,f) = L(M)$ holds. □

Theorem 6. $\mathcal{L}(PA_8, join) = RE$.

Proof. Let $L \subseteq \Sigma^*$ and let $M = (k,Q,\Sigma,q_0,P,F)$ be a k-counter machine in the first normal form accepting L. We construct a generalized communicating P automaton $\Pi = (V,E,c_0,P',\{5\},\mathcal{F})$ and a mapping $f : V^* \to \Sigma^*$, such that $L(M) = L(\Pi,f)$.

Let $V = E \cup \{q,q_r \mid q \in Q, r \in P\}$ where $E = \Sigma \cup \{Z\} \cup \{C_i \mid 1 \leq i \leq k\}$, and let $c_0 = (w_0,w_1,\ldots,w_7)$ with $w_0 = w_3 = w_5 = w_6 = w_7 = w_8 = \emptyset$, $w_1 = \{q_0\}$, $w_2 = Q \setminus \{q_0\}$, and $w_4 = \{q_r \mid q \in Q, r \in P\}$.

The rule set P' of Π is defined as follows. For any $q \in Q$, the rule

$$\bar{r}_q : (q,4)(Z,0) \to (q,1)(Z,1) \tag{10.1}$$

is added to P', and we also have $(Z,0)(Z,1) \to (Z,8)(Z,8)$.

Furthermore, for any $r : (\varepsilon,p,C(i){+};q) \in P$ we have in P'

$$r_1 : (p,1)(p_r,4) \to (p,2)(p_r,2),\ r_2 : (p_r,2)(C_i,0) \to (p_r,3)(C_i,3),$$
$$r_3 : (p_r,3)(q,2) \to (p_r,4)(q,4).$$

For any $r : (a,p,C(i){-};q,s) \in P$, $a \neq \varepsilon$, we have in P'

$$\begin{array}{ll} r_4 : (p,1)(p_r,4) \to (p,8)(p_r,8), & r_5 : (p_r,8)(a,0) \to (p_r,5)(a,5), \\ r_6 : (p,8)(Z,0) \to (p,5)(Z,5), & r_7 : (p,5)(Z,0) \to (p,6)(Z,6), \\ r_8 : (p_r,5)(C_i,3) \to (p_r,0)(C_i,0), & r_9 : (p,6)(p_r,5) \to (p,3)(p_r,3), \\ r_{10} : (p,6)(p_r,0) \to (p,7)(p_r,7), & r_{11} : (p_r,3)(s,2) \to (p_r,4)(s,4), \\ r_{12} : (p_r,7)(q,2) \to (p_r,4)(q,4), & r_{13} : (p,3)(Z,0) \to (p,2)(Z,2), \\ r_{14} : (p,7)(Z,0) \to (p,2)(Z,2). & \end{array}$$

For any $r : (\varepsilon,p,C(i){-};q,s) \in P$, we have in P' instead of $r_4 - r_6$ the single rule

$$r_{15} \ : \ (p,1)(p_r,4) \to (p,5)(p_r,5),$$

and the rest of the rules, $r_7 - r_{14}$, are the same as above.

We prove that Π simulates M. The application of a rule of M is simulated by several steps of P'. The symbol representing the current state of the k-counter machine is in cell 1, while the values of the counters are stored in cell 3 in the form of symbols C_i where i refers to the ith counter, $1 \leq i \leq k$.

The application of rule $r : (\varepsilon, p, C(i)+; q) \in P$ is simulated in P' as follows: First, symbol p moves from cell 1 and meantime p_r leaves cell 4 and moves to cell 2 (rule r_1). Notice the role of p_r, this symbol identifies the rule to be simulated. Then, p_r brings from the environment a symbol C_i into cell 2 and both move to cell 3 (rule r_2). In the next step, p_r in cell 3 and q in cell 2, both move to cell 4, where after arrival q brings one symbol Z from the environment and both move to cell 1 (rule r_3 and rule $\bar{r}_q$). The obtained configuration of Π corresponds to the new configuration of M, i.e., changing its state p to q without reading any symbol and increasing the number of symbols in counter i by one. Due to the construction of the rule set of P' no other rules can be applied during this phase of generation and the rules can only be applied in this order.

Next we show how $r : (a, p, C(i)-; q, s) \in P,\ a \neq \varepsilon$ is simulated by rules in P'. First, symbols p and p_r leave cells 1 and 4, respectively and both move to cell 8 (rule r_4). Then p_r brings a symbol a from the environment and both move to cell 5 (rule r_5). At this point, the simulation of reading of letter a is finished. In parallel with this rule application, symbol p brings a symbol Z from the environment and both enter cell 5 (rule 6). Then, p brings one Z from the environment and the two symbols together are forwarded to cell 6 (rule r_7). There are two cases:

At the first case, there exists a symbol C_i in cell 3, then both this symbol and symbol p_r move to the environment (rule r_8), and in the next step p from cell 6 and p_r from the environment move together to cell 7 (rule r_{10}). The procedure continues by applying rules r_{12}, forwarding p_r to cell 4 and q from cell 2 to cell 4, then by applying rule $\bar{r}_q$, i.e., bringing one Z in from the environment and moving q and this Z together to cell 1. In parallel with the application of $\bar{r}_q$, p from cell 7 and a Z from the environment move to cell 2 (rule r_{14}). In this way, the simulation of the change of the configuration, when M in state p reads letter a, changes its state to q, and decrements the value of counter i by one is finished. Due to the definition of rules of P', during this phase of the generation no other rule is applicable and the application of no other rule of M can be simulated.

At the second case, there is no symbol C_i in cell 3, which implies that rule r_8 cannot be applied, so p_r remains in cell 5 during the computational step when p moves from cell 5 to cell 6. Then, the only applicable rule is r_9 which moves p from cell 6 and p_r from cell 5 to cell 3. From cell 3, p_r moves to cell 4, simultaneously with s, that leaves cell 2 and moves to cell 4. Then applying rule $\bar{r}_s$, s from cell 4 moves to cell 1 together with a symbol Z brought in from the environment, and meantime p from cell 3 and a symbol Z from the environment move to cell 2. This way that configuration change of M is simulated when state p changes to s, due to that the value of counter i is zero, i.e., it cannot be decremented. It can be seen that during this phase of the generation the application of no other rule of M can be simulated and the rules can be applied only in the above order.

Next we show the simulation of a rule $r : (\varepsilon, p, C(i)-; q, s) \in P$ by rules of Π. The procedure is almost the same as the previous one. Since M does not read any symbol, instead of using rules $r_4 - r_6$, we use rule r_{15} which moves symbol p from cell 1 and symbol p_r from cell 4 to cell 5, and then we use rules of the form $r_7 - r_{14}$ in the same way as above.

Due to the construction and functioning of Π no other instruction than that M has and able to perform can be simulated. Thus, if we have $\mathcal{F} = \{(\emptyset, q, \#, \ldots, \#) \mid q \in F\}$ as the set of accepting configurations, and define $f : V^* \to \Sigma^*$ with $f(aZ) = a$, then $L(\Pi, f) = L(M)$. □

Now we examine generalized communicating P automata with *conditional-uniport-in* rules. To show that these systems also generate any recursively enumerable language, we need a bit more involved construction.

Theorem 7. *$\mathcal{L}(PA_8, uin) = RE$.*

Proof. Let $L \subseteq \Sigma^*$ and let M be a k-counter machine in the second normal form accepting L. Before we start the construction of a generalized communicating P automaton simulating M, we first show how to construct a system with four regions in such a way that there are three symbols $A_i,\ i = 0, 1, 2$, appearing in the first region after the $(3 \cdot n + i)$th steps of the computation for each $n \geq 0$, in such a way that they are "free" in the next step, that is, there is no rule which can be applied to the symbol A_i present in the first region in the $(3 \cdot n + i + 1)$th step of the system.

Consider the alphabet $\{A_j, c_j, d_j, e_j \mid 0 \leq j \leq 2\} \cup \{f, g, T\}$ and a system with four membranes having an initial configuration

$$(A_0A_2c_0c_1c_2,\ d_1d_2A_1e_0e_1e_2,\ fd_0,\ g) \tag{10.2}$$

and the following rules. First, for each $j,\ 0 \leq j \leq 2$, let us have

$$\begin{array}{ll} r_{1,j} : (c_j,1)(A_j,2) \to (c_j,1)(A_j,1), & r_{2,j} : (d_j,2)(A_j,1) \to (d_j,2)(A_j,2), \\ r_{3,j} : (e_j,2)(d_j,3) \to (e_j,2)(d_j,2), & r_{4,j} : (f,3)(d_j,2) \to (f,3)(d_j,3), \\ r_{5,j} : (g,4)(d_j,2) \to (g,4)(d_j,4), & r_{6,j} : (d_j,4)(T,0) \to (d_j,4)(T,4). \end{array}$$

To see how these rules work, consider $n = j = 0$ and the initial configuration shown in (10.2). Observe that $A_j = A_0$ is present in region 1, and there is no rule to be applied to it in the first step because there is no d_0 present in the second region (see rule $r_{2,0}$). After the first step we get the following sequence of configurations:

$$(A_0A_2c_0c_1c_2,\ d_1d_2A_1e_0e_1e_2,\ fd_0,\ g) \Rightarrow$$

$$(A_1A_0c_0c_1c_2,\ d_2d_0A_2e_0e_1e_2,\ fd_1,\ g) \Rightarrow \tag{10.3}$$

$$(A_2A_1c_0c_1c_2,\ d_0d_1A_0e_0e_1e_2,\ fd_2,\ g) \Rightarrow \tag{10.4}$$

$$(A_0A_2c_0c_1c_2,\ d_1d_2A_1e_0e_1e_2,\ fd_0,\ g) \Rightarrow \tag{10.5}$$

Note that there is no rule to be applied to A_0 in the first step. Then, after the first step, in (10.3), A_1 has appeared in region 1 and there is no rule to be applied to it in the second step. After the second step, in (10.4), A_2 appears in region 1 and there is no rule to be applied to it in the third step, resulting in (10.5) where the state of the system returns to its initial configuration given in (10.2), and the whole process can start again.

If we have these four membranes as part of a greater system, A_j in region one could possibly be "used" in steps $3 \cdot n + j + 1$, $n \geq 0$, by other rules in the additional part of the greater system without having any influence on the above described processing cycle. If, however, any of the A_j symbols would be used by the additional rules of the additional part of the system in a computational step different from those given above, then one of the d_j symbols, $0 \leq j \leq 2$, would have to be moved to region 4 by the rules $r_{5,j}$, and then the rules $r_{6,j}$ would prevent the halting of the computation of the system.

For the construction of the generalized communicating P automaton simulating the k-counter machine M, we will extend the above system in such a way that the symbols A_j, $j = 0,1,2$, appearing in the first region will only be "used" when they are "free", that is, in the computational step directly after their appearance. This way the three steps cycle described above can continue to occur during the whole computation of the system.

Let $M = (k,Q,\Sigma,q_0,P,F)$, and let us denote for a rule $r \in P$ of one of the forms $(\varepsilon,p,(i,>0);q,-1)$, $(\varepsilon,p,(i,=0);q,0)$, $(x,p,(i,>0);q,+1)$, or $(x,p,(i,=0);q,+1)$, $x \in \Sigma \cup \{\varepsilon\}$, $p,q \in Q$, $i \in \{1,\ldots,k\}$, the state p as $State(r) \in Q$ and the state q as $NextState(r) \in Q$. Let us also define for any $r \in P$ the set $NextRule(r) \subseteq P$ as follows:

$$NextRule(r) = \{r' \in P \mid NextState(r) = State(r')\}.$$

Now we construct $\Pi = (V,E,c_0,P',\{5\},\mathcal{F})$, a generalized communicating P automaton such that $L(M) = L(\Pi,f)$ for the mapping $f : V^* \to V^*$ with $f(a) = a$, $a \in V$. To construct Π, we start with a system of eight regions which is constructed according to the ideas above in such a way, that the symbols $A_{i,0}$, $A_{i,1}$, $A_{i,2}$ appear periodically in each region $1 \leq i \leq 8$, and can be "used" by other rules in the $3 \cdot n + 1$, $3 \cdot n + 2$, and $3 \cdot n$ step of the computation, respectively, for $n \geq 0$, in the same way as described above. Let V_{cycl} and P_{cycl} denote the set of rules and the set of symbols used by them, similar to those listed above (with the difference that instead of A_j, $0 \leq j \leq 2$, we have $A_{i,j}$ for each $1 \leq i \leq 8$), and let $w_{cycl,i}$, $1 \leq i \leq 8$ denote the initial membrane contents of objects from V_{cycl} necessary for their functioning.

Let $V = V_{cycl} \cup E \cup \{r'_0\} \cup \{r, r' \mid r \in lab(P)\}$ where $E = \Sigma \cup \{Z,D,X\} \cup \{C_i \mid 1 \leq i \leq k\}$ and let the initial configuration be

$$c_0 = (w_0,\ldots,w_8)$$

with $w_1 = w_{cycl,1} \cup \{r'_0\}$, $w_2 = w_{cycl,2} \cup \{D\}$, $w_3 = w_{cycl,3} \cup \{r \mid r \in lab(P)\}$, $w_4 = w_{cycl,4} \cup \{r' \mid r \in lab(P)\}$, $w_i = w_{cycl,i}$ for $5 \leq i \leq 7$, and $w_8 = w_{cycl,8} \cup \{X\}$.

The rule set P' is defined as follows. Besides the rules in P_{cycl}, for all rules $r \in P$ with $State(r) = q_0$, we add to P' the rules

$$(r'_0,1)(r,3) \to (r'_0,1)(r,1), \text{ and } (A_{4,1},4)(r'_0,1) \to (A_{4,1},4)(r'_0,4),$$

and for all rules $r, p \in P$, such that $r \in \mathit{NextRule}(p)$, we add to P' the rules

$$(p',1)(r,3) \to (p',1)(r,1), \text{ and } (A_{1,2},1)(r',5) \to (A_{1,2},1)(r',1).$$

Furthermore, for any $r \in P$ where $r : (x, p, (i, > 0); q, +1)$ or $r : (x, p, (i, = 0); q, +1)$ for some $p, q \in Q$, $i \in \{1, \ldots, k\}$, and $x = \varepsilon$, we have in P'

$$\begin{array}{ll}
r_1 : (A_{6,1},6)(r,1) \to (A_{6,1},6)(r,6), & r_2 : (D,2)(r,6) \to (D,2)(r,2), \\
r_3 : (r,2)(C_i,0) \to (r,2)(C_i,2), & r_4 : (A_{4,1},4)(r',1) \to (A_{4,1},4)(r',4), \\
r_5 : (A_{5,2},5)(r',4) \to (A_{5,2},5)(r',5), & r_6 : (r',5)(r,2) \to (r',5)(r,5), \\
r_7 : (A_{3,2},3)(r,5) \to (A_{3,2},3)(r,3), & r_8 : (X,8)(r,2) \to (X,8)(r,8), \\
r_9 : (r,8)(Z,0) \to (r,8)(Z,8), &
\end{array}$$

and if $x = a \in \Sigma$ $(x \neq \varepsilon)$, we also have $r_{6'} : (r,5)(a,0) \to (r,5)(a,5)$.

In addition, for any $r : (\varepsilon, p, (i, > 0); q, -1) \in P$, we have in P' the rules

$$\begin{array}{ll}
r_{10} : (A_{7,1},7)(r,1) \to (A_{7,1},7)(r,7), & r_{11} : (A_{8,1},8)(C_i,2) \to (A_{8,1},8)(C_i,8), \\
r_{12} : (r,7)(C_i,8) \to (r,7)(C_i,7), & r_{13} : (A_{4,0},4)(r,7) \to (A_{4,0},4)(r,4), \\
r_{14} : (A_{3,2},3)(r',4) \to (A_{3,2},3)(r',3), & r_{15} : (r',3)(r,4) \to (r',3)(r,3), \\
r_{16} : (A_{1,2},1)(r',3) \to (A_{1,2},1)(r',1), & r_{17} : (A_{5,2},5)(r,7) \to (A_{5,2},5)(r,5), \\
r_{18} : (r,5)(Z,0) \to (r,5)(Z,5), & r_{19} : (Z,5)(Z,0) \to (Z,5)(Z,5), \\
r_{20} : (A_{2,0},2)(C_i,8) \to (A_{2,0},2)(C_i,2), & r_{21} : (A_{8,1},8)(C_i,2) \to (A_{8,1},8)(C_i,8), \\
r_{22} : (X,8)(r,4) \to (X,8)(r,8), & r_{23} : (r,8)(Z,0) \to (r,8)(Z,8),
\end{array}$$

and for any $r : (\varepsilon, p, (i, = 0); q, 0) \in P$, we have in P' the following (for the sake of easier readability, we enumerate again also those rules, which do not depend on r, thus, which we already presented above)

$$\begin{array}{ll}
r_{24} : (A_{6,1},6)(r,1) \to (A_{6,1},6)(r,6), & r_{25} : (A_{8,1},8)(C_i,2) \to (A_{8,1},8)(C_i,8), \\
r_{26} : (r,6)(C_i,8) \to (r,6)(C_i,6), & r_{27} : (A_{4,0},4)(r,6) \to (A_{4,0},4)(r,4), \\
r_{28} : (A_{3,2},3)(r',4) \to (A_{3,2},3)(r',3), & r_{29} : (r',3)(r,4) \to (r',3)(r,3), \\
r_{30} : (A_{1,2},1)(r',3) \to (A_{1,2},1)(r',1), & r_{31} : (C_i,6)(Z,0) \to (C_i,6)(Z,6), \\
r_{32} : (Z,6)(Z,0) \to (Z,6)(Z,6), & r_{33} : (A_{8,1},8)(C_i,2) \to (A_{8,1},8)(C_i,8), \\
r_{34} : (X,8)(r,4) \to (X,8)(r,8), & r_{35} : (r,8)(Z,0) \to (r,8)(Z,8).
\end{array}$$

To see how this system simulates the computations of the k-counter machine M, consider the initial configuration where each region i, $1 \leq i \leq 8$ contains $A_{i,0}$, the first region r'_0, the third and the fourth regions objects r and r', respectively, for all $r \in P$, and the eighth region the symbol X (besides the objects in $w_{cycl,i}$ which we ignore in the following for the sake of simplicity). First r'_0 moves a symbol r corresponding to a rule $r \in P$ with $\mathit{State}(r) = q_0$ to the first region with the rule $(r'_0,1)(r,3) \to (r'_0,1)(r,1)$, then the system simulates r.

If r is an increment rule $r : (x, p, (i, > 0); q, +1)$ or $r : (x, p, (i, = 0); q, +1)$, Π uses the rules $r_1, \ldots, r_9$ to import an object C_i to region 2 (the multiplicity of this object corresponds to the value stored in the ith counter of M), and if a $x = a \in \Sigma$, then rule $r_{6'}$ reads the corresponding input symbol from the environment. After two

$A_{i,0},\ldots,A_{i,2}$ cycles (after 6 steps), r' appears in region one, the rule $(r',1)(s,3) \rightarrow (r',1)(s,1)$ chooses the next instruction $s \in P$ to be simulated. The simulation of the increment instruction is nondeterministic, thus, it might be possible that Π makes a "wrong" nondeterministic choice, in which case the object r is moved to region 8 and the rule $r_9 : (r,8)(Z,0) \rightarrow (r,8)(Z,8)$ makes sure that the computation can never reach a halting configuration.

The simulation of a decrement instruction $r : (\varepsilon, p, (i, > 0); q, -1) \in P$ is done with the rules $r_{10},\ldots,r_{23}$. These rules remove one C_i symbol from region 2, or if there were no C_i present, then r is moved either to region 5 or 8, in which cases the computation may never reach a halting configuration.

The zero check instruction is simulated in a similar way by the rules $r_{24},\ldots,r_{35}$. If the number of C_i in region 2 is not zero, then C_i is moved to region 6, and the computation of Π can never halt, otherwise, as in the previous cases, r' appears in region one after 6 steps, and the simulation of the next instruction can begin. □

10.4 Conclusions

As we have seen, special variants of generalized communicating P automata are able to recognize any recursively enumerable language. Interesting questions are how much extent the size parameters of these P automata variants can be reduced to obtain the same computational power or, what kind of restrictions can be/should be introduced on rules of generalized communicating P automata to describe various language classes. Obviously, the power of generalized communicating P automata with only rules not discussed in the paper is a topic for future research as well.

Acknowledgements. Research supported in part by the Hungarian Scientific Research Fund, "OTKA", grant no. K75952, and by the European Union through the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project which is co-financed by the European Social Fund.

References

1. Csuhaj-Varjú, E., Oswald, M., Vaszil, Gy.: P automata. In: [8], ch. 6, pp. 144–167 (2010)
2. Csuhaj-Varjú, E., Vaszil, Gy.: P automata or purely communicating accepting P systems. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC 2002. LNCS, vol. 2597, pp. 219–233. Springer, Heidelberg (2003)
3. Csuhaj-Varjú, E., Vaszil, Gy., Verlan, S.: On Generalized Communicating P Systems with One Symbol. In: Gheorghe, M., Hinze, T., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) CMC 2010. LNCS, vol. 6501, pp. 160–174. Springer, Heidelberg (2010)
4. Csuhaj-Varjú, E., Verlan, S.: On generalized communicating P systems with minimal interaction rules. Theoretical Computer Science 412, 124–135 (2011)
5. Fischer, P.C.: Turing machines with restricted memory access. Information and Control 9, 364–379 (1966)
6. Krishna, S.N., Gheorghe, M., Dragomir, C.: Some classes of generalised communicating P systems and simple kernel P systems. In: Bonizzoni, P., Brattka, V., Löwe, B. (eds.) CiE 2013. LNCS, vol. 7921, pp. 284–293. Springer, Heidelberg (2013)

7. Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000)
8. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
9. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Berlin (1997)
10. Salomaa, A.: Formal Languages. Academic Press, New York (1973)
11. Verlan, S., Bernardini, F., Gheorghe, M., Margenstern, M.: Generalized communicating P systems. Theoretical Computer Science 404, 170–184 (2008)

Chapter 11
Computational Models Based on Splicing

Yurii Rogozhin and Sergey Verlan

Abstract. In this paper we overview twelve different computational models that use the splicing operation. We explain the methods used for the organization of the computational process in this area and give examples for each considered model.

11.1 Introduction

A splicing system is a formal model of the recombinant behavior of DNA molecules under the influence of restriction enzymes and ligases. This model introduced by T. Head in [8] relies on the splicing operation which cuts two strings at specific points and then recombines them crosswise. The basic model of an *H system* consists of a set splicing rules which are iterated over initial strings (axioms), yielding a formal language. It was shown that in the case of a finite set of rules and axioms the generated language is strictly included in the family of regular languages (REG). Numerous extensions of the basic variant were done in order to extend the computational power of splicing systems. In most of the cases these models are inspired by formal models from regulated rewriting and grammar systems area and they permit a strict increase in the computational power, often till the computational completeness. An important particularity of all these variants is the fact that the splicing is a binary operation, unlike the rewriting. This permits reacher possibilities for the organization of the computation, which are different from their classical (rewriting) counterparts.

Yurii Rogozhin
Institute of Mathematics and Computer Science, Academy of Sciences of Moldova,
Str. Academiei 5, Chişinău, MD-2028, Moldova
e-mail: `rogozhin@math.md`

Sergey Verlan
LACL, Département Informatique, Université Paris Est, 61,
av. Général de Gaulle, 94010 Créteil, France
e-mail: `verlan@u-pec.fr`

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_11

In this paper we overview several computational models based on splicing. We consider corresponding definitions and discuss the particularities of each variant. For this purpose we present the generation of the language $L_{ab} = \{a^n b^n, n \geq 1\}$ for every considered model and also give ideas how to reach the computational completeness.

The paper is organized as follows. Section 11.2 contains the definitions and some key results for the basic model of splicing – H systems. It also presents two possible modifications of the standard definition that use the ingredients in a different manner: iterated splicing systems and multiset splicing systems, both having Turing machines' power. Section 11.3 considers the enhancement of H systems with control mechanisms from regulated rewriting: permitting/forbidding context, order (priority) between rules, matrix, programmed, graph and time varying control. Section 11.4 considers distributed splicing systems, i.e. systems having locations where strings evolve locally and after that are redistributed across the system. We consider splicing test tube systems, length-separating test tube systems, splicing P systems and networks of splicing processors. Section 11.5 presents an original method of refining the control for a splicing system that is powerful enough to support the necessary operations. This method is based on the fact that the splicing is a binary operation, hence in order for a rule to be applied both stings should be present at the same moment in the system, so varying the appearance of one of these words the applicability of the rule is also modified. Finally, Section 11.6 presents some concluding remarks and open questions.

11.2 Splicing Operation and H Systems

In this paper we consider the Păun definition of splicing operation, which is widely used in the area. For more details on the other definitions of splicing as well as for the biological motivation we refer to [10,22,32].

A *splicing rule* (over an alphabet V) is a 4-tuple (u_1, u_2, u_3, u_4) where u_1, u_2, u_3, $u_4 \in V^*$. It is frequently written as $u_1\#u_2\$u_3\#u_4$, $\{\$,\#\} \not\subseteq V$, or in two dimensions as $\dfrac{u_1 \,|\, u_2}{u_3 \,|\, u_4}$. Strings u_1u_2 and u_3u_4 are called splicing *sites*.

We say that a word x *matches* rule r if x contains an occurrence of one of the two sites of r. We also say that x and y are *complementary* with respect to a rule r if x contains one site of r and y contains the other one. In this case we also say that x or y may *enter* rule r. When x and y can enter a rule $r = u_1\#u_2\$u_3\#u_4$, i.e., we have $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$, it is possible to define the application of r to couple x, y. The result of this application is w and z where $w = x_1u_1u_4y_2$ and $z = y_1u_3u_2x_2$. We also say that x and y are spliced and w and z are the result of this splicing. We write this as follows: $(x,y) \vdash_r (w,z)$ or $(x_1u_1|u_2x_2, y_1u_3|u_4y_2) \vdash_r (x_1u_1u_4y_2, y_1u_3u_2x_2)$.

The pair $\sigma = (V,R)$ where V is an alphabet and R is a finite set of splicing rules is called a *splicing scheme* or an *H scheme*. The diameter of a splicing rule $x\#y\$w\#z$ is defined as $(|x|,|y|,|w|,|z|)$. For a splicing scheme $\sigma = (V,R)$ we define the diameter of σ as (n_1,n_2,n_3,n_4), where $n_i = \max_{x_1\#x_2\$x_3\#x_4 \in R} x_i$, $1 \leq i \leq 4$.

For a splicing scheme $\sigma = (V,R)$ and for a language $L \subseteq V^*$ we define:
$\sigma(L) \overset{\text{def}}{=} \{w,z \in V^* \mid \exists x,y \in L, \exists r \in R : (x,y) \vdash_r (w,z)\}$.

The splicing operation as defined above is called 2-splicing. If only w is kept as the result, the corresponding formalism is called 1-splicing. In this paper we consider 2-splicing operation only and we refer to [30, 32] for more details.

Now we can introduce the closure of a language under splicing with respect to σ:
$\sigma^0(L) = L$,
$\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L))$, $i \geq 0$,
$\sigma^*(L) = \cup_{i \geq 0} \sigma^i(L)$.

We remark that in the above definition $\sigma^i(L)$ is not equivalent to the iteration of the σ operator. In Section 11.2.1 we consider the iterative variant of σ.

It is known that σ^* preserves the regularity of a language:

Theorem 6. [22] *Let $L \subseteq V^*$ be a regular language and let $\sigma = (V,R)$ be a splicing scheme. Then language $\sigma^*(L)$ is regular.*

A *Head-splicing-system* [8, 9], or *H system*, is a construct:

$$\mathcal{H} = (\sigma, A) = ((V,R), A),$$

which consists of an alphabet V, a finite set $A \subseteq V^*$ of initial words over V, the *axioms*, and a finite set $R \subseteq V^* \times V^* \times V^* \times V^*$ of splicing rules. System $\mathcal{H}$ is called finite if A and R are finite sets.

The language generated by an H system $\mathcal{H}$ is defined as $L(\mathcal{H}) \overset{\text{def}}{=} \sigma^*(A)$.

Thus, the language generated by H system $\mathcal{H}$ is the set of all words that can be generated in $\mathcal{H}$ starting with A as initial words by iteratively applying splicing rules to copies of the words already generated. The family of languages generated by H systems with a finite set of axioms and rules is denoted by $H(FIN, FIN)$.

Theorem 7. *The following relations hold:*

(1) For any H system $\mathcal{H}$, $L(\mathcal{H}) \in REG$.
(2) For any $L \in REG$ there exists an H system $\mathcal{H}$ and an alphabet T such that $L = L(\mathcal{H}) \cap T^*$.

The precise characterization of the family $H(FIN, FIN)$ is a long-standing open problem in the area of splicing systems. There are several partial results for the cases when the ruleset is restricted, *e.g.* reflexive. We refer to [32] for more information. A recent result from [3] shows a necessary condition for a regular language L to be a splicing language: L must have a constant in the Schützenbergers sense. In [11] it is shown that it is possible to decide if a regular language is a splicing language.

11.2.1 Iterated Splicing

In this subsection we consider a different definition of H systems, where iterated splicing instead of the closure is used. Let $\sigma = (V,R)$ be a splicing scheme. We define $\bar{\sigma}$ as the iteration of σ in an ordinary sense:

$\bar{\sigma}^*(L) = \cup_{i\geq 0}\bar{\sigma}^i(L)$, where $(\bar{\sigma}^i(L) = \underbrace{\sigma(\sigma(\dots\sigma}_{i}(L)\dots))$.

For an H system $\mathcal{H} = (\sigma,A)$ we denote by $\bar{L}(\mathcal{H}) = \bar{\sigma}^*(A)$.

We remark that the operation $\bar{\sigma}^*$ is different from σ^* as it keeps only the resulting strings of each iteration available for the next iteration. This provides a powerful feature that permits to eliminate all strings that are not produced by a splicing operation at the corresponding step. It is somehow surprising that this modification suffices by itself to achieve the computational completeness:

Theorem 8. *[16, 17] For any RE language $\mathcal{L} \subseteq T^*$ there is an H system $\mathcal{H}$, such that $\bar{L}(\mathcal{H}) \cap T^* = \mathcal{L}$.*

Moreover, as it was shown in [1] there exists a universal iterated splicing H system having only 17 splicing rules.

The study of the operation $\bar{\sigma}^*$ is motivated by the study of time-varying distributed H (TVDH) systems, see Section 11.3.6, because it is identical to TVDH systems with one component. In Section 11.5 we consider some more examples on iterated splicing.

11.2.2 Multisets

Another idea used to increase the computational power of H systems is to use multisets of strings instead of sets of strings. In this case the application of a splicing consumes one copy of each input string and produces one copy of the output strings. We refer to [22] for the corresponding formal definitions and we remark that in such systems it is possible to eliminate strings by consuming them. H systems with multisets are shown to be computationally complete [22].

Example 1. We present now a multiset splicing system generating the language $L_{ab} = \{a^nb^n, n \geq 1\}$. Let $\mathcal{H} = (V,T,A,R)$, where $V = \{a,b,c,d,Y\}$, $T = \{a,b\}$, $A = \{ab, caabbd, caZd\} \cup \{caY, Ybd\}^\infty$ and let set R be defined as follows:

$$\begin{array}{ll} 1: c\#aa\$ca\#Zd & 2: cZ\#d\$Y\#bd \\ 3: bb\#d\$cZ\#bd & 4: c\#Zd\$ca\#Y \\ 5: c\#aa\$\varepsilon\#caZd & 6: bb\#d\$ccaZd\#\varepsilon \end{array}$$

It is clear that the evolution of $\mathcal{H}$ can be done either using the sequence of rules $1-4$, repeatedly, or the sequence of rules $5,6$, once. This is done by considering the second word that participate in splicing in one copy and by using its product as the second string at the next step.

11.3 Controlled Splicing

The result from Theorem 6 states that a splicing closure of a finite set of strings is regular. Making analogy to the string rewriting it is possible to consider control mechanisms for splicing like it is done in the area of regulated rewriting. In this section we will consider counterparts for random-context, ordered, matrix, programmed, graph-controlled and time-varying grammars. Like in the case of rewriting these control mechanisms permit to strictly increase the computational power of corresponding variants. We remark that we give only the main ideas for each definition, full details can be consulted in [22].

11.3.1 Main Example

In the following we will consider different variants of systems based on splicing. In order to better see their particularities we will consider a single example that will be adapted for each type of system.

We will consider the generation of the language $L_{ab} = \{a^n b^n, n \geq 1\}$. Consider following splicing rules that permit to generate it.

$$\begin{array}{ll} 1 : c\#a\$c'a\#Z & 2 : b\#d\$Z\#bd' \\ 3 : c'\#a\$c\#Z & 4 : b\#d'\$Z\#d \\ 5 : c\#a\$\varepsilon\#Z' & 6 : b\#d\$Z'\#\varepsilon \end{array}$$

Starting from a word $cabd$ the successive application of rules 1-4 permits to increase at the same time the number of symbols a and b. The application of the last two rules permits to eliminate leading symbol c and trailing symbol d. While this is not the only possibility to generate L_{ab} using splicing rules, it is probably the simplest one. We remark that it is important to keep end markers c and d in the string during the first stage (increasing the number of a's and b's), otherwise there is no guarantee that the splicing happens at the ends of the string. At the second stage it is important prohibit the application of rules 1-4, otherwise the number of a's or b's can be increased in an uncontrolled manner. We also note that strings $c'aZ$, cZ, Zbd', Zd and Z' should be available for splicing at corresponding time moments.

Now in order to perform a correct generation it should be ensured by the control of the splicing system that the rules 1-4 are applied (in a loop) one after another, as well as rules 5 and 6. A special attention should be done to the results of the splicing of last two rules, as new words cZ' and $Z'd$ can be produced and these words can be further eligible for rules 5 and 6.

11.3.2 Rotate-and-Simulate Technique

The *rotate-and-simulate* technique is the basic tool for universality proofs in the area of splicing.

Let S be a formal system based on splicing simulating a formal grammar G. Then, for any string $w \in (N \cup T)^*$ of G there are strings $Xw''Bw'Y$ $(X,Y,B \notin N \cup T, w = w'w'')$ in S. These strings are called *rotational versions* of w. More precisely, the symbols X and Y flank w and the symbol B marks the beginning (or ending) of the string. It should be clear that w can be obtained from any of its rotational versions.

The system S simulates the formal grammar G as follows. For each production $u \to v$ of G the axiom ZvY and the splicing rule $\lambda\#uY\$Z\#vY$ are in S. The production of G is simulated as follows. The system S rotates the string Xw_1uw_2BY into Xw_2Bw_1uY and then it applies the corresponding splicing rule. Next, the resulting string Xw_2Bw_1vY is rotated into Xw_1vw_2BY. The rotation is made symbol by symbol, i.e., the string Xwa_iY, $a_i \in N \cup T \cup \{B\}$, is transformed into Xa_iwY after some computational steps.

The symbol by symbol rotation is often implemented as follows. The system starts with the string Xwa_iY. Later, this string becomes XwY_i, i.e., the presence of a_i at the right-hand end side is encoded by Y_i (it is also possible to use a unary encoding 1^iY). Next, the strings $X_ja_jwY_i$, $1 \le j \le n$, are generated, where n is the total number of symbols that may occur in the string. Afterwards, the system works in a cycle where indices i and j are decreased. If $j \neq i$, then the strings with these indices are removed. If $i = j$, the string $X_1a_iwY_1$ is obtained. Next, the string Xa_iwY is produced. Therefore the initial string Xwa_iY is rotated by one symbol.

We give below an example of rules that permit to perform the rotation procedure.

$$\begin{array}{ll}
1 : \varepsilon\#a_iY\$Z\#Y_i & 2 : X\#\varepsilon\$X_ja_j\#Z \\
3 : \varepsilon\#Y_i\$Z\#Y'_{i-1} & 4 : X_i\#\varepsilon\$X'_{i-1}\#Z \\
5 : \varepsilon\#Y'_i\$Z\#Y_i & 6 : X'_i\#\varepsilon\$X_i\#z \\
7 : \varepsilon\#Y'_0\$Z\#Y & 8 : X'_0\#\varepsilon\$X\#Z
\end{array}$$

The rules performed in the sequence $1-8$ yield a rotation of a symbol as described above. The needed axioms correspond to the second word in the splicing rule.

We also note that in the examples above the computation is performed in a specific way. At each step a main word enters a splicing that changes one of its ends. The second word used to perform the operation is predefined and is part of axioms. Using rotate-and-simulate method such a strategy is sufficient for obtaining the computational completeness. We remark that most of the proofs and examples for splicing-based systems have this property. As exception we can cite proofs using tag systems or Turing machine simulations that are based on different ideas [14,15].

11.3.3 Using Permitting/Forbidding Contexts

In this section we consider the adaptation of the idea of random-context control conditions to the area of splicing. We also consider ordered splicing systems as they are extremely close to forbidding conditions. Because splicing is more powerful than context-free rewriting it is sufficient to consider either permitting or forbidding conditions only.

We start with splicing systems with permitting context. Such systems consider rules of form $p = (r;C_1,C_2)$, where r is a splicing rule and $C_1,C_2 \subseteq V$ (we remark that in [22] C_1 and C_2 are finite languages over V. A rule p is applicable to the strings w_1 and w_2 if and only if (1) r is applicable to w_1 and w_2; (2) w_1 contains all letters from C_1 and (3) w_2 contains all letters from C_2. The language generating by such systems is defined as usual – the terminal words in the splicing closure of the set of axioms. The family of extended H systems with permitting contexts is denoted as $EH(FIN,pFIN)$.

Example 2. Consider $\mathcal{H} = (V,T,A,R)$, with $V = \{a,b,c,c',d,d',Z,Z'\}$, $T = \{a,b\}$, $A = \{cabd, c'aZ, Zbd', cZ, Zd, Z'\}$ and let R be defined as follows:

$$\begin{array}{ll} 1: (c\#a\$c'a\#Z;\{d\},\emptyset) & 2: (b\#d\$Z\#bd';\{c'\},\emptyset) \\ 3: (c'\#a\$c\#Z;\{d'\},\emptyset) & 4: (b\#d'\$Z\#d;\{c\},\emptyset) \\ 5: (c\#a\$\varepsilon\#Z';\emptyset,\emptyset) & 6: (b\#d\$Z'\#\varepsilon;\emptyset,\emptyset) \end{array}$$

It can be easily seen that $L(\mathcal{H}) = L_{ab}$. The sequencing of rules 1-4 is assured by corresponding permitting conditions that check a unique symbol at the ends of the string that is introduced by the previous rule. Once rules 5 and 6 are applied the above loop breaks and the only possibility to continue is to apply both rules 5 and 6. We also remark that additional words cZ' and $Z'd$ do not interfere with the correct line of the computation.

From the example above it should be clear how to construct longer sequences of rule applications, so using the rotate-and-simulate method it can be easily shown that $EH(FIN,pFIN) = RE$.

The definition of splicing systems with forbidding context is similar. The rules are of the form $p = (r;C_1,C_2)$, where r is a splicing rule and $C_1,C_2 \subseteq V$ (we remark that in [22] C_1 and C_2 are finite languages over V. A rule p is applicable to the strings w_1 and w_2 if and only if (1) r is applicable to w_1 and w_2; (2) w_1 does not contain any letter from C_1 and (3) w_2 does not contain any letter from C_2. The family of extended H systems with permitting contexts is denoted as $EH(FIN,fFIN)$.

Example 3. We give an H system with forbidding context $\mathcal{H} = (V,T,A,R)$ such that $L(H) = L_{ab}$. Let $V = \{a,b,c,c',c'',d,d',d'',Z,Z'\}$, $T = \{a,b\}$, $A = \{cabd, c'aZ, Zbd', cZ, Zd, c''Z, Zd'', Z'\}$ and the set R is defined as follows:

$$\begin{array}{ll}
1:(c\#a\$c'a\#Z;\{d'\},\emptyset) & 2:(b\#d\$Z\#bd';\{c,c''\},\emptyset)\\
3:(c'\#a\$c\#Z;\{d,d''\},\emptyset) & 4:(b\#d'\$Z\#d;\{c'\},\emptyset)\\
5a:(c\#a\$c''\#Z;\{d'\},\emptyset) & 6a:(b\#d\$Z\#d'';\{c,c'\},\emptyset)\\
5b:(c''\#a\$\varepsilon\#Z';\{d\},\emptyset) & 6b:(b\#d''\$Z'\#\varepsilon;\emptyset,\emptyset)
\end{array}$$

As in example above, the sequencing 1-4 is assured by the forbidding conditions that check that the unique symbol from the previous rule was removed. Due to the nature of forbidding conditions, the second stage becomes more complicated as it is impossible to enforce the presence of a symbol. For example, rule 1 will be applicable to the string *cabd* as well as to the string *cab*, yielding an increase in number of symbols a. To handle this case the end markers are changed to double primed versions, meaning that the simulation enters in stage 2, and after that are safely removed, as rules 1-4 are no more applicable.

As for permitting case it is not difficult to show that $EH(FIN, fFIN) = RE$.

Now we consider ordered H systems that have the control defined in terms of a partial order relation ($>$) between rules. A rule r is applicable to w_1 and w_2 if there is no other rule $r' > r$ such that w_1 or w_2 may enter this rule. The family of extended ordered H systems is denoted as $EH(FIN, ord)$.

Example 4. We construct a ordered H system $\mathcal{H} = (V, T, A, R, >)$ such that $L(\mathcal{H}) = L_{ab}$. Let $V = \{a, b, c, c', c'', d, d', d'', Z, Z'\}$, $T = \{a, b\}$, $A = \{cabd, c'aZ, Zbd', cZ, Zd, c''Z, Zd'', Z'\}$ and let R be defined as follows:

$$\begin{array}{llll}
1: c\#a\$c'a\#Z & 2: b\#d\$Z\#bd' & & \\
3: c'\#a\$c\#Z & 4: b\#d'\$Z\#d & & \\
5a: c\#a\$c''\#Z & 6a: b\#d\$Z\#d'' & & \\
5b: c''\#a\$c''\#Z & 6b: b\#d''\$Z\#d'' & 5c: c''\#a\$\varepsilon\#Z' & 6c: b\#d''\$Z'\#\varepsilon
\end{array}$$

The relation $>$ is defined as follows (we denote by $a > b > c$ two relations $a > b$ and $b > c$): $1 > 2 > 3 > 4$, $5a > 6a > 5c > 6c$, $5b > 2$, $6b > 3$.

The strategy is to keep the main loop using the order relation and to perform the second stage as in the forbidding case. The choice to switch to between stages is validated by dummy rules $5b$ and $5c$ that permit to inhibit the action of rules 2 and 3 if one of rules $5a$ or $6a$ was applied.

The priority control is similar to forbidding conditions, so we can easily obtain that $EH(FIN, ord) = RE$.

11.3.4 Double Splicing

In this section we consider the counterpart of matrix grammars for splicing systems. However, instead of sequences of prescribed rules, we consider the double splicing operation, which is composed of two splicings and requires that the result of the first splicing to be the input of the second one. More precisely, we define

$$(x,y) \vdash_{r_1,r_2} (w,z) \text{ iff } (x,y) \vdash_{r_1} (u,v) \text{ and } (u,v) \vdash_{r_2} (w,z).$$

We remark that in most cases (and in particular for universality proofs) it is possible to take a word y of special form that guarantees that r_1 and r_2 can be used in this sequence only, hence the relation to matrix sequences can be immediately observed. We note that it could still be interesting to investigate matrix splicing systems based on prescribed sequences of rules, as this permits to simplify some proofs. The family of double splicing H systems is denoted as $EH(FIN,d)$.

Example 5. Consider the following double splicing H system $\mathcal{H} = (V,T,A,R)$ with $V = \{a,b,c,d,Z,Z'\}$, $T = \{a,b\}$, $A = \{cabd, caZbd, Z'\}$ and let R be defined as follows:

$$1: c\#a\$ca\#Z \qquad\qquad 2: b\#d\$Z\#bd$$

$$3: c\#a\$\varepsilon\#Z' \qquad\qquad 4: b\#d\$Z'\#\varepsilon$$

It can be easily seen that $L(\mathcal{H}) = L_{ab}$. We remark that comparing to examples above in the case of the double splicing four rules suffice, because there is a guarantee that both rules from the sequence are applied. We observe that the first axiom determines a consecutive application of rules 1 and 2, while the second axiom determines the sequence 3 and 4.

Since by applying two rules it is possible to encode a chain of applications we obtain that $EH(FIN,d) = RE$. We also would like to highlight the result from [2] giving a construction of a universal double splicing H system having 5 splicing rules only.

11.3.5 Programmed/Graph Controlled H Systems

In this section we consider the adaptation of the idea of programmed/graph-controlled grammars for splicing systems. We recall that the idea of the programmed/graph control is to associate with each rule r a set of next rules $NEXT(r)$ and choose one of the rules from this set as the next applicable rule. This property can be reformulated in terms of a control graph, where nodes are states and edges correspond to rules that could be applied being in that state. The derivation tracks the current state and after a rule application it switches to the state that is reached by following the corresponding edge. We remark that the variants where the initial state is fixed or not are equivalent. Below we give the definition in the graph-controlled terms, the programmed version with the $NEXT$ mapping could be found in [22].

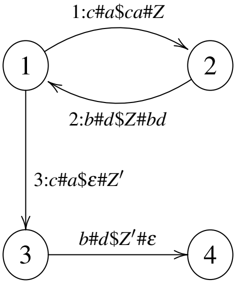
More precisely, for an H system $\mathcal{H} = (V,T,A,R)$ consider a control multigraph $G = (N,E)$ with edges labeled by splicing rules from R and denote by $lab(e)$, $e \in E$

the corresponding rule. Then the language generated by a graph-controlled splicing H system is defined as follows(let i_0 be the initial state):

$$L(H) = \{w \in T^* \mid (x_1,y_1) \vdash_{r_1} (x_2,y_2'),\ldots,(x_n,y_n) \vdash_{r_n} (w,y_{n+1}'),\\ \text{such that } y_i \in A, r_1 = lab(i_0,s), s \in N \text{ and}\\ \text{there is a path labeled by } r_1 \ldots r_n \text{ in } G\}.$$

We remark that at each step the second word from the splicing is an axiom. It could be interesting to consider a slightly different variant allowing the second word to be also produced by a sequence of splicing rules forming a path in G. The family languages generated by graph-controlled H systems is denoted as $EH(FIN,gc)$.

Example 6. Consider the following graph-controlled splicing H system $\mathcal{H} = (V,T,A,R,G,i_0)$ with $V = \{a,b,c,d,Z,Z'\}$, $A = \{cabd,caZ,Zbd,Z'\}$, $i_0 = 1$, $T = \{a,b\}$ and let G be defined as follows:



We remark that the set R can be deduced from G.

It can be easily seen that $L(H) = L_{ab}$. We observe that like in the case of double splicing there is no need to prime symbols c and d because the control of the system is implementing a strict sequencing of rules 1 and 2, or 3 and 4.

Clearly, with graph-controlled splicing systems it is possible to choose an arbitrary chain of rules, so we obtain that $EH(FIN,gc) = RE$.

11.3.6 (E)TVDH

In this section we consider the adaptation of the idea of time-varying grammars to the area of splicing. We recall that time-varying grammars can be seen as graph-controlled grammars where the corresponding control graph has the form of a ring. In the splicing case the definition is slightly different as one allows using words that are simultaneously obtained to perform a splicing instead of axioms. More precisely, for an H system $\mathcal{H} = (V,T,A,R)$ consider a partition of R in n (not necessary disjoint) subsets: $R = R_1,\ldots,R_n$. We call each R_i a component of $\mathcal{H}$ and n its degree. The language generated by $\mathcal{H}$ is defined as follows:

$$\begin{aligned}
&L_0(\mathcal{H}) = A\\
&L_{i+1}(\mathcal{H}) = \sigma_k(L_i(\mathcal{H})), \text{ where } \sigma_k = (V,R_k), \quad i = k \pmod n\\
&L(\mathcal{H}) = \cup_{i\geq 0} L_i(\mathcal{H}) \cap T^*
\end{aligned}$$

In the literature there are also considered enhanced time-varying distributed H (ETVDH) systems, which differ from TVDH by the fact that σ_k^* is used at each iteration instead of σ_k.

The family of languages generated by (E)TVDH systems of degree n is denoted as $(E)VDH_n$.

Example 7. Consider the ETVDH system of degree 4 $\mathcal{H}$ defined as follows: $\mathcal{H} = (V,T,A,R_1,R_2,R_3,R_4)$ with $V = \{a,b,c,c',d,d',Z,Z'\}$, $T = \{a,b\}$, $A = \{cabd, c'aZ, cZ, Zd, Zbd', Zbd, Z'\}$ and the set of rules defined as follows:

$$\begin{aligned}
R_1 &= \{1.1: c\#a\$c'a\#Z, \quad 1.2: c\#a\$\varepsilon\#Z'\} \cup R_A,\\
R_2 &= \{2.1: b\#d\$Z\#bd', \quad 2.2: b\#d\$Z'\#\varepsilon\} \cup R_A,\\
R_3 &= \{3.1: c'\#a\$c\#Z\} \cup R_A,\\
R_4 &= \{4.1: b\#d'\$Z\#d\} \cup R_A,\\
R_A &= \{w\#\varepsilon\$w\#\varepsilon \mid w \in A \text{ and } w \neq cabd\}.
\end{aligned}$$

We claim that $L(\mathcal{H}) = L_{ab}$. Clearly, the control of the system allows to perform the rules 1.1, 1.2, 3.1 and 4.1 in a sequence thus increasing the number of a's and b's simultaneously. Rules 1.2 and 2.2 permit to cut off endmarkers obtaining the terminal word. We remark that the σ^* operation makes the evolution of this system more complicated, e.g. at the second step both words $cabd$ and $c'aabd$ will be present in $L_1(\mathcal{H})$. Moreover, both words will evolve yielding $cabd$, $cabbd'$, $c'aabd$ and $c'aabbd'$ in $L_2(\mathcal{H})$. Now the first two words are eliminated as there is no rule applicable to them in R_3, so $L_3(\mathcal{H})$ will contain $caabd$ and $caabbd'$ only. The first word will be eliminated, as R_4 does not contain any rule applicable to it, so only $caabbd$ will be kept for the new cycle of iterations.

We note that in the example above the elimination of undesired strings was relatively easy, because the rule cycle was long. When a smaller cycle is used, the task becomes more difficult. With ETVDH systems of degree 2 it is possible to achieve the computational completeness, however another method for string elimination should be used [27]. More details can be found in Section 11.5. We remark that ETVDH systems with one component are almost identical to H systems and cannot produce more than regular languages.

Example 8. Consider the TVDH system of degree 2 $\mathcal{H}$ defined as follows: $\mathcal{H} = (V,T,A,R_1,R_2)$ with $V = \{a,b,c,c',d,d',Z,Z',Z'',A,B,C\}$, $T = \{a,b\}$, $A = \{cabd, c'aZ, Zbd', cZ, Zd, Z'A, CZ'', B, Z''\}$ and the set of rules defined as follows:

$$\begin{aligned}R_1 =&\{1.1 : c\#a\$c'a\#Z,\ 1.2 : c\#a\$\varepsilon\#Z'',\ 1.3 : c'\#a\$c\#Z,\ 1.4 : Z'\#A\$B\#\varepsilon\} \cup R_A,\\ R_2 =&\{2.1 : b\#d\$Z\#bd',\ 2.2 : b\#d\$Z'\#\varepsilon,\ 2.3 : b\#d'\$Z\#d,\ 2.4 : C\#Z''\$\varepsilon\#B\} \cup R_A,\\ R_A =&\{w\#\varepsilon\$w\#\varepsilon \mid w \in A \setminus \{cabd, Z''\}\}.\end{aligned}$$

The simulation is similar to the previous example so $L(\mathcal{H}) = L_{ab}$. However, it should be noted that using an axiom Z' and a rule $Z'\#\varepsilon\$Z'\#\varepsilon$ yields to erroneous computations as the word $Z'd'$ that can be obtained can false the check that the correct sequence of rules is applied. This problem is solved by using the method of directing molecules, see Section 11.5, by making Z' and Z'' to appear only at the needed steps of the computation. This is done by rules 1.4 and 2.4.

In the case of TVDH systems of degree 1 the set of rules is applied to the result of the previous application. This corresponds to the iteration of the splicing operation, see Section 11.2.1. It is somehow surprising that even in this case the computational completeness can be achieved. We present in Section 11.5 some details on this result.

11.4 Distributed Splicing

In previous sections we considered models of splicing systems based on H systems. One of the particularities of these models is that the system starts from a single initial set of axioms and at each step the current set of words is replaced by a new one, computed according to the control of the system. In this section we consider splicing systems based on a different idea. Namely, instead of evolving a single set of words, a fixed number of such sets (a vector) is evolved. This corresponds to a distributed system containing some units that we call *components*. The computation is then divided in two different steps: a *computation* step and a *communication* step. During the computation step splicing rules are applied in each component, independently from each other according to the underlying control. During the communication step the contents of components is redistributed in the system according to some algorithm.

We remark that such distributed constructions cannot be expressed in terms of single set models, because splicing is a *binary* operation. This means that strings evolving in different components can interact with each other after the communication step and, moreover, this interaction cannot be predicted in the general case. In contrast, the rewriting operation is *unary* so any distribution of the system makes no sense as it is equivalent to the union of evolutions of every single axiom and the path through the components can be decided in advance as the rewriting rule gives enough information for doing this.

11.4.1 Communicating Distributed H Systems

The model considered in this section is the counterpart of the parallel communicating grammar systems with communication by command. In the literature it is also known under the name of *splicing test tube* systems. The idea is to consider a group of H systems, called (splicing) tubes, as components, use σ^* operation for the computational step in each component and use input permitting filters for the communication according to some communication graph. More precisely, a communicating distributed H (CDH) system of degree n is a construct

$$\Gamma = (V,T,G,(A_1,R_1,F_1),\ldots,(A_n,R_n,F_n))$$

where V is an alphabet, $T \subseteq V$ is the terminal alphabet, A_i are finite languages over V, the axioms, $G = (\{1,\ldots,n\},E)$ is the communication graph of the system, R_i are finite sets of splicing rules and $F_i \subseteq V^*$ are called filters, $1 \leq i \leq n$. We remark that in the original definition [4] as well as in [22] filters F_i are defined in terms of the Kleene star closure of subsets of V. In the same definition the communication graph G is a complete graph with self loops.

The configuration of the system is given by the vector of languages $C = (L_1,\ldots,L_n)$. During the computation step, each component acts like an H system:

$$(L_1,\ldots,L_n) \vdash (L'_1,\ldots,L'_n), \text{ iff } L'_i = \sigma_i^*(L_i), \text{ where } \sigma_i = (V,R_i)$$

During the communication step the contents of each component is redistributed among other components according to the communication graph and the permitting filters. More precisely, if a string from component i belongs to F_j and there is an edge (i,j) in the communication graph, then at the next step this string will belong to component j. The strings that cannot pass any input filter of a connected node remain in the component of their origin. We remark a subtle point here: if a string can go to some other component(s) then each of the components will receive a copy of that string and the initial component will not contain any copy of this string, except the case of a communication graph with a self-loop at the corresponding node and positive filter check. Formally, the communication step is defined as follows:

$$(L_1,\ldots,L_n) \models (L'_1,\ldots,L'_n), \text{ iff } L'_i = \{L_j \cap F_i \mid (i,j) \in E\} \cup (L_i \cap (V^* \setminus \bigcup_{k=1}^{n} F_k))$$

Now a step in a CDH system is done as follows:

$$C \Longrightarrow C' \text{ iff } C \vdash C'' \models C'.$$

The language generated by a CDH system Γ is

$$L(\Gamma) = \{w \in T^* \mid (A_1,\ldots,A_n) \Longrightarrow^* (L_1,\ldots,L_n) \text{ and } w \in L_1\}.$$

The family of languages generated by CDH systems of degree n is denoted as CDH_n.

Example 9. Consider the following CDH system of degree 3 $\Gamma = (V,T,G,(A_1,R_1,F_1),(A_2,R_2,F_2),(A_3,R_3,F_3))$ with $V = \{a,b,c,c',d,d',Z,Z'\}$, $T = \{a,b\}$, and let G be a complete graph with self loops. The components of the system are defined as follows.

$$\begin{aligned}
A_1 &= \{cabd, c'aZ, Zbd', c''Z, Zd''\}\\
R_1 &= \{1.1{:}\, c\#a\$c'a\#Z,\ 1.2{:}\, b\#d\$Z\#bd',\ 1.3{:}\, c\#a\$c''\#Z,\ 1.4{:}\, b\#d\$Z\#d''\}\\
F_1 &= \{a,b,c,d\}^*\\
A_2 &= \{cZ, Zd\}\\
R_2 &= \{2.1 : c'\#a\$c\#Z,\ 2.2b\#d'\$Z\#d\}\\
F_2 &= \{a,b,c',d'\}^*\\
A_3 &= \{cZ, Zd\}\\
R_3 &= \{3.1 : c''\#a\$\varepsilon\#Z',\ 3.2 : b\#d''\$Z'\#\varepsilon\}\\
F_3 &= \{a,b,c'',d''\}^*
\end{aligned}$$

We claim that $L(\Gamma) = L_{ab}$. After the first computational step word $c'aabbd'$ is communicated to the second component. Then after the second computational step word $caabbd$ is obtained and is communicated to component 1. By continuing this loop ca^nb^nd is obtained. At some point endmarkers can be replaced by c'' and d'' and the corresponding word is communicated to component 3. In that component the endmarkers are removed and the resulting terminal string can go back to component 1, yielding the result.

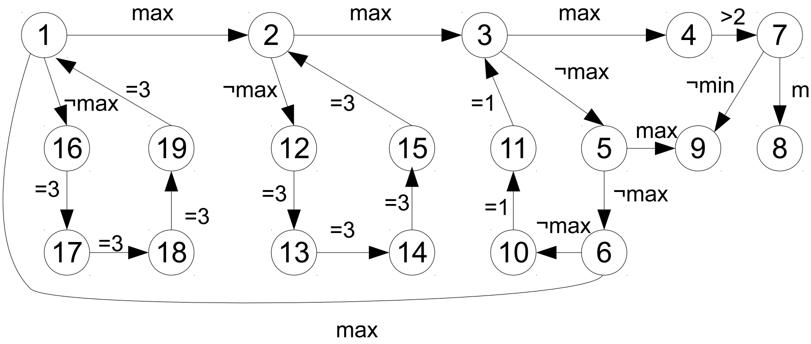
We remark that the filters are expressed in the form of star closure of a subset of V, so they correspond to the definition from [22].

It is possible to adapt the construction from the previous example in order to simulate an arbitrary grammar using the rotate-and-simulate method. So $CDH_3 = RE$. Moreover, as it is shown in [2] a universal system with 8 rules can be constructed. In the case of one component the corresponding system is identical to an H system, so the generated language family is at most regular. In the case of two components there are several results, depending on the type of filters used. When a complete graph with self loops is used (like in the original definition) the computational completeness is obtained using (a) regular filters [6], (b) Kleene star closure of subsets of V^2 [7], (c) alternation of sets of traditional filters (Kleene star closure of subsets of V) [25]. When a complete graph without self-loops is considered, it is possible to achieve the computational completeness with traditional filters [29].

11.4.2 Length-Separating Splicing Test Tube Systems

In this section we consider a variant of splicing test tube systems where the communication mechanism is different and it was never considered in the case of rewriting. More precisely, context conditions on the edges of the communication graph are replaced by numerical predicates acting on the string length. Ten types of predicates are considered: $\leq k$, $< k$, $\geq k$, $>$, $= k$, $\neq k$, max, $\neg$max, min, $\neg$min. For example, the predicate $< k$ permits to pass strings of length smaller than k, while the predicate max permits to pass strings of maximal length. In [5] it is shown that 11 components suffice to generate any RE language.

Example 10. Consider a system Γ having the communication graph below.



Let the axioms and rules of the system be defined as follows:

$$\begin{aligned}
&A_1 = \{cabd, c'aZ, Zbd'\},\\
&R_1 = \{1 : c\#a\$c'a\#Z, 2 : b\#d\$Z\#bd'\}\\
&A_2 = \{ccZ, Zdd\}\\
&R_2 = \{3 : c'\#a\$cc\#Z, 4 : b\#d'\$Z\#dd\}\\
&A_3 = \{Z\}\\
&R_3 = \{5 : c\#c\$\varepsilon\#Z, 6 : d\#d\$Z\#\varepsilon\}\\
&A_4 = \{Z\}\\
&R_4 = \{7 : cc\#a\$\varepsilon\#Z, 8 : dd\#b\$Z\#\varepsilon\}\\
&A_i = \emptyset, R_i = \emptyset, 4 \leq i \leq 19
\end{aligned}$$

It can be seen that $L(\Gamma) = L_{ab}$. Indeed, rules $1-6$ permit to increase simultaneously the number of symbols a and b, while the rules 7 and 8 cut off endmarkers. Until node 3 the right word is the longest one, after that it is the third longest one, so it is filtered as two times $\neg$max and one times max. The axioms are returned to their initial places after 5 steps.

11.4.3 Splicing P Systems

In this section we consider the generalization of the idea of graph-controlled splicing H systems. The idea is to consider n components containing strings and splicing rules having two target indicators – the numbers of components where the corresponding splicing results shall be moved. The language consists of words over terminal alphabet collected in some designated node. The obtained model, called splicing P systems, was introduced in [20] and more details can be found in [21, 23].

Example 11. Consider the following splicing P system $\Pi = (V,T,A_1,A_2,A_3,A_4,R_1,R_2,R_3,R_4,4)$. $V = \{a,b,c,d,Z,Z'\}$, $T = \{a,b\}$, $A_1 = \{cabd, caZ, Z'\}, A_2 = \{Zbd\}, A_3 = \{Z'\}, A_4 = \emptyset$

$$\begin{aligned}
R_1 =& \{1 : c\#a\$ca\#Z; (2,2),\ 2 : c\#a\$\varepsilon\#Z'; (3,3)\}\\
R_2 =& \{3 : b\#d\$Z\#bd; (1,1)\}\\
R_3 =& \{4 : b\#d\$Z'\#\varepsilon; (4,4)\}\\
R_4 =& \emptyset
\end{aligned}$$

We observe that the construction is almost identical to the construction from Example 6, the difference being in the placement of axioms.

It is shown that splicing P systems are computationally complete. Two nodes suffice for that result [28]. When the intersection with terminal alphabet is not used, the computational completeness can be obtained with four nodes [24]. In the same article it is shown that it is also possible to achieve the computational completeness with both targets having the same number. In this case corresponding rules can be associated with the edges of the communication graph. Finally, we would like to mention an interesting result from [2] showing that there exists a universal splicing P system with 5 splicing rules. This result is quite important as it shows that the splicing operation is more powerful than rewriting and it exhibits a very small universal device different from the Turing machine.

An interesting feature of splicing P systems is that some variants allow the modification of the graph structure (creation and deletion of nodes). In this case dynamical systems with dynamical structure are obtained yielding an extremely complex behavior.

11.4.4 Networks of Splicing Processors

Another distribution idea comes by the adaptation of the idea of networks of language processors to the area of splicing. During the communication step a single splicing (σ operation) is used. During the communication step the strings are communicated to the nodes connected on the communication graph if they pass two (regular) filters: the output filter (OF) of the source node and the input filter (IF) of the target node. In some restricted variants regular filters are replaced by permitting and forbidding conditions expressed in terms of a Kleene star closure of a subset of V. We refer to [12, 13] for more details.

Example 12. Consider the following network of splicing processors $\Gamma = (V,T,G,(A_1,R_1,IF_1,OF_1),(A_2,R_2,IF_2,OF_2),(A_3,R_3,IF_3,OF_3))$, $V = \{a,b,c,c',d,d',Z,Z'\}$, $T = \{a,b\}$ and G is a complete graph with three nodes. The components of Γ are defined as follows.

$$\begin{aligned}
A_1 &= \{cabd, c'aZ, Zbd', Z'\}\\
R_1 &= \{1.1\!:c\#a\$c'a\#Z,\ 1.2\!:b\#d\$Z\#bd',\ 1.3\!:c\#a\$\varepsilon\#Z',\ 1.4\!:b\#d\$Z'\#\varepsilon\}\\
IF_1 &= V^*\\
OF_1 &= \{a,b,c',d'\}^*\\
A_2 &= \{cZ, Zd\}\\
R_2 &= \{2.1 : c'\#a\$c\#Z,\ 2.2 : b\#d'\$Z\#d\}\\
IF_2 &= V^*\\
OF_2 &= \{a,b,c,d\}^*\\
A_3 &= \emptyset\\
R_3 &= \emptyset\\
IF_3 &= \{ab\}^*\\
OF_3 &= \emptyset
\end{aligned}$$

It can be easily seen that $L(\Gamma) = L_{ab}$. Indeed, a string *cabd* in the first component needs to be spliced two times using rules 1.1 and 1.2 (yielding $c'aabbd'$) in order to be able to pass the output filter of the component. In the second component only after both primes are removed the corresponding string (*caabbd*) can pass the output filter OF_2. In order to produce the result, the endmarkers are cut off using rules 1.3 and 1.4 and the resulting terminal strings can pass the input filter of the output component 3. We remark that the filters are star languages, so the example corresponds to the simplified variant of the model.

It can be easily seen that the model of networks of splicing processors is quite powerful, so the computational completeness can be easily achieved with 2 nodes [12].

11.5 Refining the Control

In this section we consider a different approach that permits to refine the control of a splicing system. It exploits the idea that splicing is a binary operation, so in order to be applied both strings participating in a splicing should be present. So, by making the second string that can match a splicing rule to appear at specific time moments it is possible to limit the application of the splicing rule to that specific moments. This is possible if the control of the system permits to eliminate (or to move to a different location in the case of distributed systems) strings from the system.

This method is well suited for computations where there is a clear distinction between strings representing the information and strings changing the information by splicing with above strings. More precisely, suppose that during the computation at each step i the configuration C_i of the system can be split into two parts, D_i representing the data and a fixed configuration M containing the strings that may be spliced with the data. Suppose also that there are no possible splicings between elements of D_i, for all i. Then the method of *directing molecules* works by varying in time available elements of M. This can be done by marking by a number, the *state*, the corresponding strings and increasing this number, using new splicing rules, modulo k. Therefore, the presence of a string at some step of the computation will depend on its period k and on its initial state.

The method of directing molecules works well in combination with rotate-and-simulate technique, because the last one fits in the scheme described above. More details can be found in [27] and [26].

We illustrate this procedure using TVDH systems of degree 1 (iterated splicing).

Example 13. Consider TVDH system of degree 1: $\mathcal{H} = (V,T,A,R)$, where $V = \{a,b,c,d,Z,Z_1,Z_1^{(1)},Z_2,Z_2^{(1)}\}$, $T = \{a,b,c,d\}$, $A = \{abb,cZ_1,Z_1c,dZ_2^{(1)},ZZ_1^{(1)},Z_1^{(1)}Z,ZZ_2\}$ and R is defined as follows:

$$\begin{array}{lll}
1: a\#b\$c\#Z_1 & 2: a\#b\$d\#Z_2 & 3: b\#b\$Z_1\#c \\
a.1.1: c\#Z_1\$Z\#Z_1^{(1)} & a.1.2: c\#Z_1^{(1)}\$Z\#Z_1 & a.2.1: Z_1\#c\$Z_1^{(1)}\#Z \\
a.2.2: Z_1^{(1)}\#c\$Z_1\#Z & a.3.1: d\#Z_2\$Z\#Z_2^{(1)} & a.3.2: d\#Z_2\$Z\#Z_2^{(1)}
\end{array}$$

First consider the evolution of words cZ_1 and $ZZ_1^{(1)}$ that can enter rule $a.1.1$. By the definition of the system they will not be part of $L_1(\mathcal{H})$, which will include the result of their splicing: words $cZ_1^{(1)}$ and ZZ_1. The last two strings can enter rule $a.1.2$ yielding cZ_1 and $ZZ_1^{(1)}$ in $L_2(\mathcal{H})$. It is not difficult to see that these words will be present in each L_{2k}, $k \geq 0$ (odd step), while words $cZ_1^{(1)}$ and ZZ_1 will be present in each L_{2k+1} (even step). A similar mechanism permits to make appear words Z_1c and $Z_1^{(1)}Z$ at each odd step and words dZ_2 and $ZZ_2^{(1)}$ at each even step. Hence, Z_1c, cZ_1 and dZ_2 are directing molecules with the period equal to two. The first two words have the initial state 0, while the third word has the initial state 1.

So we can abstract our system to a system having only rules $1-3$ and words Z_1c and cZ_1 present during odd steps of the computation and word dZ_2 present only during even steps. Under these assumptions it can be easily seen that $L(\mathcal{H}) = \{abb,cbb,abc,dbc\}$.

Example 14. In this example we construct a TVDH system $\mathcal{H} = (V,T,A,R)$ of degree 1 generating L_{ab}. This system is defined as follows:

$V = \{a,b,c,c',c'',d,d',d'',Z,X,Y,K,L,M,N\} \cup \{Z_i,Z_i' \mid 1 \leq i \leq 6\}$, $T = \{a,b\}$,

$A = \{cabd,c'aZ_1,cZ_2,c''Z_3,Z_4'bd',Z_5'd,Z_6'd''\} \cup \{ZZ_1',ZZ_2',ZZ_3',Z_4Z,Z_5Z,Z_6Z\} \cup$
$\{Z^2X_{1,1},Z^3R_1,ZX_{1,2},Z^1R_1\} \cup \{Z^1X_{2,1},Z^2R_2,Z^3X_{2,2},ZR_2\}$

The set of rules R is given below (we assume that $1 \leq p,q \leq 2$):

$1.1 : c\#a\$c'a\#Z_1$	$t.1.1.1 : c'a\#Z_1\$Z\#Z'_1$	$t.1.1.2 : c'a\#Z'_1\$Z\#Z_1$
$1.2 : c'\#a\$c\#Z_2$	$t.1.2.1 : c\#Z_2\$Z\#Z'_2$	$t.1.2.2 : c\#Z'_2\$Z\#Z_2$
$1.3 : c\#a\$c''\#Z_3$	$t.1.3.1 : c''\#Z_3\$Z\#Z'_3$	$t.1.3.2 : c\#Z'_3\$Z\#Z_3$
$2.1 : b\#d\$Z_4\#bd'$	$t.2.1.1 : Z_4\#bd'\$Z'_4\#Z$	$t.2.1.2 : Z'_4\#bd'\$Z_4\#Z$
$2.2 : b\#d'\$Z_5\#d$	$t.2.2.1 : Z_5\#d\$Z'_5\#Z$	$t.2.2.2 : Z'_5\#d\$Z_5\#Z$
$2.3 : b\#d\$Z_6\#d''$	$t.2.3.1 : Z_6\#d''\$Z'_6\#Z$	$t.2.3.2 : Z'_6\#d''\$Z_6\#Z$
$1.4 : c''\#a\$\varepsilon\#ZX_{1,1}$	$2.4 : b\#d''\$ZX_{2,1}\#\varepsilon$	
$r.1 : Z\#X_{p,q}\$Z^1\#R_p$	$r.2 : Z^1\#X_{p,q}\$Z^2\#R_p$	$p,q = \{1,2\}$
$r.3 : Z^2\#X_{p,q}\$Z^3\#R_p$	$r.4 : Z^3\#X_{p,q}\$Z\#R_p$	$p,q = \{1,2\}$

We will explain each group of rules in more details. Rules having the number $1.i$, (resp. 2.i) $1 \le i \le 3$ are applicable during odd (resp. even) steps only. This becomes possible as the construction uses the method of directing molecules and corresponding words appear with a period equal to two. Rules $t.k.i.j$, $1 \le k, j \le 2$, $1 \le i \le 3$ permit to implement this behavior. Rules 1.4 and 2.4 are applicable each four steps as corresponding directing words $ZX_{p,q}$, $1 \le p,q \le 2$ appear with a period equal to four. This behavior is implemented using rules $r.1 - r.4$.

Using a similar idea it is possible to show that for any *RE* language $\mathcal{L}$ there exists a TVDH system $\mathcal{H}$ of degree 1 (or an iterated H system) such that $L(\mathcal{H}) = \mathcal{L}$. We also remark that the construction above has the property that $L_i \cap L_{i+1} = \emptyset$.

We note that the control using directing words was possible because in TVDH systems it is possible to eliminate strings. In the case of splicing systems that do not permit string elimination (like splicing test tube systems) a sink node can be used to gather strings that are not used anymore. Another possibility is to redistribute strings to the nodes where they cannot evolve, e.g. in the case of CDH_3 systems [19] or to make “eliminated” strings to move from a tube to another without being able to enter a splicing rule by using the method of directing molecules, e.g. CDH_2 systems without self-loops [29].

11.6 Conclusions

In this paper we made an overview of twelve computational models based on splicing. While using ideas from the area of formal language theory, the functioning of these models is very different from their rewriting counterpart. This is due to the properties of the splicing operation which has a contextual dependence as well as an exchange of information not specified in advance. All these points make the study of such systems fascinating and yield to the discovery of new ideas how to perform and control the computation.

We remark that we did not present all existing models based on splicing. We only mention some omissions like H systems with *target languages*, *locally evolving* H systems, *splicing grammar systems* and *two level distributed* H systems that can be consulted in more details in [22]. We also remark that in some works, e.g. in [31], the splicing is considered as a unary operation. This is achieved by associating the second participating word to the definition of the splicing rule. In this case corresponding models are simpler as there are less evolution possibilities. We remark that we did not present the models from the historical perspective. More bibliographical notes on this topic can be found in [18, 22, 26, 32].

We would like to mention that in this paper we did not discuss descriptional parameters associated to the corresponding systems. We cite the most used parameters in this area: the diameter of splicing rules, the number of splicing rules and axioms, the size of the alphabet, the number of distributed components, the complexity of filters and random conditions. It can be often observed a trade-off between all these measures – for example the set of axioms can be in many cases reduced to a singleton, the trade-off being the increase of the size of the alphabet and of the number of splicing rules [22]. Conversely, it is possible to bound the alphabet to two symbols, the number of rules to 5 and the diameter to (9,6,6,3) paying a huge increase in the number of axioms [2]. Many papers in the area of splicing exhibit such trade-offs, but still a systematical study is missing.

References

1. Alhazov, A., Kogler, M., Margenstern, M., Rogozhin, Y., Verlan, S.: Small universal TVDH and test tube systems. International Journal of Foundations of Computer Science 22(1), 143–154 (2011)
2. Alhazov, A., Rogozhin, Y., Verlan, S.: On small universal splicing systems. International Journal of Foundations of Computer Science 23(07), 1423–1438 (2012)
3. Bonizzoni, P., Jonoska, N.: Regular splicing languages must have a constant. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 82–92. Springer, Heidelberg (2011)
4. Csuhaj-Varjú, E., Kari, L., Păun, G.: Test tube distributed systems based on splicing. Computers and AI 15(2-3), 211–232 (1996)
5. Csuhaj-Varjú, E., Verlan, S.: On length-separating test tube systems. Natural Computing 7(2), 167–181 (2008)
6. Freund, R., Freund, F.: Test tube systems: When two tubes are enough. In: Rozenberg, G., Thomas, W. (eds.) Developments in Language Theory, pp. 338–350. World Scientific (1999)
7. Frisco, P., Zandron, C.: On variants of communicating distributed H systems. Fundamenta Informaticae 48(1), 9–20 (2001)
8. Head, T.: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. Bulletin of Mathematical Biology 49(6), 737–759 (1987)
9. Head, T.: Splicing languages generated with one sided context. In: Computing with Bio-Molecules. Theory and Experiments, pp. 158–181 (1998)
10. Kari, L.: DNA computing: Arrival of biological mathematics. The Mathematical Intelligencer 19(2), 9–22 (1997); Earlier version under the title DNA computers, tomorrow's reality. Bulletin of the European Association for Theoretical Computer Science (59), 256–266 (1996), `http://www.csd.uwo.ca/lila/amsn.ps`
11. Kari, L., Kopecki, S.: Deciding whether a regular language is generated by a splicing system. In: Stefanovic, D., Turberfield, A. (eds.) DNA 2012. LNCS, vol. 7433, pp. 98–109. Springer, Heidelberg (2012)
12. Loos, R., Manea, F., Mitrana, V.: On small, reduced, and fast universal accepting networks of splicing processors. Theoretical Computer Science 410(4-5), 406–416 (2009)

13. Manea, F., Martín-Vide, C., Mitrana, V.: Accepting networks of splicing processors: Complexity results. Theoretical Computer Science 371(1-2), 72–82 (2007)
14. Margenstern, M., Rogozhin, Y.: Time-varying distributed H systems of degree 2 generate all RE languages. In: MFCS 1998 Workshop on Frontiers of Universality (1998)
15. Margenstern, M., Rogozhin, Y.: A universal time-varying distributed H system of degree 2. Biosystems 52, 73–80 (1999)
16. Margenstern, M., Rogozhin, Y.: Time-varying distributed H systems of degree 1 generate all recursively enumerable languages. In: Ito, M., Păun, G., Yu, S. (eds.) Words, Semigroups, and Transductions, pp. 329–339. World Scientific (2001)
17. Margenstern, M., Rogozhin, Y., Verlan, S.: Time-varying distributed H systems with parallel computations: the problem is solved. In: Chen, J., Reif, J.H. (eds.) DNA 2003. LNCS, vol. 2943, pp. 48–53. Springer, Heidelberg (2004)
18. Margenstern, M., Rogozhin, Y., Verlan, S.: Time-varying distributed H systems: An overview. Fundamenta Informaticae 64(1-4), 291–306 (2005)
19. Priese, L., Rogozhin, Y., Margenstern, M.: Finite H-systems with 3 test tubes are not predictable. In: Altman, R., Dunker, A., Hunter, L., Klein, T. (eds.) Proceedings of Pacific Symposium on Biocomputing, 3, Kapalua, Maui, pp. 547–558. World Scientific, Hawaii (1998)
20. Păun, G.: Computing with membranes. Journal of Computer and System Sciences 1(61), 108–143 (2000); Also TUCS Report No. 208 (1998)
21. Păun, G.: Membrane Computing. An Introduction. Springer(2002)
22. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing: New Computing Paradigms. Springer (1998)
23. Păun, G., Rozenberg, G., Salomaa, A.: The Oxford Handbook Of Membrane Computing. Oxford University Press (2009)
24. Verlan, S.: About splicing P systems with immediate communication and non-extended splicing P systems. In: Martín-Vide, C., Mauri, G., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 369–382. Springer, Heidelberg (2004)
25. Verlan, S.: Communicating distributed H systems with alternating filters. In: Jonoska, N., Păun, G., Rozenberg, G. (eds.) Aspects of Molecular Computing. LNCS, vol. 2950, pp. 367–384. Springer, Heidelberg (2003)
26. Verlan, S.: Head systems and applications to bioinformatics. Ph.D. thesis, University of Metz (2004)
27. Verlan, S.: A boundary result on enhanced time-varying distributed H systems with parallel computations. Theoretical Computer Science 344(2-3), 226–242 (2005)
28. Verlan, S., Margenstern, M.: About splicing P systems with one membrane. Fundamenta Informaticae 65(3), 279–290 (2005)
29. Verlan, S., Margenstern, M.: Universality of splicing test tube systems with two tubes. Fundam. Inform. 110(1-4), 329–342 (2011)
30. Verlan, S., Zizza, R.: 1-splicing vs. 2-splicing: Separating results. In: Harju, T., Karhumäki, J. (eds.) Proceedings of WORDS 2003, 4th International Conference on Combinatorics on Words, Turku, Finland, September 10-13, pp. 320–331. TUCS General Publication No. 27 (2003)
31. Zandron, C.: A model for molecular computing: Membrane systems. Ph.D. thesis, Dipartimento di Scienze dell'Informazione, Universita' degli Studi di Milano, Milano, Italy (2002)
32. Zizza, R.: Splicing systems. Scholarpedia 5(7), 9397 (2010)

Chapter 12
Linear Cellular Automata and Decidability

Klaus Sutner

Abstract. We delineate the boundary between decidability and undecidability in the context of one-dimensional cellular automata. The key tool for decidability results are automata-theoretic methods, and in particular decision algorithms for automatic structures, that are inherently limited to dealing with a bounded number of steps in the evolution of a configuration. Undecidability and hardness, on the other hand, are closely related to the full orbit problem: does a given configuration appear in the orbit of another?

12.1 Linear Cellular Automata

Cellular automata arise as a natural discretization of continuous dynamical systems. In the continuous case, we are dealing with configurations of the form, say, $X : \mathbb{R} \to \Sigma$, where Σ is some set of values whose precise nature is not important here. Write $\mathcal{C}$ for the space of all configurations. The shift operation $\sigma_a(x) = x + a$ acts on $\mathcal{C}$ in the natural way and accounts for a change in coordinates. The dynamics of the system are then given by a family of evolution operators $\tau_t : \mathcal{C} \to \mathcal{C}$, where "time" t is a non-negative real. These operators form a monoid under composition: $\tau_t \circ \tau_s = \tau_{t+s}$ and $\tau_0 = \mathrm{Id}$. By discretizing the underlying space, as well as the operators σ and τ, we naturally arrive at maps $G : \Sigma^{\mathbb{Z}} \to \Sigma^{\mathbb{Z}}$ that are continuous in the standard product topology and invariant under the discrete shift operator $\sigma : \Sigma^{\mathbb{Z}} \to \Sigma^{\mathbb{Z}}$, $\sigma(X)(i) = X(i+1)$. This is expressed in the classical Curtis-Hedlund-Lyndon theorem, see [18]. Hence, in a cellular automaton the *global map* $G = G_\rho$ can be represented by a finite lookup table, essentially a map $\rho : \Sigma^w \to \Sigma$, and is thus amenable to analysis from the perspective of decidability and computational complexity. It is customary to refer to this finite function as the *local map* or *rule* of

Klaus Sutner
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213, USA
e-mail: sutner@cs.cmu.edu

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_12

the cellular automaton. The operator monoid is generated by G and we are therefore interested in the iterates G^t for all $t \in \mathbb{N}$.

One of the main areas of interest in the study of cellular automata is their *classification* with respect to the evolution of configurations, i.e., the repeated application of the global map. This problem was expressed by Wolfram in [53] as "What overall classification of cellular automata behavior can be given?" While aspects of short-term evolution such as reversibility are not excluded, it is clear from the classification proposed by Wolfram that he had the long-term evolution in mind, see [52,54]. In a nutshell, the Wolfram classification looks like so. The evolution of a configuration leads to

- Class I: homogeneous fixed points,
- Class II: periodic configurations,
- Class III: chaotic, aperiodic patterns,
- Class IV: persistent, complex patterns of localized structures.

In [27,28] Li and Packard introduce an alternative version of this hierarchy. Again, their classification is based on the asymptotic behavior of the automaton.

Another well-known attempt at classification is due to Kurka, see [25,26,31, 47], and is based on the topological notions of equicontinuity, sensitivity to initial conditions and expansivity. Needless to say, these notions all relate to long-term behavior.

In [30], Margenstern gave a comprehensive description of the "frontier between decidability and undecidability," in a variety of computational settings. In this paper we will focus on one-dimensional cellular automata and discuss an approach to classification that is based on model theory and computability rather than classical dynamics. Our key tool is the use of automata theory to obtain decidability results for the first-order theory of cellular automata, construed as first-order structures. Using first-order logic one can easily formalize assertions such as "the global map is surjective," "the system is reversible", "the global map is k-to-1" where k is fixed, "there exists a 5-cycle" or "there are exactly 2 fixed points." Thus, we will deal exclusively with one-dimensional cellular automata; their higher-dimensional analogues are not amenable to these techniques as can be seen for example from Kari's theorem concerning the undecidability of injectivity and surjectivity of the global map in dimension 2, see [20]. As Douglas Lind pointed out, the study of higher-dimensional cellular automata requires one to step into "the Swamp of Undecidability," see [29]. In fact, even the most basic question of whether a shift of finite type, given by a finite collection of forbidden 2-dimensional patterns, is nontrivial already turns out to be undecidable.

More formally, suppose a one-dimensional cellular automaton has local map $\rho : \Sigma^w \to \Sigma$. We consider the associated first-order relational structure

$$\mathfrak{C}_\rho = \langle \mathcal{C}, \to \rangle$$

where Σ is a finite set, the *alphabet* of the cellular automaton, and w the *width*, $\mathcal{C}$ denotes the space of *configurations*. We always assume that our language includes equality without further mention. Given a first-order sentence φ, we want to determine whether φ is valid over $\mathfrak{C}_\rho$, in symbols $\mathfrak{C}_\rho \models \varphi$. In computer science this problem is often referred to as *model checking* , see [9, 19]. There are two natural variants that are both relevant in the context of cellular automata: first, in *expression model checking* the structure is fixed and we are interested in establishing the validity of a number of assertions. For example, we may be interested in understanding properties of particularly interesting specific elementary cellular automata such as rule 30 or rule 110, [54]. Second, in *data model checking* the sentence is fixed and one tries to determine validity in as many structures as possible. For example, one might try to examine the existence of a particular finite subgraph of $\mathfrak{C}_\rho$ for all elementary cellular automata ρ. In our case, the machinery for both problems is quite similar, though in particular for data model checking it is important to optimize the decision procedures in order to avoid efficiency problems.

The reasons for representing the global map as a binary relation rather than a function are purely technical and need not concern us here. To avoid trivial cases, we assume that Σ has cardinality at least 2, so configuration spaces are infinite and indeed uncountable, but see section 12.2 below for a natural reduction to countable subspaces. For our purposes, there are three natural choices of the configuration space $\mathcal{C}$: the bi-infinite case Σ^ζ, the one-way infinite case Σ^ω and the finite case Σ^n. In the last case, $n \in \mathbb{N}$ is a parameter and we are mostly concerned with the *spectrum* of a first-order sentence, see section 12.2.1 below. Correspondingly, we write $\mathfrak{C}_\rho^n$ for the finite structure $\langle \Sigma^n, \rightarrow \rangle$, $\mathfrak{C}_\rho^\omega$ for the one-way infinite structure $\langle \Sigma^\omega, \rightarrow \rangle$ and $\mathfrak{C}_\rho^\zeta$ for the two-way infinite structure $\langle \Sigma^\zeta, \rightarrow \rangle$, following the terminology established in [35, 40] for the corresponding automata. In all cases, we think of configurations as *words* over a finite alphabet Σ and use appropriate finite state machines to obtain decision algorithms: ordinary word automata in the finite case, Büchi automata in the infinite case and ζ-automata in the bi-infinite case. Note that special care is needed to deal with boundary conditions both in the finite and one-way infinite case. We point out that the notion of finite configuration here refers exclusively to configurations that are finite in the sense that they involve a finite grid of cells. Unfortunately, it is customary in the literature to use the same terminology with respect to infinite configurations that have finite support: only finitely many cells are in a state different from some specially designate null state. We will avoid this usage here. At any rate, $\mathfrak{C}_\rho$ is an *automatic structure* in the sense of Khoussainov and Nerode [22, 23] and generalizations in [5]. Historically, automaticity was first exploited in work by Gilman, Cannon, Holt, Thurston and others in study of various types of groups, see [12]. More recently, there has been substantial progress in elucidating the structure of automorphism groups of the infinite binary tree that are described by certain types of Mealy automata, see [4, 15, 16, 34, 50]. As we will see in section 12.2, automaticity can be exploited in all three cases to produce decision algorithms for the first-order theory of these structures. Alas, the efficiency of the algorithms varies considerably; in particular in the bi-infinite case the corresponding ζ-automata are algorithmically difficult to handle and it is hard to push the decision methods beyond very basic properties. One interesting technique in this connection is to enlarge the language by adjoining suitable automatic predicates. For example, the predicate “differ in only finitely many places” is useful to express surjectivity of the global map in the bi-infinite case and easily seen to be automatic.

Needless to say, any full classification of cellular automata will require stronger logics such as monadic second-order logic or transitive closure logic. Alternatively, we can consider augmented structures

$$\mathfrak{T}_\rho = \langle \mathcal{C}, \to, \xrightarrow{*} \rangle.$$

where $\xrightarrow{*}$ denotes the transitive reflexive closure of $\to$, the *reachability* or *orbit relation* of the automaton. Whenever necessary, we use qualifiers n , ω and ζ to indicate the type of cellular automaton under consideration. Pace Lind, this leads back into the undecidability swamp: the first-order theory of $\mathfrak{T}_\rho$ is not decidable in general, though there are interesting automatic relations where the augmented structure surprisingly remains automatic and can thus be handled in very much the same way as the plain structure, see [50].

In the following section 12.2 we will describe the machinery required to produce decision algorithms for the first-order theory of all three kinds of cellular automata. As we will see, there are significant efficiency obstacles to overcome, in particular in the bi-infinite case. Section 12.3 then summarizes corresponding undecidability results in connection with the long-term evolution of configurations. Lastly, we comment on open problems and future work.

12.2 The First-Order Theory

In this section we show how to solve a very limited version of the Entscheidungsproblem for one-dimensional cellular automata: the first-order theory of all these automata is decidable. Of course, the first-order theory of $\mathfrak{C}_\rho$ is too weak to deal with aspects of the long-term behavior of the cellular automaton, but it easily captures elementary properties such as injectivity, surjectivity, k-to-1-ness, the existence of k-cycles and so on. We can think of these properties as being local in the temporal sense, but note that we can obviously construct a first-order sentence that asserts, say, that every configuration has distance at most k from an ℓ-cycle. Or we can express the assertion that a particular finite directed graph has an isomorphic copy somewhere in $\mathfrak{C}_\rho$. As we will see, all these temporally local properties are decidable, at least in principle.

12.2.1 Finite Configurations

In the finite case we are dealing with structures $\mathfrak{C}^n_\rho$ which are clearly automatic, uniformly in n, meaning that the same automata can be used for all these structures.

As a sample decision problem, consider the standard question of testing reversibility of the system, i.e., injectivity of the global map. The corresponding problem for bi-infinite cellular automata was handled by Amoroso and Patt [2] and appears to be the first clear example of a decidability result in the context of cellular automata, following the paradigm established by Rabin and Scott [36]. To see how to tackle injectivity in our setting, consider first an automaton $\mathcal{A}_\rho(x,y)$ that tests whether two finite words $X,Y \in \Sigma^n$ are related by $X \to Y$. The automaton works on words over the alphabet Σ^2 which we may consider as having two tracks:

$$X{:}Y = \quad \begin{array}{|c|c|c|c|c|c|c|}\hline x_1 & x_2 & \dots & x_i & \dots & x_{n-1} & x_n \\ \hline y_1 & y_2 & \dots & y_i & \dots & y_{n-1} & y_n \\ \hline \end{array} \in \Sigma^2$$

This two-track word $X{:}Y \in (\Sigma^2)^n$ is often referred to as the *convolution* of X and Y, unfortunate but well-established terminology. For simplicity assume that the width $w = 2r+1$ of the local map is odd. To test whether word y is obtained from x by uniform application of the local map on all finite blocks of the form $x_{i-r},\dots,x_i,\dots,x_{i+r}$, the appropriate automaton $\mathcal{A}_\rho(x,y)$ has state set $\Sigma^{2r} \times \Sigma^r$ and the transitions are of the form

$$\begin{array}{c} a_1,\dots,a_{2r} \\ b_1,\dots,b_r \end{array} \xrightarrow{a:b} \begin{array}{c} a_2,\dots,a_{2r},a \\ b_2,\dots,b_r,b \end{array}$$

provided that $\rho(a_1,\dots,a_{2r},a) = b_1$. Here $a{:}b$ indicates the 2-track letter composed of $a,b \in \Sigma$ and labels the transition. Thus, we are dealing with a subautomaton of the complete de Bruijn automaton over Σ^2 of order $2r$: we remove all the directed edges from the full de Bruijn automaton that do not conform to ρ. An example is shown in figure 12.1. Note that the underlying CA is the additive rule 150, as a consequence the automaton uses the full de Bruijn graph.

A further complication is caused by the boundary conditions associated with a finite cellular automaton. In the case of fixed boundary conditions, we can augment the de Bruijn automaton with additional initial and final states that represent the phantom cells. More precisely, the initial states have indegree 0 and transitions leading to the main automaton; the final states have outdegree 0 and are reachable via transitions leaving the main automaton. To deal with periodic boundary conditions, these additional states have to be associated appropriately.

Injectivity is now easily expressed in our relational setting as the first-order formula

$$\varphi \equiv \forall x,y,z\,(x \to z \land y \to z \Rightarrow x = y)$$

To convert the sentence φ into an automaton $\mathcal{A}_\varphi$ we use the 3-track alphabet $\Gamma = \Sigma^3$. We combine two copies of the $\to$-testing automaton, $\mathcal{A}_\rho(x,z)$ and $\mathcal{A}_\rho(y,z)$, associated with tracks in the obvious manner. A standard product construction is used to express logical conjunction of $x \to z$ and $y \to z$, refer to the result as $\mathcal{A}'$. Lastly, let $\mathcal{A}$ be the conjunction of $\mathcal{A}'$ and an automaton testing inequality $x \neq y$. This last step amounts to creating two copies of $\mathcal{A}'$, with appropriate cross transitions that verify inequality. To check injectivity we need to test the acceptance language of $\mathcal{A}$ for emptiness. Clearly, this last step can be handled in time linear in the size of

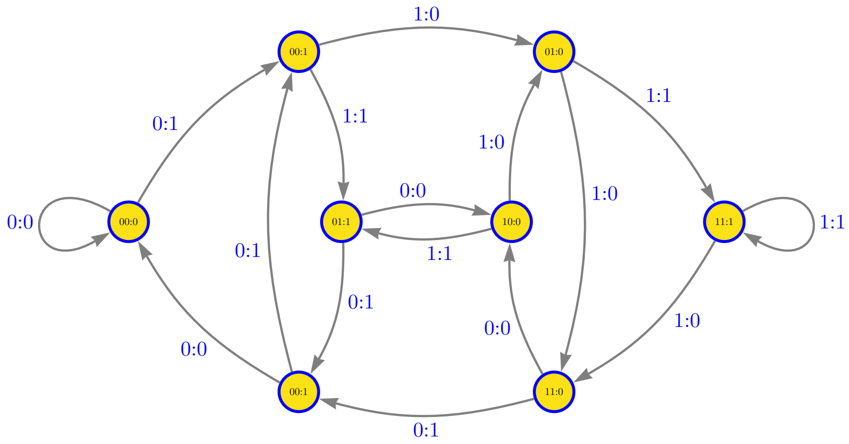


Fig. 12.1 An example of an automaton $\mathcal{A}_\rho$; the underlying cellular automaton here is the additive elementary automaton number 150

$\mathcal{A}$, which size is quadratic in the size of the lookup table for the local map of the cellular automaton and we have a perfectly practical algorithm.

In the general case, the algorithm rests on the fact that regular languages form an effective Boolean algebra. Suppose we are given a first-order formula $\varphi(x_1,\dots,x_k)$ with k free variables as indicated. The decision algorithm constructs an automaton $\mathcal{A}_\varphi$ that accepts exactly those k-track words that satisfy the formula:

$$\mathcal{L}(\mathcal{A}_\varphi) = \{u_1{:}u_2{:}\dots{:}u_k \in \Gamma^\star\}\, \mathfrak{C}_\rho \models \varphi(u_1,u_2,\dots,u_k)$$

where $\Gamma = \Sigma^k$ is the k-track version of Σ. The construction proceeds by induction on the subformulae of φ. For simplicity, assume that the given formula is in prefix normal form, but note that this may not be the most desirable setup for efficiency reasons.

We can handle the Boolean connectives in the matrix of the formula using standard algorithms such as product machines and determinization, see [40] for a recent description of the requisite machinery. Note that determinization here refers to the classical Rabin-Scott algorithm [36] used for word automata; in the following sections significantly more complicated procedures are required.

Universal quantifiers can be translated into existential ones via the standard transformation $\forall x\, \varphi \equiv \neg\exists x \neg\varphi$. Somewhat surprisingly, existential quantifiers are actually easier to handle than logical connectives, at least from a purely algorithmic perspective: we can simply erase the corresponding track from the transition labels. Note, however, that this transformation usually introduces nondeterminism, which can cause problems at later stages if determinization is required to deal with logical negation. Unfortunately, this typically happens when dealing with universal quantifiers in the manner described above.

A first-order sentence has no free variables. To handle this case properly it is convenient to adopt the convention that $(\Sigma^0)^\star$ is the 2-element Boolean algebra $\{\top, \bot\}$. Projection yields $\bot$ if the set is empty and $\top$ otherwise. To determine which case obtains one has to test the corresponding regular language for emptiness, which can be handled by standard path-existence algorithms in the unlabeled directed graph that results after all quantifiers have been removed. At any rate, we have the following result, see also [48].

Theorem 1. *First-order logic for finite one-dimensional cellular automata is decidable.*

It should be noted that the use of product automata and determinization has the potential effect of exponential blow-up in the size of the finite state machines involved in the decision algorithm. Hence, as a practical matter, only relatively simple formulae can be handled.

While the automaton $\mathcal{A}_\varphi$ can be used to determine validity for specific values of n, it is usually more interesting to exploit it to compute the *spectrum* of φ, defined as the tally language

$$\mathsf{spec}(\varphi) = \{0^n\} \mathfrak{C}_\rho^n \models \varphi \subseteq 0^\star$$

As a tally language, the spectrum can be determined by a finite state machine and hence must be regular.

Lemma 1. *Any sentence in first-order logic has regular spectrum over the structures $\mathfrak{C}_\rho^n$.*

Alternatively we can think of the spectrum as a set of natural numbers, in which case the set must be semi-linear (i.e., a finite union of linear sets). As an example consider the elementary cellular automaton number 90, an additive automaton whose local rule corresponds to the exclusive-or of the left and right argument. The property "every configuration has exactly 4 predecessors" here has spectrum $2\mathbb{N}$. Likewise, elementary cellular automaton number 30 has spectrum $12\mathbb{N}$ for the property "there is a 3-cycle." See Wolfram [54] and references therein for more background on elementary cellular automata. One particularly important case arises when the spectrum of a sentence is universal: all finite grids have the property in question. We can test universality of a first-order sentence φ by testing universality of the associated automaton, at least disregarding the fact that this test is PSPACE-hard in general [14]. In general, efficiency is a non-negligible issue in the finite case, but with some effort one can obtain feasible decision algorithms for interesting properties.

12.2.2 Infinite Configurations

To lift our decision algorithm to infinite configurations of the form $X \in \Sigma^\omega$ we need to generalize ordinary word automata to machines operating on infinite inputs. More precisely, we can retain the finite transition systems that describe word automata, but we have to work with a significantly more complicated acceptance condition. To this end, a *Büchi automaton* $\mathcal{B} = \langle Q, \Sigma, \tau; I, F \rangle$ is said to accept a word $X \in \Sigma^\omega$ if there is a computation $\mathcal{B}$ on X that starts at a state in I and touches F infinitely often, see [35,40]. In other words, for a computation π, let $\mathrm{Inf}(\pi) = \{ p \in Q \mid \overset{\infty}{\exists} i \in \mathbb{N}\, (p_i = p) \}$ denote the set of recurrent states in π. Then $\mathcal{B}$ accepts X if there exists some computation π on X, starting at I, such that $\mathrm{Inf}(\pi) \cap F \neq \emptyset$. Of course, as a finite data structure a Büchi automaton is indistinguishable from an ordinary nondeterministic automaton.

The addition of automatic predicates to our language makes it possible to describe other properties of interest. For example, consider the left-shift operator L and suppose we adjoin a predicate xLy, also denoted by L, that is interpreted as "x is the left-shift of y." Using the standard sloppy notation to express chains of relations, we can then write in the extended language $\mathcal{L}(\rightarrow, L)$

$$\exists x, y, z, u, v\, (x \rightarrow y \rightarrow z \rightarrow u \wedge uLvLx)$$

to formalize the assertion that, for some configuration X, we have $G^3(X) = L^2(X)$. Thus we can state that 3 steps in the temporal evolution correspond to a double left-shift. Similarly we could restrict our attention to spacially periodic configurations of a fixed period. Note that chains such as $x \rightarrow y \rightarrow z \rightarrow u$ in the preceding formula require an adjustment in the basic de Bruijn automaton; except for u both tracks of a state will now contain $2r$ symbols.

Symbolic dynamics is concerned with the Cantor space Σ^ω, but our use of finite state machines to determine first-order properties has the side effect that we are essentially dealing with a smaller space. Define a configuration to be *ultimately periodic* if it is of the form vw^ω where v and w are finite words, w not empty. Thus, after a finite initial segment v the configuration simply repeats the block w forever. We write $\mathscr{C}_{\mathsf{up}}$ of this space of configurations. From the perspective of computability, ultimately periodic configurations may seem ad hoc, but they have a perfectly good justification in terms of model theory:

Theorem 2. *The space $\mathscr{C}_{\mathsf{up}}$ of ultimately periodic configurations is an elementary substructure of the full space.*

This is easy to see using the standard Tarski-Vaught test, see [8, 49]. Thus, ultimately periodic configurations are finitary objects but are indistinguishable from arbitrary configurations from the perspective of first-order logic. In fact, they form the least class of configurations that contain configurations of finite support and form an elementary subspace. Another useful property of this subspace is that its elements afford a natural finite description as pairs of finite words (v, w). Hence the orbits on $\mathscr{C}_{\mathsf{up}}$ are *recursively enumerable*, or r.e. for short, see [41]. As in the finite case, the collection of ω-regular languages over some alphabet forms an effective Boolean algebra and we can lift the construction from the finite to the infinite case.

Theorem 3. *First-order logic for infinite one-dimensional cellular automata is decidable.*

It was pointed out by O. Finkel that we can extend our language slightly by quantifiers for "there exist infinitely many" and "there exist r mod k many" without affecting decidability, see [13]. As a consequence, we can check, say, whether there are infinitely many fixed points. Alas, efficiency problems now become dominant: it is still straightforward to determine emptiness of a Büchi automaton, but the construction of the machines becomes problematic. As an example, consider the following test related to *nilpotency*: a cellular automaton is nilpotent if there exists a quiescent configuration Y such that all configurations evolve to Y in finitely many steps. A standard compactness argument shows that there has to be a fixed bound n, the nilpotency index, that limits the length of the corresponding transient:

$$\exists y \text{ quiescent } \forall x \exists x_1, \dots, x_{n-1} \left(x \to x_1 \to x_2 \to \dots \to x_{n-1} \to y\right)$$

A priori, quiescence is not first-order definable in $\mathcal{L}(\to)$, but we can easily add an automatic predicate that singles out quiescent configurations. The matrix of the formula has a simple structure, and can easily be converted into a product de Bruijn automaton; alas, the size of this automaton is exponential in n and there is little hope to be able to deal with the universal quantifier, except for exceedingly small values of n.

While Büchi automata can be represented by the exact same data structure as a standard nondeterministic automaton, some of the attendant algorithms are significantly more complicated. In particular determinization based on Safra's algorithm [39] is notoriously difficult to implement well; the upper bound of $O(n^n)$ is known to be tight in general. Indeed, much effort has gone into finding ways around determinization, see [24] and references there. Minimization techniques are also of little help, since one would need to minimize nondeterministic machines: deterministic Büchi automata are strictly weaker than their nondeterministic counterparts, so one has to resort to other devices such as Rabin and Muller automata, see [35]. The data structures associated with these types of automata can be quite large and are difficult to deal with in conversion algorithms between the various types of machines. Overall, and in contrast to the finite case, it is quite difficult to test interesting properties and more effort is needed to find ways to design feasible decision algorithms.

12.2.3 Bi-infinite Configurations

The study of classical symbolic dynamics deals with another, yet more complicated situation: bi-infinite words $X \in \Sigma^{\zeta}$. Acceptance conditions become correspondingly more unwieldy and, informally, need to cover an infinite past as well as an infinite future. To this end let $\mathrm{Inf}^{-}(\pi) = \{ p \in Q \mid \overset{\infty}{\exists} i \in \mathbb{N}\,(p_{-i} = p) \}$ and $\mathrm{Inf}^{+}(\pi) = \{ p \in Q \mid \overset{\infty}{\exists} i \in \mathbb{N}\,(p_{i} = p) \}$ denote the set of negatively and positively recurrent states in π. Then $\mathcal{A}$ accepts a bi-infinite word X if there exists some computation π on X such that $\mathrm{Inf}^{-}(\pi) \cap I \neq \emptyset$ and $\mathrm{Inf}^{+}(\pi) \cap F \neq \emptyset$. While these conditions are arguably more complicated, they sometimes are actually easier to deal with than in the plain infinite case. To wit, in the the basic de Bruijn automata all states are initial and final, so that $\mathcal{A}_{\rho}(x,y)$ accepts its input if there is a bi-infinite computation: no complications arise from boundary conditions as in the finite and infinite case. This was used in [44] to give simple quadratic algorithms to test for injectivity, surjectivity and openness of the global map.

Of course, for more complicated formulae the construction of the associated automaton overall becomes more involved. The only known approach is to split a ζ-word into two ω-words and to use two Büchi automata to handle these one-way infinite input words, see again [35] for a careful discussion. Note, though, that we are actually dealing with a whole family of pairs of Büchi automata, corresponding to a representation of the ζ-regular languages in the form

$$L = \bigcup_{i \in I} U_i^{\mathsf{op}} V_i$$

where U^{op} stands for reversed ω-words obtained from U. The languages U_i and V_i are ω-regular and the index set I is finite. As in the infinite case we can identify a collection of ultimately periodic words, this time words of the form ${}^{\omega}uvw^{\omega}$ where u, v, w are finite words. Again we obtain an elementary subspace which we denote by $\mathscr{C}_{\mathsf{up}}$.

Theorem 4. *The space $\mathscr{C}_{\mathsf{up}}$ of ultimately periodic configurations is an elementary substructure of the full space of bi-infinite configurations.*

One can then express all the necessary Boolean operations as well as projections in terms of this union-of-pairs representation. Alas, there are substantial efficiency problems; in particular determinization becomes highly problematic. To see why, note that the first step in the determinization process is to make sure that that the languages U_i are pairwise disjoint, and likewise for the V_i. This can be achieved by exploiting the fact that ω-regular languages form and effective Boolean algebra, but may cause an exponential blow-up in the size of the index set. In conjunction with the complexity of determinization of Büchi automata this makes if typically unfeasible to handle first-order sentences of all but the most limited complexity. Even pure Σ_1 sentences can cause major efficiency problems if the matrix of the formula is sufficiently large. A typical example for this kind of problem is again the existence of "long" $\rightarrow$-chains, the automaton for the matrix grows exponentially in k. Still, we have a decision algorithm, at least in principle.

Theorem 5. *First-order logic for bi-infinite one-dimensional cellular automata is decidable.*

In this context it is particularly important to exploit computational shortcuts whenever possible. For example, surjectivity is a priori the Π_2 statement $\forall x \exists y (y \to x)$. The automaton corresponding to $\mathcal{A}_{\exists y (y \to x)}$ is nondeterministic and thus the test for universality necessitated by the outermost quantifier can cause exponential blow-up. On the other hand, one can exploit Hedlund's classical result that characterizes surjectivity in terms of injectivity on configurations with finite support. More precisely, introduce the obviously automatic predicate "equal except for finitely many places," in symbols $x\,E\,y$. We can now express surjectivity as $\forall x,y,z\,(x \to z \wedge y \to z \wedge x\,E\,y \Rightarrow x = y)$ in the extended language $\mathcal{L}(\to, E)$. Testing this formalization is significantly easier: once the automaton for the matrix of the formula has been constructed we simply have to solve a path-existence problem in a directed graph, a problem easily tackled in linear time and space.

Perhaps more importantly, the choice of suitable additional predicates makes it possible to formalize properties that fail to be first-order in the original language $\mathcal{L}(\to)$. As an example, consider openness of the global map, a property known to be equivalent to the map being k-to-1 for some k. As such, the property cannot be formalized in first-order. However, consider the predicate $x =_L y$ if $\exists n \in \mathbb{Z} \forall i < n\,(x_i = y_i)$ and likewise for $x =_R y$. It is easy to see that both predicates can be tested by a finite state machine. Both are weaker versions of being almost equal: $x\,E\,y \iff x =_L y \wedge x =_R y$. Hence, the global map is open if, and only if,

$$\forall x,y,z\,\big(x \to z \wedge y \to z \wedge (x =_L y \vee x =_R y) \Rightarrow x = y\big),$$

following the same pattern as injectivity and surjectivity and using the language $\mathcal{L}(\to, =_L, =_R)$. With a little more effort this produces an alternative proof for the quadratic algorithms from [44].

12.3 Undecidability and Hardness

When it comes to undecidability, there is little difference between the one-way infinite situation and the two-way infinite one, and we will focus on the latter. It is clear from the previous sections that the framework appropriate to the study of undecidability and computational hardness is given by the augmented structures $\mathfrak{T}_\rho^\zeta = \langle \mathcal{C}, \to, \overset{*}{\to} \rangle$ and their finite counterparts $\mathfrak{T}_\rho^n = \langle \Sigma^n, \to, \overset{*}{\to} \rangle$. Moreover, since the orbit relation $\overset{*}{\to}$ involves transitive closure one can expect hardness results for the plain Reachability Problem: for two configurations X and Y that have finitary descriptions, is $X \overset{*}{\to} Y$? Technically we need to augment our language by appropriate constants to reflect Reachability, but we will ignore these details. The first observation is

Theorem 6. *Reachability over $\mathfrak{T}_\rho^n$, uniformly in n, is* PSPACE*-hard.*

This follows from the fact that cellular automata are easily capable of simulating linear bounded automata. On the other hand, over $\mathfrak{T}_\rho^n$, validity of any first-order sentence is trivially decidable in polynomial space, so the validity problem is PSPACE-complete. Thus, the problem is still decidable in principle, though the computation may well fail to be feasible. This changes drastically when we try to classify all finite structures at once by determining the spectrum of a sentence. In particular, consider

$$FP \equiv \forall x \exists y \, (x \xrightarrow{*} y \wedge y \rightarrow y),$$

a Π_2 sentence that expresses the assertion that every orbit ends in a fixed point. We can construe *FP* as a formalization of the first Wolfram class, see [11,42] for a discussion of more general attempts to formalize Wolfram's heuristics. To test membership in this class we have to solve the data version of the model checking problem. Alas, this problem is undecidable.

Theorem 7. *It is undecidable whether the spectrum of the sentence FP from above is universal. In fact, this property is Π^0_1-complete.*

It should be noted that this result does not follow directly from standard results about the computations of Turing machines: we need to deal with all possible configurations of the cellular automaton, not just the ones that code stages in a computation. As it turns out, similar assertions about the length of the limit cycle of all orbits in $\mathfrak{C}^n_\rho$ for all n are undecidable, see [43].

With a view towards the importance of Reachability, it is desirable to address decidability questions in the augmented structures $\mathfrak{T}_\rho$ not in the full Cantor space of configurations but in a subspace that is amenable to the methods of classical computability theory as presented in, say, [37,41]. As we have seen in sections 12.2.2 and 12.2.3, arguably the most plausible choice is the set of ultimately periodic configurations. Further evidence for the naturalness of this choice is given by Cook's result [10] that, in the bi-infinite case, there is an elementary cellular automaton capable of universal computation. Cook's proof of computational universality uses ultimately periodic configurations. Interestingly, the construction seems to rest on the ability to have two different periodic blocks, one extending to the left and the other to the right: in the construction, the left block serves to time the computation whereas the right block encodes the cyclic tag-system that is essential for universality. By contrast, Reachability for rule 110 is trivially decidable for configurations of finite support.

It has since been shown by Neary and Woods that the exponential slow-down inherent in the original construction can be avoided entirely, so that the simulation is actually quite efficient, at least from the perspective of complexity theory, see [32,33]. This has the consequence that it is $\mathbb{P}$-complete to determine the state of a particular cell at time t of the evolution of a finite configuration under rule 110.

From now on, let us assume that the configurations in our structures, infinite or bi-infinite, are always ultimately periodic. It is obvious that in both cases Reachability is undecidable in general. Significantly more effort shows that one can carefully control the evolution of configurations on the cellular automaton in question so as to make sure that the complexity of Reachability is an arbitrarily chosen recursively enumerable degree, see [41]. Of course, at heart the construction rests on the ability of a one-dimensional cellular automaton to simulate Turing machines. Alas, there are two substantial problems to overcome. First, hardness results for Turing machines are always phrased in terms of instantaneous descriptions, snap-shots of a computation that actually occur. By contrast, we have to deal with all configurations of the cellular automaton, most of which are meaningless from the perspective of the Turing machine. Second, we need to make sure that there are no unintended simulations that could drive the degree of Reachability up to, say, r.e.-completeness. It was shown in [42,45,46] and theorem 9 below how to cope with this problems. In summary, we can derive the following result.

Theorem 8. *The Reachability problem of an infinite or bi-infinite cellular automaton over $\mathscr{C}_{\mathsf{up}}$ can be chosen to be any r.e. degree.*

The theorem suggests that, if one is interested in computational properties, as opposed to the more classical dynamical systems aspects, one should consider a more fine-grained hierarchy based on the complexity of Reachability. The resulting classification is again highly undecidable: testing whether a cellular automaton has Reachability problem of degree $\mathbf{d}$ is $\Sigma_3^{\mathbf{d}}$-complete. In particular, testing for computational universality is Σ_4^0-complete. As it turns out, no constraints arise if we were to choose to base our classification on both Reachability and Confluence: two configurations are confluent if their orbits overlap. Thus, confluence is an equivalence relation and corresponds vaguely to basins of attraction. It is clear from the definitions that Confluence also has r.e. degree, but it is quite surprising that there is no connection between the two: given arbitrary r.e. degrees $\mathbf{d}_1$ and $\mathbf{d}_2$ there is a cellular automaton whose Reachability and Confluence problems have exactly those two chosen degrees as their complexity. It remains to be seen how other more complicated properties fit into this pictures.

As is well-understood, the semi-lattice of the r.e. degrees exhibits a rather complicated structure. For example, by a famous theorem of Sacks, the partial order of the r.e. degrees is dense: whenever $A <_T B$ for two r.e. sets A and B there is a third r.e. set such that $A <_T C <_T B$, [38]. Here $<_T$ denotes Turing reducibility. The first order theory of this semi-lattice is highly undecidable [17].

More complicated sentences in the first-order logic of $\mathfrak{T}_\rho$ will, in general, yield undecidable versions of the data model checking problem. For example, the nilpotency statement $\exists y \forall x (x \xrightarrow{*} y \wedge y \to y)$ is undecidable according to [21]. By interchanging quantifiers we obtain the "fixed point" sentence $FP \equiv \forall x \exists y \, (x \xrightarrow{*} y \wedge y \to y)$ from above. For finite cellular automata deciding FP is Π_1^0-complete, but for infinite and bi-infinite ones it is harder, given a slight restriction of the configuration space. For simplicity, let us only consider the bi-infinite automata, the argument is entirely similar in the one-way infinite case. Choose a particular symbol \$ in the alphabet and define $\mathcal{C}_\$ \subseteq \mathscr{C}_{\mathsf{up}}$ to be the class of all ultimately periodic configurations that contain \$ infinitely often in both directions. In other words, in the representation (u,v,w) both u and w contain \$. This property is easy to check by a pair of Büchi automata, so we are dealing with an automatic subspace.

Theorem 9. *Deciding FP for an infinite cellular automaton is Π_2^0-complete on $\mathcal{C}_\$$.*

For the proof first note that the problem lies in Π^0_2, since we only consider ultimately periodic configurations and these can be coded naturally as triples (u,v,w) of finite words, representing ${}^\omega uvw^\omega$. For hardness, recall that *INF*, the collection of all indices of infinite r.e. sets is well-known to be Π^0_2-complete, see [41]. Now $e \in \mathit{INF} \iff \forall n \exists m (n < m \wedge m \in W_e)$ where n and m range over the naturals. The construction of the cellular automaton ρ_e begins with a Turing machine M_e with binary tape that, on input 1^n dovetails on computations $m \in W_e$ for all $m > n$. More precisely, at stage s of the construction M_e will perform s steps in the computation of $m \in W_e$ for all $n < m < n+s$. If any of these computations converges, M_e halts.

A standard simulation of the Turing machine M_e by a cellular automaton will not produce the desired results since there are configurations of the cellular automaton that do not translate into instantaneous descriptions of M_e. For example, there might be several cells indicating state and head position of M_e; in fact, there could be infinitely many such cells due to the periodicity of the blocks on either side. A significantly more difficult problem is that a configuration may syntactically look like an instantaneous description of the Turing machine, but may actually not appear in any computation of M_e on any input 1^n. For example, it might contain an inaccessible state of the Turing machine M_e, which state then launches a divergent computation despite of the fact that the actual machine on input 1^n would halt. Let us refer to an instantaneous description as *admissible* if it occurs during a computation on some input. Needless to say, accessibility is not decidable in general but one can convert M_e into an equivalent Turing machine M'_e that is *stable* in the sense that it ultimately halts on all inadmissible descriptions, see [45, 49] for a more detailed description. The idea is to modify the machine so that it becomes self-verifying: retraces its steps every so often to try to verify that the current instantaneous description is indeed admissible. To this end, the Turing machine keeps a copy of the alleged original input throughout the computation. Roughly, we consider instantaneous descriptions of the form

$$ {}^\omega \underline{b}\#x\#s\#D\#t\#E\#\underline{b}^\omega $$

where x, s and t are numbers written in unary and D and E are instantaneous descriptions of M_e, the original machine. The symbol $\underline{b}$ is the blank symbol for M_e and # is a new separator symbol. The idea is that x is the original input to M_e and that instantaneous description D occurs after s steps in the computation. The remaining fields are used for the admissibility test, t as a counter and E as a corresponding instantaneous description.

As to the actual cellular automaton that simulates the self-verifying Turing machine, consider the alphabet

$$ \Gamma = \underset{\rightarrow}{\Sigma} \cup Q \cup \underset{\leftarrow}{\Sigma} \cup \{\$\}. $$

As usual, Q denotes the state set of the stable Turing machine and $\underset{\rightarrow}{\Sigma}$ and $\underset{\leftarrow}{\Sigma}$ are copies of the tape alphabet modified by additional direction bits that indicate whether the symbol is to the right or to the left of the state symbol. The special symbol \$ is the one that is required to appear infinitely often in both directions and serves the purpose of limiting the part of the configuration that can code instantaneous descriptions. Any local configurations that represent syntactically incorrect descriptions are fixed points; for example, two adjacent symbols from $\underset{\rightarrow}{\Sigma}$ and $\underset{\leftarrow}{\Sigma}$ will stay fixed. Similarly two adjacent symbols from Q do not change. Changes do occur at local configurations of the form $\underset{\rightarrow}{\Sigma} \times Q \times \underset{\leftarrow}{\Sigma}$, except when the state in Q is the halting state of the stable Turing machine. The special symbol \$ can be converted into a symbol in $\underset{\rightarrow}{\Sigma} \cup Q \cup \underset{\leftarrow}{\Sigma}$ in the presence of the state head. As a consequence, the only orbits free of "frozen" components are the ones that correspond to instantaneous descriptions of M'_e, though a priori not necessarily admissible ones and not necessarily single ones either. But since M'_e is stable, inadmissible descriptions will be recognized and lead to the Turing machine halting. At the configuration level we obtain a fixed point. Thus, the only situation where a configuration can not evolve to a fixed point is when we are in fact simulating a divergent computation of M_e. But then $e \in INF$ if, and only if, the sentence FP holds over the structure $\mathfrak{C}_{\rho_e}$.

We note that the argument relies quite heavily on the fact that we consider configurations in $\mathcal{C}_{\$}$, for general ultimately periodic configurations there appears to be no way to organize the self-testing mechanism for the intermediate Turing machine. To wit, a non-fixed point computation could occur because of an ultimately periodic configuration of the cellular automaton that corresponds to an infinitely instantaneous description of the Turing machine.

12.4 Summary

We have shown that a small part of the classification of one-dimensional cellular automata can be handled automatically by using ideas from model checking. Unfortunately, this approach leads to considerable efficiency problems and it is not clear at present how far the corresponding decision algorithms can be pushed. Efforts are under way to give a local classification for elementary cellular automata on one-way infinite grids, but even in this relatively simply case there are problems dealing with the corresponding Büchi automata. In particular, determinization using Safra's classical algorithm often disrupts the decision algorithm. It is safe to assume that the problems become virtually unsurmountable when one moves on to bi-infinite automata. One could try use compact representations such as binary decision diagrams to try to cope with large state spaces [6, 7], but first experiments in this direction have been less than compelling.

Needless to say, any plausible classification scheme for cellular automata will have to go further and include aspects of the long-term evolution of configurations. Even in the one-dimensional case, assertions about long term evolution tend to be undecidable and fully automatic classification is simply impossible. On the other hand, as shown conclusively by Cook, simulation tools can help to sharpen one's intuition and provide sufficient insights into the behavior of a particular cellular automaton to complete classification "by hand." This is somewhat similar to the current situation in theorem proving and formally verified mathematics: fully automatic tools can cover only so much ground, but with sufficient intervention from the user, proof assistants can produce quite interesting results, see the recent [3]. Some efforts in this direction can be found in [1, 51, 55]. One wonders whether crowd-sourcing might be helpful; in astronomy, classification of galaxies by large numbers of amateur volunteers has produced spectacular results, see `http://www.galaxyzoo.org/`. For example, the pseudo-random behavior of elementary cellular automaton rule 30 seems to make it a hopeless undertaking to encode any kind of undecidability proof. Perhaps many eyes could find enough structure to support such an argument.

One natural challenge is to determine the degree of the full first-order theory of $\mathfrak{T}_\rho$, perhaps first over some specialized configuration space such as $\mathcal{C}_\$$. One suspects that any level in the arithmetic hierarchy can be represented by a suitable assertion about orbits of a cellular automaton. Note, though, that the preceding construction differs in significant technical details from the result in [49] that shows that Reachability has arbitrary r.e. degree. In particular, a special annihilator symbol $\perp$ is used there to reduce the complexity of orbits of syntactically incorrect configurations to being decidable–the corresponding light cones of $\perp$ cells would break the current argument. Hence, it is not entirely clear how to generalize this kind of argument to longer chains of arithmetic quantifiers. Lastly, there is the question of the degree of the full first-order theory of $\mathfrak{T}_\rho$ over $\mathscr{C}_{\mathsf{up}}$ or the full configuration space.

References

1. Adamatzky, A.: Identification of Cellular Automata. Taylor & Francis, London (1994)
2. Amoroso, S., Patt, Y.N.: Decision procedures for surjectivity and injectivity of parallel maps for tesselation structures. Journal of Computer and Systems Sciences 6, 448–464 (1972)
3. Avigad, J., Harrison, J.: Formally verified mathematics. Comm. ACM 57(4), 66–75 (2014)
4. Bartholdi, L., Silva, P.V.: Groups defined by automata. CoRR, abs/1012.1531 (2010)
5. Blumensath, A., Grädel, E.: Automatic structures. In: Proc. 15th IEEE Symp. on Logic in Computer Science, pp. 51–62. IEEE Computer Society Press (1999)
6. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers C-35(8), 677–691 (1986)
7. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98(2), 142–170 (1992)
8. Chang, C.C., Keisler, H.J.: Model Theory. In: Studies in Logic and the Foundations of Mathematics, Elsevier (1990)
9. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (2000)
10. Cook, M.: Universality in elementary cellular automata. Complex Systems 15(1), 1–40 (2004)
11. Culik, K., Yu, S.: Undecidability of CA classification schemes. Complex Systems 2(2), 177–190 (1988)
12. Epstein, D.B.A., Cannon, J.W., Holt, D.F., Levy, S.V.F., Patterson, M.S., Thurston, W.P.: Word Processing in Groups. Jones and Bartlett, Burlington (1992)
13. Finkel, O.: On decidability properties of one-dimensional cellular automata. Computing Research Repository, abs/0903.4615 (2009)
14. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman (1979)

15. Grigorchuk, R., Šunić, Z.: Self-Similarity and Branching in Group Theory. In: Groups St. Andrews 2005. London Math. Soc. Lec. Notes, vol. 339. Cambridge University Press (2007)
16. Grigorchuk, R.R., Nekrashevich, V.V., Sushchanski, V.I.: Automata, dynamical systems and groups. Proc. Steklov Institute of Math. 231, 128–203 (2000)
17. Harrington, L., Shelah, S.: The undecidability of the recursively enumerable degrees. Bull. Amer. Math. Soc. 6, 79–80 (1982)
18. Hedlund, G.A.: Endomorphisms and automorphisms of the shift dynamical system. Math. Systems Theory 3, 320–375 (1969)
19. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge, UP (2000)
20. Kari, J.: Reversibility of 2D cellular automata is undecidable. Physica D 45, 379–385 (1990)
21. Kari, J.: The nilpotency problem of one-dimensional cellular automata. SIAM J. Comput. 21(3), 571–586 (1992)
22. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
23. Khoussainov, B., Rubin, S.: Automatic structures: overview and future directions. J. Autom. Lang. Comb. 8(2), 287–301 (2003)
24. Kupferman, O.: Avoiding determinization. In: Proc. 21st IEEE Symp. on Logic in Computer Science (2006)
25. Kurka, P.: Languages, equicontinuity and attractors in cellular automata. Ergodic Th. Dynamical Systems 17, 417–433 (1997)
26. Kurka, P.: Topological and Symbolic Dynamics. Number 11 in Cours Spécialisés. Societe Mathematique de France, Paris (2003)
27. Li, W., Packard, N.: The structure of the elementary cellular automata rule space. Complex Systems 4(3), 281–297 (1990)
28. Li, W., Packard, N., Langton, C.G.: Transition phenomena in CA rule space. Physica D 45(1-3), 77–94 (1990)
29. Lind, D.: Multi-dimensional symbolic dynamics. In: *Symbolic Dynamics and its Applications*. Proc. Sympos. Appl. Math., vol. 60, pp. 61–79. AMS (2004)
30. Margenstern, M.: Frontier between decidability and undecidability: a survey. TCS 231, 217–251 (2000)
31. Meyers, R.A. (ed.): Encyclopedia of Complexity and System Science. Springer, Berlin (2009)
32. Neary, R., Woods, D.: On the time complexity of 2-tag systems and small universal turing machines. In: FOCS, pp. 439–448. IEEE Computer Society, Washington (2006)
33. Neary, T., Woods, D.: P-completeness of cellular automaton rule 110. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 132–143. Springer, Heidelberg (2006)
34. Nekrashevych, V.: Self-Similar Groups. In: Math. Surveys and Monographs, vol. 117. AMS (2005)
35. Perrin, D., Pin, J.-E.: Infinite Words. In: Pure and Applied Math., vol. 141, Elsevier, Amsterdam (2004)
36. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM Jour. Research 3(2), 114–125 (1959)
37. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)
38. Sacks, G.E.: The recursively enumerable degrees are dense. Ann. Math. 80, 300–312 (1964)

39. Safra, S.: On the complexity of ω-automata. In: Proc. 29th FOCS, pp. 319–327. IEEE Computer Soc. Press, Washington (1988)
40. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press (2009)
41. Soare, R.I.: Recursively Enumerable Sets and Degrees. In: Perspectives in Mathematical Logic. Springer, Berlin (1987)
42. Sutner, K.: A note on Culik-Yu classes. Complex Systems 3(1), 107–115 (1989)
43. Sutner, K.: Classifying circular cellular automata. Phys. D 45(1-3), 386–395 (1990)
44. Sutner, K.: De Bruijn graphs and linear cellular automata. Complex Systems 5(1), 19–30 (1991)
45. Sutner, K.: Cellular automata and intermediate degrees. Theoretical Computer Science 296, 365–375 (2003)
46. Sutner, K.: Universality and cellular automata. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 50–59. Springer, Heidelberg (2005)
47. Sutner, K.: Encyclopedia of Complexity and System Science, chapter Classification of Cellular Automata. In: Meyers [31], (2009)
48. Sutner, K.: Model checking one-dimensional cellular automata. J. Cellular Automata 4(3), 213–224 (2009)
49. Sutner, K.: Cellular automata, decidability and phasespace. Fundamenta Informaticae 140, 1–20 (2010)
50. Sutner, K., Lewi, K.: Iterating invertible binary transducers. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 294–306. Springer, Heidelberg (2012)
51. Vorhees, B.: Computational Analysis of One-Dimensional Cellular Automata. World Scientific, Singapore (1996)
52. Wolfram, S.: Computation theory of cellular automata. Comm. Math. Physics 96(1), 15–57 (1984)
53. Wolfram, S.: Twenty problems in the theory of cellular automata. Physica Scripta T9, 170–183 (1985)
54. Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign (2002)
55. Wuensche, A.: Classifying cellular automata automatically. Complexity 4(3), 47–66 (1999)

Chapter 13
Algorithms with Active Cells Modeled by Cellular Automata with Write-Access (CA-w)

Rolf Hoffmann

Abstract. The CA-w model (cellular automata with write-access) was introduced in order to simplify the description of problems with dynamic activities, like moving agents or active particles in a grid of cells. This model allows an active cell to send data to a neighboring cell. Thereby neighbors can be activated or deactivated, or the movement of agents can directly be described. The usefulness of the model is demonstrated for basic computational problems, the well-known 1-D CA traffic rule, Pascal's triangle, Fibonacci numbers, sorting on the ring by agents, and leader election for agents.

13.1 Introduction

A special class of distributed algorithms can be defined by using *active cells*, also called *active particles* [12] or *agents*, which work together in parallel in order to find an aimed final global configuration. In many applications only a part of all cells in the environment are active cells, and their number may dynamically change over time, and some cells may behave like moving agents. The question here is how such algorithms can be described in a natural and concise way. One way is to use the Cellular Automata (CA) model. Although such algorithms can be described by CA, the description often becomes complicated, artificial, and redundant. The main reason is that in CA a cell can only read information from its neighbors and cannot change its neighbors' states directly. Therefore the movement of agents, the change of neighboring data, and the control of activity becomes difficult to describe. The solution presented here is to use the **CA-w** (local version of the **GCA-w** model) instead of the CA model.

Rolf Hoffmann
Technische Universität Darmstadt, FG Rechnerarchitektur,
Hochschulstr. 10, 64289 Darmstadt, Germany
e-mail: `hoffmann@ra.informatik.tu-darmstadt.de`

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_13

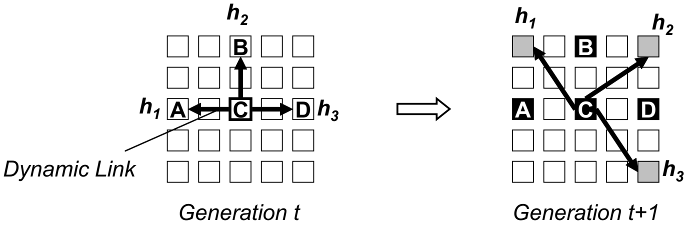


Fig. 13.1 A *2-D* GCA-w example with three links per cell: Each cell is dynamically connected to a set of global neighbors, only the activity of the center cell C is shown. The state of C including the links h_i, and the states of its neighbors can be changed (white to black) by a local rule. Thereby information can be transferred from C to the current neighbors A, B, D, and the activity of the neighbors may be changed. Write conflicts may occur, e.g. if C itself and other cells try to update C at the same time.

The **GCA-w** parallel computing model (*GCA with write-access*) introduced in [6, 7] is an extension of the **GCA** (*Global Cellular Automata*) model [3,4]. If the neighborhood of the GCA-w model is *locally restricted*, we will call the model **CA-w** (*Cellular Automata with write-access*) [8].

A GCA cell can *dynamically* establish links to any of its *global* neighbors, whereas a CA cell can only use the *fixed* links to its *local* neighbors. CA and GCA do not allow to modify the state of a neighbor. Therefore no write-conflict can occur, simplifying implementations in hardware and in software. However, for applications where the amount of active cells in the whole field is low or is varying over time, or the locations of the active cells are changing, the GCA-w model (with global dynamic write-neighbors) or the CA-w model (with local dynamic write-neighbors) are a better choice. These models allow a cell to write information to its neighbors. This feature is very important because information can actively be transferred to a destination, and the activity of the destination can be switched on or off. Therefore, write-access is very useful for the description of problems with moving particles or moving agents, or problems with dynamic activities.

Several GCA-w applications were already described in [5–7] (one-to-all communication, synchronization, moving agents, different random walks of particles, pointer inversion, sorting with pointers, traffic simulation). And it was already shown in [8] that problems with local movements (like rotor-routing and chip-firing) can be described adequately by the CA-w model. The purpose of this contribution is to show that other interesting computational problems can easily be modeled by CA-w. In the following two sections the properties of the GCA-w and CA-w models are reviewed.

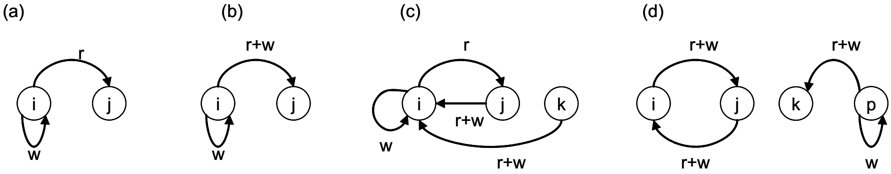


Fig. 13.2 Some read / write situations: (a) Cell *i* reads from cell *j* and modifies its own state. (b) Cell *i* reads *j* and modifies its own state and the state of its neighbor *j* (the basic idea of GCA-w). (c) Cell *i* modifies itself and is modified by cell *j* and *k*, but there occurs a conflict of three write-accesses which has to be resolved. (d) Cell *i* modifies *j* and vice versa. Cell *p* modifies cell *k* and *p*. No write conflicts occur (exclusive write situation).

13.1.1 Global Cellular Automata with Write-Access (GCA-w)

Fig. 13.1 shows the general idea of the GCA-w model: Each cell C is dynamically connected via links (also called *hands* [1]) h_i to other cells, in the example to A, B, D. Cell C updates its own state and the states of its neighbors A, B, D. Thereby the links h_i are also updated. In the following, only one link per cell is assumed, which seems to be sufficient to model most applications. In addition, in the following, the cells will be arranged in an *1-D* fashion, although *n-D* fields can easily be handled by using an *n-D* indexing scheme.

A potential difficulty is that write-conflicts may appear. The worst case scenario is that all cells want to write onto the same cell. In order to reduce the implementation effort to resolve conflicts, the GCA-w rules should be designed in such a way that either no conflict will occur (which is the case for the traffic rule, Sect. 13.2.1), or that the amount of conflicts is low or restricted, or that the conflicts can be resolved within a local neighborhood.

In Fig. 13.2 some situations are depicted, showing some possible interactions (reads and writes) between cells. In situation the cell *i* reads *j*, and modifies only its own state like in CA. In situation (b) cell *i* reads *j*, and modifies its own state and the state of its neighbor *j* (the basic idea of GCA-w). In situation (c) cell *i* modifies itself and is modified by *j* and *k*, a conflict with three write-accesses occurs that has to be resolved. Situation (d) is an “exclusive-write situation”: Cells *i* and *j* exchange data, and cell *p* modifies cell *k* and *p*; no write conflicts occur.

In the following the model is formally defined.

13.1.1.1 Formal Description

$$\text{GCA-w } = (I, Q, \delta, h, f, g, e) \tag{1}$$

Index Set, unique labels identifying the cells:

$$I = \{0, 1, \ldots, i, \ldots, N\text{-}1\} \tag{2}$$

[1] If k access pointers are used, then we call the GCA-w “$k-handed$”; this terminology was already used for the GCA model [3].

States: $Q = A \cup P \cup D$ (3)

Active States: $A = \{a_1, a_2, \ldots, a_n\}$ (4)

Passive States: $P = \{p_1, p_2, \ldots, p_m\}$ (5)

Dead States: $D = \{d_1, d_2, \ldots, d_k\}$ (6)

Don't-Write-Symbol: δ

Write-Values: $Q_\delta = Q \cup \{\delta\}$ (7)

Configurations: Q^N (8)

A Configuration: $L = (q_0, q_1, \ldots, q_i, \ldots, q_{N-1}) \in Q^N, i \in I$ (9)

Neighbor's Address (in the case of absolute addressing) $h(i,q)$:
$h : I \times Q \to I$ (10)

Neighbor's Address (in the case of relative addressing) $h_{rel}(i,q)$:
$h_{rel} : I \times Q \to I_{rel} = \{0, \pm 1, \pm 2, \ldots, \pm(N\text{-}1)\}$ (11)
such that $h(i,q) = i + h_{rel}$ [2] (12)

Local-Rule $f(i,q,q^*)$, where $q^* =$ *neighbor's state*:
$f : I \times Q \times Q \to Q_\delta$ (13)

Write-Rule $g(i,q,q^*)$:
$g : I \times Q \times Q \to Q_\delta$ (14)

Conflict-Rule $e(i, f, g^0, g^1, \ldots, g^j, \ldots, g^{N-1})$:
$e : I \times Q_\delta^{N+1} \to Q_\delta$ (15)

Rule Application (synchronous updating) $\forall i \in I$:

$$q_i \leftarrow \begin{cases} e(i, f(i, q_i, q_{h(i,q_i)}), g^0_{\to i}, .., g^j_{\to i}, .., g^{N\text{-}1}_{\to i}) & \text{IF } q_i \in A \wedge e \neq \delta \qquad (16)\\ q_i & \text{IF } q_i \in D \vee e = \delta \qquad (17)\\ e(i, \delta, g^0_{\to i}, \ldots, g^{j=i}_{\to i} = \delta, \ldots, g^{N\text{-}1}_{\to i}) & \text{IF } q_i \in P \wedge e \neq \delta \qquad (18) \end{cases}$$

where $\forall (i,j) \in I \times I$:

$$g^j_{\to i} = \begin{cases} g(j, q_j, q_{h(j,q_j)}) & \text{IF } \; h(j,q_j) = i \wedge q_j \in A \qquad (19)\\ \delta & \text{IF } \; h(j,q_j) \neq i \vee q_j \notin A \qquad (20) \end{cases}$$

The cells are arranged as a sequence $\langle a_i \rangle_{i \in I}$ of cells, each cell is labeled by its index i (1). The index can also be accessed by the cell itself in order to implement non-uniform rules. A cell can be in an *active* state (4), in a *passive* state (5), or in a *dead* state (6). These three classes of states are also called *operational states*. Operational states are introduced in order to switch on or off the activity of the cells, and thereby allowing to reduce the computational effort. Dead cells are totally excluded from the computation because they remain dead forever. Passive cells do

[2] All addresses are mapped onto the interval $0..N-1$ by the modulo operation *mod N*.

not compute themselves but can be activated by other cells. In addition, passive or dead cells can be used to define a termination condition, e.g. if all cells become dead, or if all cells become passive.

If a cell is active, it computes all the local functions h, f, g, e. If a cell is passive, it does not compute its local functions h, f, g, but it can be switched into another operational state from outside, e.g., it can be changed into active. If a cell is dead, it will stay dead forever (it can be used as a constant).

The symbol delta ("Don't-Write") is a special output value of the functions f, g, and e. As a possible input of the conflict rule e it signals that no valid value is received from another cell for writing. In the case that the conflict rule does not receive any valid input ($\neq \delta$), it produces δ, too. Then the cell's state will remain unchanged.

The address function h defines the actual neighbor (absolute address, index) in access (read and write) (10). The neighbor's address can also be determined by the use of the relative addressing function h_{rel} (11). Relative addressing is often more adequate and more general to describe spatial relative situations. The local rule f computes the cell's new state if no neighbor is writing additionally. The *write-rule* computes a state value that can be written to the actual neighbor. The *conflict-rule* is dedicated to resolve the conflicts. It receives *N+1* messages (write-values), the own rule value f and messages $g^j_{\to i}$ from all cells j.

Not all messages need to be valid: A non-valid δ-message is interpreted as a "*Don't-Write*"- command, meaning that a sender j does not want to write to a receiver i. A message is valid if the sender j is active and the receiver i is selected (18). Note that δ may be produced by g, f, e. If the sender is passive or the receiver is not selected, then δ is received by default.

Rule Application. All cells are updated synchronously in parallel. Only cells that are not dead can be updated. If a cell is active (16) then the own rule's value f and the messages $g^j_{\to i}$ from all cells j have to be taken into account. In case that a δ–message is received, it is disregarded by the conflict rule e.

If a cell is passive (18) then no computation of h, f, g takes place and the default values $f = \delta$ and $g = \delta$ are assumed, and no neighbor is selected. Although the cell is passive, the conflict-rule e has to be awake because activating messages might arrive.

At a first glance the GCA-w model seems to be too complex because of the conflicts and not very useful. But in many applications the complexity of the conflict resolution can be reduced significantly, e.g. if the dynamic neighborhood access patterns are restricted or are known in advance, or if the rules fulfill the *exclusive-write* condition (as in the following algorithms), or if the conflict resolution corresponds to a reduction operator (e.g. summing up the inputs). The main advantage of the GCA-w model is that it allows to describe a certain class of algorithms more natural and concise, less redundant and energy saving (only the active cells are computing). The novel idea is to organize the computational task logically as an array of cells with different operational states (active, passive, dead) with write-access onto dynamically selected neighbors by the use of local rules only.

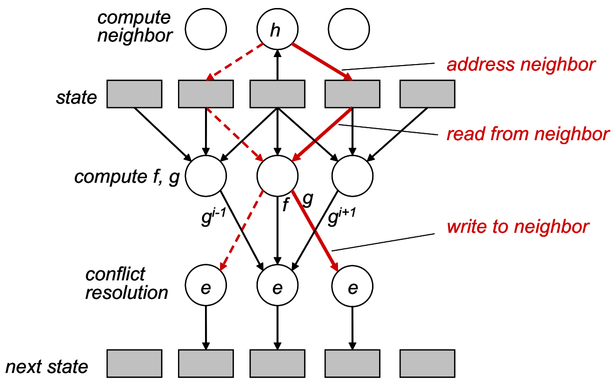


Fig. 13.3 *1-D* CA-w model with one hand. The right neighbor is selected in this example. The functions f, g are computed, the value g is sent to the right neighbor. The center cell receives g^{i-1} from the left, and g^{i+1} from the right. These messages are taken into account by the conflict resolution function e.

13.1.2 Cellular Automata with Write-Access (CA-w)

We will call the GCA-w model "*CA-w*" (*Cellular Automata with write access*) if the neighborhood is locally restricted. As in the GCA-w model, the CA-w model may use k "hands" to access k neighbors in parallel, then we call the CA-w *k-handed*. Each hand can read from and / or write to a dynamically selected neighbor. Many applications can be described with one hand only. The formal description of an *1-D* CA-w with one hand is the following (only the formulas differing from the ones in Sect. 1.1 are given):

Cellular Automata with Write Access:

$$\text{GCA-w } = (I, Q, \delta, h, f, g, e) \tag{1}$$

Neighbor's Address (in case of absolute addressing) $h(i,q)$:

$$h : I \times Q \to I, \ \ (i-R) \le h' \le (i+R), h = h' \ mod\ N \tag{10}$$

Neighbor's Address (in case of relative addressing) $h_{rel}(i,q)$:

$$h_{rel} : I \times Q \to I_{rel} = \{0, \pm 1, \pm 2, \ldots, \pm R\} \tag{11}$$

$$\text{such that } h(i,q) = (i + h_{rel})\ mod\ N \tag{12}$$

Conflict-Rule $e(i, f, g^{i-R}, \ldots, g^{i-1}, g^{i}, g^{i+1}, \ldots, g^{i+R})$:

$$e : I \times Q_{\delta}^{2R+1} \to Q_{\delta} \tag{15}$$

Rule Application (synchronous updating) $\forall i \in I$:

$$q_i := \begin{cases} e(i, f(i, q_i, q_{h(i,q_i)}), g_{\to i}^{i-R}, \ldots, g_{\to i}^{i+R}) & \text{IF } q_i \in A \wedge e \neq \delta \quad (16) \\ q_i & \text{IF } q_i \in D \vee e = \delta \quad (17) \\ e(i, \delta, g_{\to i}^{i-R}, \ldots, g_{\to i}^{j=i} = \delta, \ldots, g_{\to i}^{i+R}) & \text{IF } q_i \in P \wedge e \neq \delta \quad (18) \end{cases}$$

In this definition it is assumed that the neighborhood is symmetric, the neighbor's address lies within a certain radius R (10, 11), and exactly one neighbor is selected. But in general, a set of possible neighborhoods is given (e.g. one or two cells ahead in the moving direction of an agent), one of them being dynamically selected by a neighborhood index. It is also possible, that the actual access-right (read, write, read and write) differs from hand to hand (if the CA-w has multiple hands).

Inputs of the conflict rule (15) are the own function f and the possible messages $g^{i-R}, \dots, g^{i+R}$ from the neighbors. The next state q_i at time $t+1$ is given by the output of e (16, 17, 18).

A more specific *1-D* case (see Fig. 13.3 as an example): The radius is $R = 1$ and only the left neighbor $(i-1)$ or the right neighbor $(i+1)$ can be addressed. (The access to the own cell i is excluded here because the own state is available through the function f.) Then the definition simplifies to

Conflict-Rule $e(i, f, g^{i-1}, g^{i+1})$:

$$e : I \times Q_\delta^3 \to Q_\delta \tag{15}$$

Rule Application (synchronous updating) $\forall i \in I$:

$$q_i := \begin{cases} e(i, f(i, q_i, q_{h(i,q_i)}), g_{\to i}^{i-1}, g_{\to i}^{i+1}) & \text{IF } q_i \in A \wedge e \neq \delta \quad (16) \\ q_i & \text{IF } q_i \in D \vee e = \delta \quad (17) \\ e(i, \delta, g_{\to i}^{i-1}, g_{\to i}^{i+1}) & \text{IF } q_i \in P \wedge e \neq \delta \quad (18) \end{cases}$$

As result, in the *1-D* CA-w model with one hand and with read/write-radius 1, there are only three inputs of the conflict rule: f (own function), $g_{\to i}^{i-1}$ (message from left neighbor), and $g_{\to i}^{i+1}$ (message from right neighbor). Analyzing the data flow depicted in Fig. 13.3, it can be observed that data from the neighbors within the radius 2 can influence the new state of the own cell. Therefore the CA-w model with radius 1 can be emulated by a CA model with radius 2.

13.1.3 Related Work

The PSA (Parallel Substitution Algorithm) model [1] of computation is a very general and powerful model based on *substitution rules*. It allows also to modify a set of arbitrary target cells (*right side* of the substitution) using a *base* and a *context*. In relation to the GCA-w the base corresponds to the cell(s) under consideration, the context corresponds to the read neighbors and the right side corresponds to the cells which are modified (usually a part of the base). There is also a relation to the CRCW-PRAM [9, 10] model. The PRAM model is based on a physical view with p processors that have global memory access to physical data words whereas the GCA-w is based on logical computing cells with local memory tailored to the application. Another difference of the GCA-w model compared to PRAM is the direct support of dynamic links and the rule based approach similar to the CA model.

13.2 CA-w Algorithms

13.2.1 Traffic Rule

(a)

L	C	R	C‘	operation		toggle
0	0	-	0	δ	no operation	0
-	1	1	1	δ	no operation	0
-	1	0	0	← 0	set(0)	1
1	0	-	1	← 1	set(1)	1

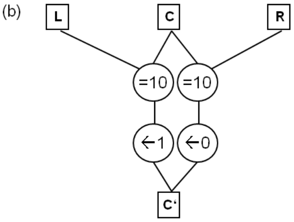


Fig. 13.4 Traffic Rule. (a) This table describes the new state C’ depending on the neighborhood LCR. The behavior can also be described by the set(0) and set(1) operations, or the toggle operation. (b) The graph shows how the new state is updated by set operations ($C' \leftarrow 1/0$).

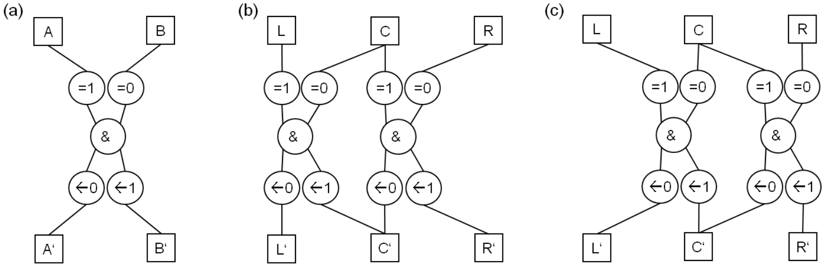


Fig. 13.5 Traffic Rule. (a) Substitution, the pattern 10 is substituted by 01. (b) Push rule, particles are active and are pushed to the right cell if it is empty. (c) Pull rule, empty cells are active and pull particles from left to right.

The elementary *1-D* CA rule 184, also known as traffic rule, describes the moving of particles or cars in one direction, as given by the table Fig. 13.4. In classical CA the new state is computed by the logical function $C' \leftarrow (C \wedge R) \vee (L \wedge \bar{C})$. If the cell’s state is 1, its new state is taken from the cell R to its right. Otherwise, its new state is taken from the cell L to its left. This means that the rule can be implemented by a multiplexer controlled by C: $C' \leftarrow if\ \ C\ \ then\ \ R\ \ else\ \ L.$

Another way to compute the next state is to use the operations: *set(value)* and δ (no operation). By default, δ is received by each cell, and in this case the cell remains in its old state. The traffic rule, described by set-operations only, is

$$C' \leftarrow \begin{cases} 1 & \text{if } LC = 10 \\ 0 & \text{if } CR = 10 \\ C & \text{otherwise } (\delta) \end{cases} .$$

Depending on who is the sender of the operations, four different cases can be distinguished.

(1) *CA Cell Rule.* As shown in Fig. 13.4 b, both operations (*set*(0), *set*(1)) are sent and received by the cell itself, according to the CA principle that a cell can change its own state only.
(2) *Substitution.* (Fig. 13.5 a) Both operations are enabled by a unit that can be imagined in between of each pair of cells (A, B). Thus, wherever the pattern 10 appears it is substituted by 01 without any conflict.
(3) *Push Rule.* (Fig. 13.5 b) Only cells containing particles are considered to be active. The value 1 is pushed (beamed, copied) by the active cell to the right neighbor. This can be interpreted in a way where the particle moves actively to the next free position. The operation *set*(1) is sent to the right neighbor, and the operation *set*(0) is performed on the cell itself. Although two different values can be accepted by a cell, there will be no conflict because the rule is inherently conflict-free.
(4) *Pull Rule.* (Fig. 13.5 c) Only empty cells are considered to be active. If an empty cell perceives a particle coming from left, it is copied (pulled) from L to C by the operation *set*(1), and the particle on L is deleted by sending *set*(0) to it. The pull rule is useful for applications where the particle is driven by an external field or force.

The push rule and pull rule show clearly that the CA-w model can be applied successfully to describe the movement of particles in an adequate way. The substitution (Fig. 13.5 a) can be shifted to the left cell, yielding the push rule, or shifted to the right, yielding the pull rule.

A program is presented for the *push rule*. Only the cells that are 1 are considered to be active.

```
TYPE cell = (data: BIT)                    // BIT = 0/1
C: ARRAY [0 .. n-1] OF cell

self <=> C[k], right <=> C[k+1]

REPEAT
   PARALLEL self WHERE (self.data = 1)     // only do for active cells
      IF (right.data=0) THEN
         self.data  <= 0
         right.data <= 1
      ENDIF
   ENDPARALLEL
ENDREPEAT
```

A program for the *pull rule* is the following. Only the cells that are 0 are considered to be active.

```
TYPE cell = (data: BIT)                    // BIT = 0/1
C: ARRAY [0 .. n-1] OF cell

self <=> C[k], left <=> C[k-1]

REPEAT
   PARALLEL self WHERE (self.data = 0)     // only do for active cells
      IF (left.data=1) THEN
```

```
         self.data <= 1
         left.data <= 0
      ENDIF
   ENDPARALLEL
ENDREPEAT
```

13.2.2 Pascal's Triangle

This well-known triangle computes the binomial coefficients. It can be drawn in different ways. Filling empty sites with zeros (Fig. 13.6 a), the computation can easily be formulated by the cellular automata rule: ($Center \leftarrow Left + Right$). However, redundant "zero" additions are performed. In order to avoid unnecessary zero additions, the aligned representation (Fig. 13.6 b) can be used. Filling the empty sites with zeros the computation can be described by the CA rule ($Center \leftarrow Left + Center$) more efficiently.

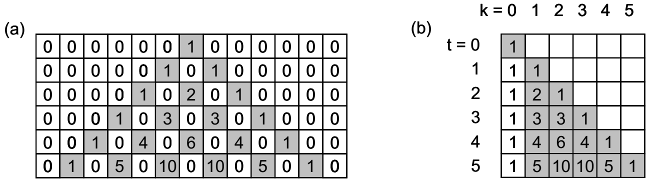
Note that in a CA with n cells always $(n-1)$ cells are updated, although in this example the interesting information is propagated step by step from left to right. Only in the last generation, all cells produce a useful result (except cell 0). On average, approximately only one third of the cells are producing the coefficients. Of course, by the use of a central control, the cells can increasingly be included into the computational process. But it is desired to control the amount of active cells by the cells themselves in a decentralized way. The CA-w model supplies this feature. The redundant production of zeros at the right part of a line can be avoided. In the following program only useful computations are performed.

For simplicity, we start at time step $t = 1$. Initially the two cells $(k = 1, 2)$ are set to one and only the cell $(k = 1)$ is active. Then two actions are performed repeatedly: (1) the rightmost active cell writes $d \leftarrow 1$ to its right neighbor and activates it, (2) each active cell adds the value of its left neighbor to its own value. Thereby, at every time step, a new cell active cell is generated at the right margin.

```
TYPE cell = (data: integer; active: (FALSE, TRUE))

C: ARRAY [0 .. n] of cell      // example n=4

self <=> C[k], left <=> C[k-1], right <=> C[k+1]    // cell and neighbors
```



(a)

0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	1	0	2	0	1	0	0	0	0
0	0	0	1	0	3	0	3	0	1	0	0	0
0	0	1	0	4	0	6	0	4	0	1	0	0
0	1	0	5	0	10	0	10	0	5	0	1	0

(b)

	k = 0	1	2	3	4	5
t = 0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

Fig. 13.6 Pascal's Triangle. (a) The value "0" is used to indicate the spaces. The value $x[k]$ in line t is the sum of $x[k-1] + x[k+1]$ of line $(t-1)$, corresponding to the *1–D* CA rule: $Center \leftarrow Left + Right$). (b) Aligned representation, the result of deleting the "0"s from Fig. 13.6 a and shifting the remaining numbers to the left border.

```
INITIAL
   C[0].data <= 1, C[1].data <= 1
   C[1].active <= TRUE
   C[/=1].active <= FALSE
ENDINITIAL1

REPEAT
   PARALLEL self WHERE (self.active)     // only do for active cells
      self.data <= self.data + left.data
      IF (self.data = 1) THEN right.data <= 1, right.active <= TRUE ENDIF
   ENDPARALLEL
ENDREPEAT
```

13.2.3 Fibonacci Numbers

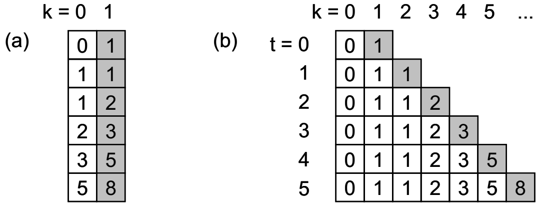


Fig. 13.7 Fibonacci Numbers. (a) The cell $k = 1$ is always the only active cell (shaded). The new value at $k = 0$ in line $t+1$ is a copy of the old value at $k = 1$ from line t. The new value at $k = 1$ in line $t+1$ is the sum of the old values from line t. The Fibonacci numbers appear sequentially in column $k = 1$. (b) The active cell at the right edge is moving step by step to the right $(k = 1, 2, ...)$. When the active cell k is moving to the right, it writes also the sum of its own $value(k)$ and the $value(k-1)$ of its left neighbor to the new position $k+1$.

There are several ways to describe the time evolution of the Fibonacci Numbers by CA-w. Two possibilities are shown in Fig. 13.7. As shown in Fig. 13.7 a there are only two cells, the right of them is the only active one. It copies its own value to its left neighbor at $k = 0$, computes the sum of the value of its left neighbor and its own value and writes it to itself.

```
      left.data <= self.data
      self.data <= self.data + left.data
```

In order to let the numbers appear in spatial and not in temporal order, the active cell is moving step by step to the right (Fig. 13.7 b). It computes the sum of the value of its left neighbor and its own value and writes it to the right neighbor, thereby activating the right neighbor and deactivating itself.

```
TYPE cell = (data: integer; active: (FALSE, TRUE))
C: ARRAY [0 .. ] of cell

\clearpage{}

self <=> C[k], left <=> C[k-1], right <=> C[k+1] //active cell and neighbors

INITIAL
```

```
   C[0].data <= 0
   C[1].data <= 1
   C[1].active <= TRUE
   C[/=0].active <= FALSE
ENDINITIAL

REPEAT
   PARALLEL self WHERE (self.active)        // only do for active cells
      self.active  <= FALSE
      right.active <= TRUE
      right.data   <= self.data + left.data
   ENDPARALLEL
ENDREPEAT
```

13.2.4 Sorting on the Ring with Agents

The goal is to sort *N* numbers placed on a ring (*1-D* array of *N* cells with cyclic boundary) using agents (Fig. 13.8). We will call an algorithm which is performed by agents "agent algorithm" (*a-algorithm*). We may distinguish between the *local* agent algorithm which is performed by each agent, and the *global* agent algorithm which is performed by all agents together.

Each agent can be seen as a small processor, which often is mobile, but this is not a necessary condition. By moving around, an agent can interact with the environment including the other agents. We call an array of cells with agents situated on them "Multi-agent System" (MAS). A MAS can be classified as a distributed system [13].

The objectives and constraints for the sorting task are

- Find an agent algorithm that sorts *N* numbers placed on a ring.
- The algorithm shall not depend on global information.
- The algorithm shall work with any number *k* of agents.
- The initial direction of each agent can be arbitrary and shall not be changed during the execution.
- The algorithm shall be designed for simultaneous synchronous updating (obeying to a central clock).
- The algorithm shall also work with asynchronous updating.

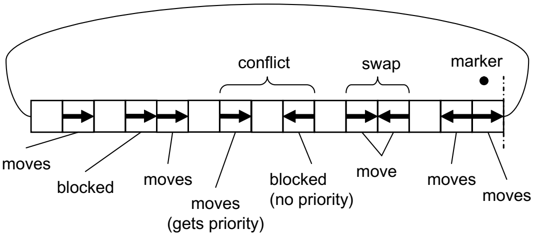


Fig. 13.8 Moving agents on a ring. The agents can notice the length of the ring by a marker (dot). The agents shall sort the numbers on the ring in increasing order (from left to right (dot)).

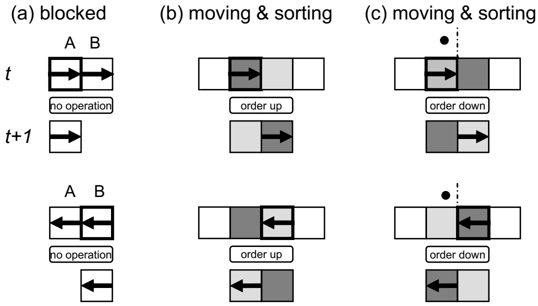


Fig. 13.9 Situations and solutions. (a) Agent A is blocked by B, A performs no action. (b) Agent is moving left or right, thereby ordering the numbers. Dark gray represents a number that is higher then the number represented by light gray. (c) When an agent notices the marker (end of the array to be sorted), the ordering relation is inverted for once.

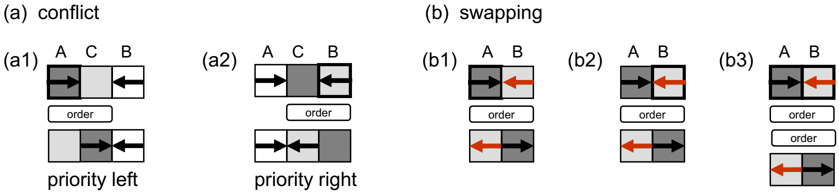


Fig. 13.10 Situations and solutions. (a) Conflict, solved by giving priority to the agent from left (a1). (b) Swapping. Agents exchange their positions, all three solutions yield the same result.

The basic algorithmic idea used here for the solution is *odd–even sort* [2]. This algorithm processes $N/2$ times the two following alternating phases:

- *Phase 1:* all $N/2$ pairs of elements $(d[i], d[i+1])$ for even i are compared and transposed if they are in the wrong order.
- *Phase 2:* all $N/2-1$ pairs of elements $(d[i], d[i+1])$ for odd i are compared and transposed if they are in the wrong order.

Now the idea is that each agent tries to order a pair of numbers when moving around. When only one agent is in the system, then the algorithm is simple. Let us assume that the agent moves rightwards (clockwise) in the ring. The agent compares the number in front with its own number and orders them if necessary (Fig. 13.9 b). The last element (maximum number at the end) in the sequence to be ordered is marked by a dot. This dot specifies the perimeter of the ring, it can be seen as an implicit information about the number N of data elements being used. When the agent detects the particular "dot"-situation (Fig. 13.9 c) when moving rightwards, it inverts the ordering relation for once from *up* to *down*, because the number following the dot is supposed to be smaller (the minimum at the end). In this way an agent circulates around the ring not more than $(N-1)^2$ times (until the sequence is sorted (see the following simulation sequences, first case, config: 0). It is interesting to observe the next simulation (config: 1), where the agent moves counter-clockwise and immediately reacts on the “dot” situation (Fig. 13.9 b). For this case the numbers are sorted already after 15 time steps.

The whole designed algorithm takes into account all the solutions shown in (Fig. 13.9 a, b, c) and (Fig. 13.10 a1, b3). It is assumed that two head-on agents can swap their positions. The swapping case solutions (Fig. 13.10 b1, b2, b3) yield the same result, it does not matter which one of the agents performs the data exchange.

In the case (b1) agent A is performing the ordering, and in the case (b2) agent B. In the case (b3) both agents are ordering, because no data conflict appears (if multiple assignments of the same data are allowed).

It should be noticed that the decision for using (a2) instead of (a1) would lead to a slightly different behavior, but still the global sorting result is guaranteed. The conflict situation (Fig. 13.10 a) could also be solved in other ways. E.g. it would be possible that the agents on A and B swap their positions (at distance 2), thereby ordering two or even three numbers. In addition, it would be possible that the agents turn in certain situations, but the idea was to do without this feature in order to keep the algorithm simple.

Here are simulations with a varying number of agents and different initial directions (< : to the left, > : to the right).

```
        initial config:0            initial config:1            initial config:2            initial config:3
i=      0   1   2   3   4   5       0   1   2   3   4   5       0   1   2   3   4   5       0   1   2   3   4   5

t= 0    5>  4   3   2   1   0       5<  4   3   2   1   0       5>  4>  3>  2>  1>  0       5>  4>  3>  2>  1>  0<
t= 1    4   5>  3   2   1   0       0   4   3   2   1   5<      5>  4>  3>  2>  0   1>      5>  4>  3>  2>  0<  1>
t= 2    4   3   5>  2   1   0       0   4   3   2   1<  5       5>  4>  3>  0   2>  1>      5>  4>  3>  0<  2>  1>
t= 3    4   3   2   5>  1   0       0   4   3   1<  2   5       5>  4>  0   3>  2>  1>      5>  4>  0<  3>  2>  1>
t= 4    4   3   2   1   5>  0       0   4   1<  3   2   5       5>  0   4>  3>  2>  1>      5>  0<  4>  3>  2>  1>
t= 5    4   3   2   1   0   5>      0   1<  4   3   2   5       0   5>  4>  3>  2>  1>      0<  5>  4>  3>  2>  1>
t= 6    4>  3   2   1   0   5       0<  1   4   3   2   5       0>  5>  4>  3>  2>  1       0>  5>  4>  3>  2>  1<
t= 7    3   4>  2   1   0   5       0   1   4   3   2   5<      0>  5>  4>  3>  1   2>      0>  5>  4>  3>  1<  2>
t= 8    3   2   4>  1   0   5       0   1   4   3   2<  5       0>  5>  4>  1   3>  2>      0>  5>  4>  1<  3>  2>
t= 9    3   2   1   4>  0   5       0   1   4   2<  3   5       0>  5>  1   4>  3>  2>      0>  5>  1<  4>  3>  2>
t=10    3   2   1   0   4>  5       0   1   2<  4   3   5       0>  1   5>  4>  3>  2>      0>  1<  5>  4>  3>  2>
t=11    3   2   1   0   4   5>      0   1<  2   4   3   5       0   1>  5>  4>  3>  2>      0<  1>  5>  4>  3>  2>
t=12    3>  2   1   0   4   5       0<  1   2   4   3   5       0>  1>  5>  4>  3>  2       0>  1>  5>  4>  3>  2<
t=13    2   3>  1   0   4   5       0   1   2   4   3   5<      0>  1>  5>  4>  2   3>      0>  1>  5>  4>  2<  3>
t=14    2   1   3>  0   4   5       0   1   2   4   3<  5       0>  1>  5>  2   4>  3>      0>  1>  5>  2<  4>  3>
t=15    2   1   0   3>  4   5       0   1   2   3<  4   5       0>  1>  2   5>  4>  3>      0>  1>  2<  5>  4>  3>
t=16    2   1   0   3   4>  5                                   0>  1   2>  5>  4>  3>      0>  1<  2>  5>  4>  3>
t=17    2   1   0   3   4   5>                                  0   1>  2>  5>  4>  3>      0<  1>  2>  5>  4>  3>
t=18    2>  1   0   3   4   5                                   0>  1>  2>  5>  4>  3       0>  1>  2>  5>  4>  3<
t=19    1   2>  0   3   4   5                                   0>  1>  2>  5>  3   4>      0>  1>  2>  5>  3<  4>
t=20    1   0   2>  3   4   5                                   0>  1>  2>  3   5>  4>      0>  1>  2>  3<  5>  4>
t=21    1   0   2   3>  4   5                                   0>  1>  2   3>  5>  4>      0>  1>  2<  3>  5>  4>
t=22    1   0   2   3   4>  5                                   0>  1   2>  3>  5>  4>      0>  1<  2>  3>  5>  4>
t=23    1   0   2   3   4   5>                                  0   1>  2>  3>  5>  4>      0<  1>  2>  3>  5>  4>
t=24    1>  0   2   3   4   5                                   0>  1>  2>  3>  5>  4       0>  1>  2>  3>  5>  4<
t=25    0   1>  2   3   4   5                                   0>  1>  2>  3>  4   5>      0>  1>  2>  3>  4<  5>
```

```
        initial config:4            initial config:5            initial config:7            initial config:8
i=      0   1   2   3   4   5       0   1   2   3   4   5       0   1   2   3   4   5       0   1   2   3   4   5

t= 0    5>  4<  3>  2<  1>  0<      5>  4>  3>  2   1   0       5>  4   3>  2   1>  0       5>  4   3<  2   1>  0
t= 1    4<  5>  2<  3>  0<  1>      5>  4>  2   3>  1   0       4   5>  2   3>  0   1>      4   5>  3<  2   0   1>
t= 2    1>  2<  5>  0<  3>  4<      5>  2   4>  1   3>  0       1>  2   5>  0   3>  4       1>  3<  5>  2   0   4
t= 3    1<  2>  0<  5>  3<  4>      2   5>  1   4>  0   3>      1   2>  0   5>  3   4>      1<  3>  2   5>  0   4
t= 4    1>  0<  2>  3<  5>  4<      2>  1   5>  0   4>  3       1>  0   2>  3   5>  4       1   2   3>  0   5>  4<
t= 5    0<  1>  2<  3>  4<  5>      1   2>  0   5>  3   4>      0   1>  2   3>  4   5       1   2   0   3>  4<  5>
t= 6                                1>  0   2>  3   5>  4                                   1>  2   0   3<  4>  5
t= 7                                0   1>  2   3>  4   5>                                  1   2>  0<  3   4   5>
t= 8                                                                                        1>  0<  2>  3   4   5
t= 9                                                                                        0<  1>  2   3>  4   5
```

The simulations (config: 2, 3) show that many agents cannot always solve the task faster than one agent only.

(config: 2). For $N = 6$ cells, 5 agents with direction to right are used. As only the leading agent is able to move and to exchange the numbers, this MAS is not faster than the (config: 0)-system. Note that 6 agents, all initially with direction to the right, are unable to solve the task because all of them are stuck.

(config: 3). 5 agents point to the right, and one to the left. In this case the agent pointing to the left is moving to the left by swapping.

(config: 4). 6 agents are used with alternating directions. There are 3 swapping situations, and 3 possible data exchanges. In fact, the global sorting behavior is nearly the same as it is for the odd-even sort. Compared to odd-even sort, only $N/2-1$ steps are needed, because in every step $N/2$ comparisons are performed, because even in the dot-situation two elements (with index 0 and $N-1$) are compared, though in inverse order.

(config: 7). $N/2$ agents are used with the same direction and one empty cell between two successive ones. The global sorting behavior is the same as in (config: 4).

Still a question remains, namely how many agents at which initial positions and with which initial directions are most effective in general? The precise answer requires a deep formal analysis or an exhaustive evaluation, which cannot provided here. Nevertherless we can draw some conclusions from the different simulation cases. We can learn from (config: 5, 8), cases with $N/2$ agents, that an initially unskilled placement may lead to a longer execution time. One the other hand, a smart initial configuration may speed up the whole process. For example, if in the case (config: 7) all the agent's directions are changed from right to left, only 4 time steps ($N/2-1$ in general) are neded instead of 5.

It can easily be seen, that the a-algorithm works for any initial configuration, except for the one where the ring is fully packed with agents having the same direction.

In the case of asynchronous updating, the same a-algorithm (originally designed for synchrounous updating) can also successfully perform the sorting task. The reason is, that the algorithm then works sequentially at randomly changing positions where only one agent is active at the same time. Because of the sequential updating, no conflict can appear. Therefore an agent may always perform the ordering. But when using the designed synchronous a-algorithm, (i) the blocked agent in (Fig. 13.9 a) does not perform ordering, and (ii) if the left agent has priority in the conflict situation (Fig. 13.10 a1) and the right agent is active (selected by asynchronous updating) then no movement and data ordering takes place (although possible). But these unsatisfying cases could be improved for asynchronous updating in a simple way, namely in a way where an active agent compares and orders always, and moves always if not blocked.

Table 13.1 In the swapping situation, agents exchange their positions, and the states of the agents are updated as defined by the table

case	(t)	(t+1)
	→←	←→
	A B	B A
1	0 #	# 0
2	0 0	# 0
3	0 1	1 #
4	0 2	2 #
5	1 #	# 1
6	1 0	# 1
7	1 1	# 1
8	1 2	# 1
9	2 #	# 2
10	2 0	# 2
11	2 1	# 2
12	2 2	# 2
13	# 0	0 #
14	# 1	1 #
15	# 2	2 #
16	# 3	3 #
17	3 #	# 3

13.2.5 Leader Election

Initially p processes are in the same initial state $(state = 0)$. The goal is to find in a decentralized way a terminal configuration in which only one process is in the state *leader* $(state = 3)$ and the others are in the state *lost* $(state = \#)$.

Originally this problem was posed by Lelann [11], several solutions are discussed in [13]. Here the problem shall be solved by agents which are moving around in a ring. At the beginning there are p agents in state 0, and at the end, only one agent shall be in state *leader*, and the others in state *lost*. We assume that agents in *lost* state are still active and are always moving around. It would also be possible to deactivate or delete such agents, then the algorithm, designed in the following, could easily be modified.

Concerning the moving situations and reactions, the algorithm is similar to the sorting a-algorithm presented before. The basic idea is to define one agent as *lost* when two agents meet head-on (the swapping situation before). In order to force the agents to meet, agents that perceive the marker on the ring, turn their direction.

There are four states an agent can take on:
$state \in \{0,1,2,3,\#\} = \{init,\ leftborder,\ rightborder,\ leader,\ lost\}$.

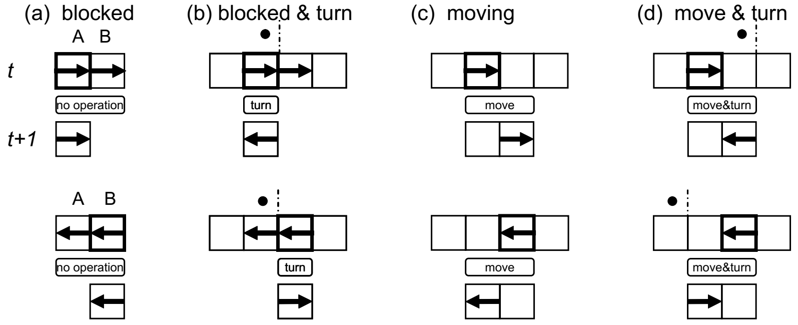


Fig. 13.11 Situations and solutions. (a) Agent A is blocked by B, it performs no action. (b) Agent is blocked at the marker and turns back. (c) Agent moves. (d) Agent moves and turns because it detects the marker.

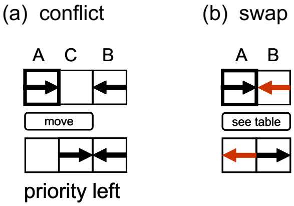


Fig. 13.12 Situations and solutions. (a) Conflict, solved by giving priority to the agent from left. (b) Swap, agents states' are updated according to Table. 13.1.

If an agent detects the left/right border (situations Fig. 13.11 b, d), it changes into state 1/2. If an agent is already in state 1/2 (visited one of the borders) and detects the other border (right/left border), it changes into state 3 (leader). When two agents meet head-on they swap (Fig. 13.12 b), and Table 13.1 defines the new states of the involved agents. For some cases the definitions are not mandatory, e.g. $(1,2) \rightarrow (\#,1)$ could be replaced by $(1,2) \rightarrow (2,\#)$. It can be presumed that another more simple solution can be derived from this one, by joining the two states 2, 3 into one (representing that the agent has met any of the borders already. The presented solution was guided by the idea that agents may change their direction in between the borders, e.g. by random noise.

The following simulations show how the leader-election a-algorithm works.

```
         initial config:0              initial config:4              initial config:10             initial config:12
i=       0   1   2   3   4   5         0   1   2   3   4   5         0   1   2   3   4   5         0   1   2   3   4   5

t= 0     0>                            0>  0>  0>                    0>  0<  0>          0<        0>  0>  0>  0>  0>
t= 1         0>                        0>  0>      0>                #>  0>      0>  0<            0>  0>  0>  0>      2<
t= 2             0>                    0>      0>      0>            #>      0>  #<  0>            0>  0>  0>      0>  2<
t= 3                 0>                    0>      0>      2<            #>  #<  0>      2<        0>  0>      0>  2<  #<
t= 4                     0>                    0>      0>  2<            #<  #>      0>  2<        0>      0>  2<  #>  #<
t= 5                         2<                    0>  2<  #<        #>          #>  2<  #<            0>  2<  #>  #<  #<
t= 6                     2<                        2<  #>  #<            #>      2<  #>  #<            2<  #>  #<  #>  #<
t= 7                 2<                        2<      #<  #<                #>  2<  #<  #<        3>      #<  #>  #<  #<
t= 8             2<                        2<      #<      #<                2<  #>  #<  #<
t= 9         2<                        3>      #<      #<                2<      #<  #>  #<
t=10     3>                                                          3>      #<      #<  #<

         initial config:13             initial config:16             initial config:17             initial config:19

t= 0     0>  0>  0>  0>  0>  0<        0<  0<  0>                        0<  0<  0>  0>            0>  0>  0>  0>  0>  0>
t= 1     0>  0>  0>  0>  #<  2<            0<      0>      0<        1>      0<  0>      2<        0>  0>  0>  0>  0>  2<
t= 2     0>  0>  0>  #<  0>  2<        1>              0>  0<            1>  0<      0>  2<        0>  0>  0>  0>  2<  #<
t= 3     0>  0>  #<  0>  2<  #<            1>          #<  2<            #<  1>      2<  #<        0>  0>  0>  2<  #>  #<
t= 4     0>  #<  0>  2<  #>  #<                1>  #<      2<        #>          1>  2<  #<        0>  0>  2<  #>  #<  #<
t= 5     #>  0>  2<  #>  #<  #<                #<  1>  2<                #>      #<  1>  #<        0>  2<  #>  #<  #>  #<
t= 6     #>  2<  #>  #<  #>  #<            #<      #<  1>                    #>  #<  #<  3<        3>  #>  #<  #>  #<  #<
t= 7     3>  #>  #<  #>  #<  #<        #>      #<          3<
```

(config: 0). One agent is starting from the left border and keeps it initial state until it reaches the right border. There the agent changes into state 2 and returns to the left border where it changes into the leader state 3.

(config: 4). Three agents are moving to the right. The first agent returns with state 2 and switches its followers into the lost state #. When reaching the left border, this agent becomes the leader.

(config: 10). Four agents move in different directions. When agents meet head-on, one more agent changes into the lost state. At $t = 2$ the rightmost agent perceives the border and afterward moves left with state 2 until reaching the right border.

(config: 12). The rightmost agent changes into state 2 and turns. When this agent meets the other ones, they change into the lost state.

(config: 13). The system is completely filled with agents. Agent at $i = 4$ moves to $i = 5$, turns and changes into state 2. Agent at $i = 5$ moves to $i = 4$ and changes into state #.

(config: 19). The system is completely filled with agents. Agent at $i = 5$ is blocked, therefore it turns and changes into state 2.

(config: 16). Agent at $i = 1$ and $t = 1$ moves to $i = 0$, turns and changes into state 1. At the end, this agent becomes the leader, winning against the other candidate in state 2 coming from the right.

(config: 17). The initial configuration is symmetric (agents at $i = (1,2)/(3,4)$ point to the left/right). Then, in the conflict and swapping situations, the agent from the left gets priority and becomes the leader at last.

13.3 Summary

The CA-w (Cellular Automata with write access) model allows to modify not only the cell's own state but also the neighbor's state. The neighbors are accessed via dynamic links that can be modified by the cell's rule. The classical CA model does not allow to modify the state of a neighbor, therefore no write-conflict can occur. However, for the description of applications with a varying number of active cells and moving agents the CA-w model is a better choice because information can actively be transferred to a neighbor, and the activity of the neighbor can be switched on or off. Compared to CA, the CA-w descriptions for such problems are more concise and "natural" because the change of the system's state is only controlled by the active cells. Furthermore, the computational effort to simulate CA-w is minimized because only active cells have to be computed, thereby minimizing the energy consumption, too. It was shown that basic computational problems with dynamic activities (traffic rule, computing the binomial coefficients and Fibonacci numbers, sorting and leader election with moving agents) can easily be modeled by CA-w.

References

1. Achasova, S., Bandman, O., Markova, V., Piskunov, S.: Parallel Substitution Algorithms, Theory and Applications. World Scientific (1994)
2. Haberman, N.: Parallel Neighbor Sort (or the Glory of the Induction Principle) CMU Computer Science Report (available as Technical report AD-759 248, National Technical Information Service, US Department of Commerce, 5285 Port Royal Rd Sprig field VA 22151) (1972)
3. Hoffmann, R., Völkmann, K.-P., Waldschmidt, S.: Global Cellular Automata GCA: An Universal Extension of the CA Model. In: Worsch, T. (ed.) ACRI Conf. Work in Progress Session (2000)
4. Hoffmann, R., Völkmann, K.-P., Waldschmidt, S., Heenes, W.: GCA: Global Cellular Automata. A Flexible Parallel Model. In: Malyshkin, V.E. (ed.) PaCT 2001. LNCS, vol. 2127, pp. 66–73. Springer, Heidelberg (2001)

5. Hoffmann, R.: GCA-w Algorithms for Traffic Simulation. Acta Physica Polonia. In: Proc. Suppl. Summer Solistice Int. Conf. on Discr. Models of Complex Systems, Nancy France. Polish Academy of Science, Cracow (2011)
6. Hoffmann, R.: The GCA-w Massively Parallel Model. In: Malyshkin, V. (ed.) PaCT 2009. LNCS, vol. 5698, pp. 194–206. Springer, Heidelberg (2009)
7. Hoffmann, R.: GCA-w: Global Cellular Automata with Write-Access. In: Acta Physica Polonia Proc. Suppl. Summer Solistice Int. Conf. on Discr. Models of Complex Systems Gdansk 2009. Polish Academy of Science, Cracow (2010)
8. Hoffmann, R.: Rotor-routing Algorithms Described by CA-w. In: Acta Physica Polonia Proc. Suppl. Summer Solistice Int. Conf. on Discr. Models of Complex Systems Turku, Finland 2011, Turku, Finland. Polish Academy of Science, Cracow (2012)
9. JaJa, J.: An Introduction to Parallel Algorithms. Addison-Wesley (1992)
10. Keller, J., Kessler, C., Träff, J.: Practical PRAM Programming. Wiley (2001)
11. LeLann, G.: Distributed Systems: towards a formal approach. In: Gilchrist, B. (ed.) Proc. Information Processing, pp. 155–160. North Holland (1977)
12. Schweitzer, F.: Brownian Agents and Active Particles, Collective Dynamics in the Natural and Social Sciences. Springer Series in Synergetics. Springer (2003)
13. Tel, G.: Introduction to Distributed Algorithms. Cambridge University Press (1994)

Chapter 14
Broadcasting Automata and Patterns on $\mathbb{Z}^2$

Thomas Nickson and Igor Potapov

Abstract. The recently introduced Broadcasting Automata model draws inspiration from a variety of sources such as Ad-Hoc radio networks, cellular automata, neighbourhood sequences and nature, employing many of the same pattern forming methods that can be seen in the superposition of waves and resonance. Algorithms for the broadcasting automata model are in the same vain as those encountered in distributed algorithms using a simple notion of waves, messages passed from automata to automata throughout the topology, to construct computations. The waves generated by activating processes in a digital environment can be used for designing a variety of wave algorithms. In this chapter we aim to study the geometrical shapes of informational waves on integer grid generated in broadcasting automata model as well as their potential use for metric approximation in a discrete space. An exploration of the ability to vary the broadcasting radius of each node leads to results of categorisations of digital discs, their form, composition, encodings and generation. Results pertaining to the nodal patterns generated by arbitrary transmission radii on the plane are explored with a connection to broadcasting sequences and approximation of discrete metrics of which results are given for the approximation of astroids, a previously unachievable concave metric, through a novel application of the aggregation of waves via a number of explored functions.

Thomas Nickson
The University of Edinburgh, Kennedy Tower,
Royal Edinburgh Hospital, Edinburgh EH10 5HF, UK
e-mail: tnickson@exseed.ed.ac.uk

Igor Potapov
University of Liverpool, Ashton Street, Ashton Building, Liverpool, L69 3BX, UK
e-mail: potapov@liverpool.ac.uk

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_14

14.1 Introduction

The recently introduced model of *Broadcasting Automata* [19] connects many of the techniques contained in distributed algorithms, ad-hoc radio networks, cellular automata and neighbourhood sequences. Much like cellular automata with variably defined neighbourhoods Broadcasting Automata can be defined on some form of grid or lattice structure and have a simple computational primitives comparative to a finite state automata with the ability to receive and send messages both from and to those automata which are within its transmission radius. Neighbourhoods are defined in the same way as with an ad-hoc network, all those points within a certain transmission radius, receive the message from the sender.

Algorithms for the broadcasting automata model are in the same vein as those encountered in distributed algorithms using a simple notion of waves, messages passed from automata to automata throughout the topology, to construct computations [20]. Wave algorithms are enhanced further with notions of composition of the information that is carried within each wave borrowed from the physical world and embellished with the new found computational power of the automata.

The waves generated by activating processes in a digital environment can be used for designing a variety of wave algorithms. In fact, even very simple finite functions for the transformation and analysis of passing information provides more complex dynamics than classical wave effects. In [19, 20] we generalized the notion of the standing wave which is a powerful tool for partitioning a cluster of robots on a non-oriented grid. In contrast to classical waves where interference patterns generate nodal lines (i.e. lines formed by points with constant values), an automata network can have more complex patterns which are generated by periodic sequences of states in time.

Here we take a different direction and aim to study the geometrical shapes of informational waves on the integer grid generated in broadcasting automata model as well as their potential use for metric approximation in a discrete space. The repeated transmission to nodes, within certain radii, applied to a certain network topology, or physical layout of nodes has been studied under the name, *Neighbourhood Sequences* (NS). The concept of neighbourhood sequences is of importance in a number of practical applications and was originally applied for measuring distances in a digital world [13]. Initially two classical digital motions (cityblock and chessboard)[1] were introduced. Based on these two types of motions periodic neighbourhood sequences were defined in [3] by allowing arbitrary mixture of cityblock and chessboard motions. In 2D the distances based on cityblock and chessboard neighbourhood sequences deviate quite substantially from the ideal Euclidian distances, so instead their combination that form "the octagon" was more often employed and studied. Later the concept of neighbourhood sequences was extended to arbitrary finite and infinite dimensions, periodic and non-periodic sequences and then analysed in terms of their geometric properties.

[1] The cityblock motion allows movements only in horizontal and vertical directions, while the chessboard allows to move in diagonal directions.

The aggregation of two classical neighbourhood sequences based on Moore and Von Neumann neighbourhoods (which correspond to cityblock and chessboard) was recently proposed as an alternative method for self-organization, partitioning and pattern formation on the non-oriented grid environment in [19]. In particular the discrete analogs of physical standing wave phenomena were proposed to generate nodal patterns in the discrete environment by two neighbourhood sequences. The power of the primitives was illustrated by giving distributed algorithms for the problem of finding the centre of a digital disk of broadcasting automata. The shapes that can be formed by neighbourhood sequences in dimension two are quite limited. However the basic notion of neighbourhood sequences can be naturally extended by relaxing the constraints on the initial definition of the neighbourhood in such a way that two points are neighbors (r-neighbours) if the Euclidean distance is less than or equal to some r, used to denote the radius of a circle. Then by *Broadcasting sequences* we understand the periodic application of the r-neighbours distance function.

The main result of this work is characterization of geometrical shapes that can be generated by Broadcasting Sequences on the square lattice. The shapes of r- neighborhoods correspond to Discrete Discs which are discrete convex polygons. First we use the language of Chain Codes (i.e. the code based on 8 degrees of motion) to describe the shape of the Discrete Discs and their Broadcasting sequences. In particular we introduce the notion of Chain code segments and Line segments to express the shapes of the polygons corresponding to Discrete Discs. Then we characterize the shapes of polygons produced by Broadcasting sequences and provide a linear time algorithm for the composition of two chain codes of Discrete Discs. Based on their composition properties we derive a number of limitations for produced polygons. For example we show that there exist an infinite number of gradients (of line segments in the polygons) that cannot be produced by Broadcasting Sequences. It also becomes clear that the set of line segments, and as such gradients, that compose any discrete circle are closed under composition.

Moreover, we provide an alternative method for enriching the set of geometrical shapes and neighbourhood sequences by aggregation of two Broadcasting Sequences. Initially we illustrate the idea on Moire and Anti-Moire aggregation function to produce an infinite family of polygons and polygonal shapes and characterize the gradients of their line segments. We have noticed that Anti-Moire aggregation function can be slightly modified to provide better approximation for the Euclidean distance on a square lattice then classical neighbourhood sequences. Finally it is possible to observe the variety of effects that are the result of the application of an aggregation function which are themselves shapes of some form.

14.2 Broadcasting Automata Model

One of the fundamental models of computation is the automaton. Finite state automata take as input a word, which may in some instances be referred to as a tape, and output is limited to an accepting state which the automata is left in if it is said to accept the word. Traditionally automata are used in cases where it is important to transform some input in to an output, beyond the use of a single state, is the use of Moore Machines. Such machines differ from finite state machines in that they are able to produce an output word from an input word.

Definition 1. A Moore machine [4] is a 6-tuple, $A = (Q, \Sigma, \Lambda, \delta, \Delta, q_0)$, where:

- Q is a finite set of states,
- Σ is the set of input symbols,
- Λ the set of output symbols,
- $\delta : Q \times \Sigma \to Q$ is the transition function mapping a state $q \in Q$ and a symbol, or set of symbols, $\sigma \in \Sigma$ to a state $q \in Q$,
- $\Delta : Q \to \Lambda$ is the output function which maps a state, $q \in Q$, to an output symbol, $\lambda \in \Lambda$, and
- q_0 is the initial or quiescent state in which the automata starts.

It is assumed that such a machine is connected to an input tape, or word, and an output tape, or word, where the result of the computation is read. In a situation, as is presented in distributed systems, whereby automata are connected to each other it is possible for the output of one automata to become the input of another automata. Such a model is known as a network of automata and connections from one automatons output to another's input may be represented as a directed graph, where direction represents the output going to input from automaton to automaton.

Definition 2. A network of finite automata is a triple, (G, A, C_0) , where:

- $G = (V, E)$ is a directed graph, with vertices, V, and edges, E, which are ordered pairs of vertices,
- A, is a Moore machine, and
- C_0 is an initial configuration which maps states, Q, of the automata, A, to vertices, V, such that, $C_0 : V \to Q$.

In this model the topology is fixed as specified by the construction of the graph, G, which dictates the flow of inputs and outputs from the Moore machines. Such symbols, where a symbol is part of the input/output word of the Moore machine, are generated as response to some input, by an automaton at vertex, $v \in V$, and then sent to all of the adjacent vertices in the graph or to a particular adjacent vertex, G, where the automata that receive the symbols process them as they would their input word.

A less abstract model is considered by adding more details about the communication between automata, where they are located in Euclidean space and what can be transmitted. It will also be required to define more refined models that will reflect the new features and constraints of the physical environment, but are still at a high level of abstraction.

Taking the inspiration from ad hoc wireless communication networks we introduced in [19] a new model of *Broadcasting Automata*, which can be seen as a network of finite automata with a dynamic network topology. Informally speaking, the model of Broadcasting Automata comprises nodes, which correspond to points in space, and connectivity between nodes depends upon the distances between the points and the strengths of the transmissions generated by the automata as may be seen in ad-hoc radio networks. Transmission strength may vary from round to round for any particular automaton in the space and is dictated by the state of the automaton. In this model the topology, or connectivity graph, of the network of automata is able to change at each time step based on the states of the automata.

In order to give a formal definition of the Broadcasting Automata model on a metric space it is first necessary to introduce the notion of a metric space and to modify the classical notion of the Moore machine.

Definition 3. A metric space is an ordered pair, (M,d), where M is a set and d is a metric on M such that $d : M \times M \to \mathbb{R}$.

Here, (M,d), is any metric space, later the two dimensional euclidean space shall be used, but this is not a necessity simply that a notion of the distance between two points is required.

Definition 4. The *Broadcasting Automaton* extends the Moore Machine by introducing a set of final states, F, along with a function, $\tau : Q \to \mathbb{R}$, which maps the state to a real number and represents the radius of transmission for the output symbol and the output alphabet, Λ, is extended by adding an empty symbol, ε and it is represented by an 8-tuple $A = (Q,\Sigma,\Lambda,\delta,\Delta,\tau,q_0,F)$.

A network of *Broadcasting Automata*, which will also be referred to as the *Broadcasting Automata model*, can now be defined.

Definition 5. The *Broadcasting Automata model* is represented by a triple $BA = ((M,d),A,C_0)$ where:

- (M,d) is a metric space,
- A is a Broadcasting Automata, $A = (Q,\Sigma,\Lambda,\delta,\Delta,\tau,q_0,F)$,
- C_0 is the initial configuration of the Broadcasting Automata model, it is a mapping from points, M, to states of the Broadcasting Automata, Q, such that, $C_0 : M \to Q$.

In some cases it may be that the input and output symbols are drawn from the same alphabet in which case, $\Sigma = \Lambda$.

The communication between automata is organised by message passing, where *messages* are symbols from the output alphabet, Λ, of the automata, A, to all of the automata within its transmission radius. Messages, symbols from the output alphabet, Λ, of the automata, A, are generated and passed instantaneously at discrete time steps, generation of message is given by the function, Δ, for the automata, A, resulting in synchronous steps. Those automata that have received a message, for the first time in the computation, are said to be *activated*.

If several messages are transmitted to an automaton, A, it will receive only a set of **unique messages**, i.e. for any multiset of transmitting messages, where the multiset represents a number of the same message being sent, received by A, over some number of rounds, the information about quantity of each type, within a single round, will be lost. This simply illustrates that the automaton may not dictate a number of the same symbol in any transition, all transitions must operate upon a set of distinct symbols.

More formally the concept of computation and communication is captured in the concept of configurations of the Broadcasting Automata model and its semantics presented here.

Definition 6. The *configuration* of the Broadcasting Automata is given by the mapping, $c : M \to Q$, from points in the metric space to states. It can be noted here that C_0 is an initial configuration for the Broadcasting Automata model.

Automata are updated, from configuration to configuration, synchronously at discrete time steps. The next state of each automaton depends upon the states of all other automata which have in their neighbourhood the automaton that is to be updated. Where here the neighbourhoods are formed by a combination of their position in M and the range of their transmission, dictated by $\tau(q)$ for all automata.

Definition 7. The set of messages received by the automaton at a point, $u, v \in M$, is expressed by the set $\Gamma_u = \{\Delta(c(v)) | v \in M \wedge d(u,v) \leq \tau(c(v))\}$ for the metric space, (M,d).

In Figure 14.1, it is possible to see an elucidation of message passing in the Broadcasting Automata model through the process of constructing a digraph representation of the broadcasting radii. The figure depicts the automatons broadcasting radius by way of a dashed circle with the automaton broadcasting at that range shown in the centre. The corresponding graph may be constructed from this by making a directed edge in the graph from the node that is broadcasting to all nodes that are within the broadcast range, where the range is dictated by the state, $q \in Q$ and the function, $\tau(q) = r$, where r is the broadcasting radius. The construction of such a graph shows both the connection to the network automata model, which operates on a similar premise with a fixed graph, but also highlights how Definition 7 is constructed. Incoming edges are generated by connecting nodes, with the arrow from transmitter to node, reachable from the transmitter. Naturally such a graph, in the Broadcasting Automata model, can change over time which is shown by images further down the page as increase time steps in Figure 14.1.

It should be noted that this set of messages can be empty. The new state of the automaton at u is now given by applying the automaton's transition function $\delta_u(\Gamma_u)$. This may be applied to all of the automata in the BA and as such determines the global dynamics of the system. It can be seen that in one time step it is possible for configuration c to become configuration e where, for all $v \in M$, $e(v) = \delta_v(\Gamma_u)$. Such a function is called the global transition function. The set of all configurations for BA is denoted as $\mathcal{C}$.

Definition 8. The global transition function, $\mathcal{G} : \mathcal{C} \to \mathcal{C}$, represents the transition of the system at discrete time steps such that for two configurations, $c \in \mathcal{C}$ and $e \in \mathcal{C}$, where e is the configuration the time step directly after c such that, $e = \mathcal{G}(c)$.

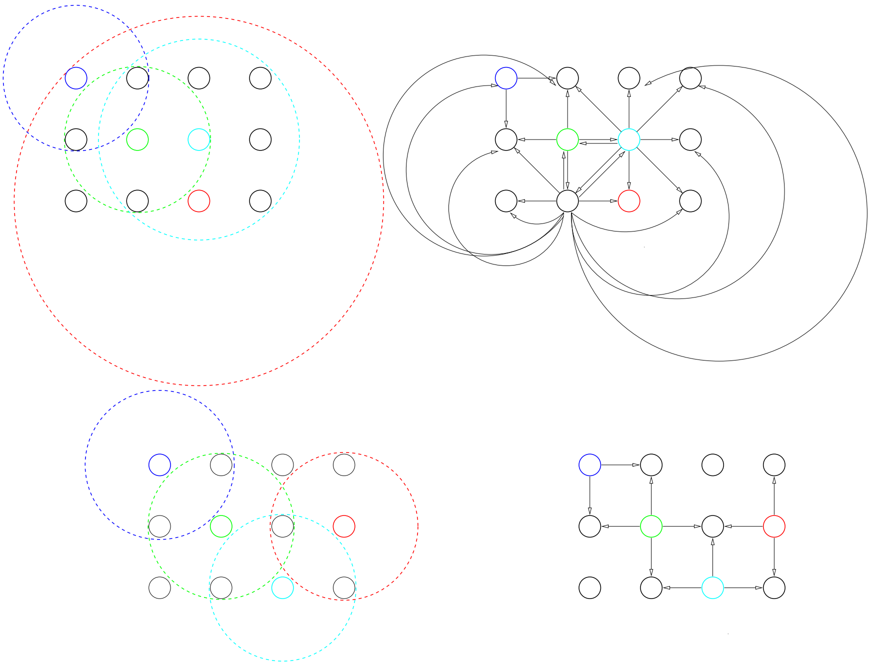


Fig. 14.1 The above figures show the possible evolution over time of a network of Broadcasting Automata. Each figure shows, on the left, the broadcast range for the automata, depicted by a dotted circle with the node at the centre, on the right, the same connectivity is shown in graph form.

Definition 9. A computation is defined as any series of applications of the function, $\mathcal{G}$, from an initial condition, C_0, that leads to the set of all automata in the system being in one of their accepting states from the set, F, or the computation halts such that there exists no defined relation from the current configuration, c, under the global transition function $G(c)$.

Here a notion of reachability is given.

Definition 10. One configuration, c, is said to be **reachable** from another, e, if from the initial configuration, C_0, there exists a series of valid intermediary configurations such that $c \rightarrow c' \rightarrow c''... \rightarrow e$. Where $c \rightarrow c'$ is equivalent to $\mathcal{G}(c) = c'$. It can also be stated, in a shorthand way, that where the sequence $c \rightarrow c' \rightarrow c''... \rightarrow e$ exists reachability from c to e may be shown as $c \rightsquigarrow e$.

In the above context the *time complexity* of an algorithm in the Broadcasting Automata model is determined by the number of applications of the global transition function during the computation.

Whilst in general Broadcasting Automata may be used represent any of the common network topologies (such as, linear array, ring, star, tree, near-neighbour mesh, systolic array, completely connected, chordal ring, 3-cubes and hyper-cubes [5, 8]), through varying the location and transmission radii of the automata in the space, a different model is employed. The model used throughout the paper is a hybrid, mixing the notion of transmission radius in the pathloss geometric random graph model and common network topologies. The method involves the physical placement of nodes on the Euclidean plane in a lattice structure such that the distances between points conform to the euclidean distance. Such regular lattices coincide with those structures that can be formed by previous works [16, 17, 27] giving the basis of the forms of lattice that can be studied. This paper will strictly concern itself with the **square grid lattice** though it can be extended in to any of the other grid configurations such as the hexagonal or triangular lattices along with many others.

The locations of the points on the lattice allow the construction of the metric space, (M,d) as such the locations of the automata. Assuming a lattice structure for positioning of the automata and transmission radii, $\mathcal{R}$, is in accordance with the principles of transmission used in the pathloss geometric random graph model put forth in Ad-Hoc networks.

A variety of topologies, the lattices that can be constructed as in [16] (square, hexagon and triangle), and the shapes of neighbourhoods generated by varying the transmission neighbourhood can be seen in Figure 14.2 where black dots represent the placement of automata on the plane according to the lattice here shown by the black lines. Neighbours are those within the dashed circle and the automata at the centre of the circle is the initial transmitter of any messages.

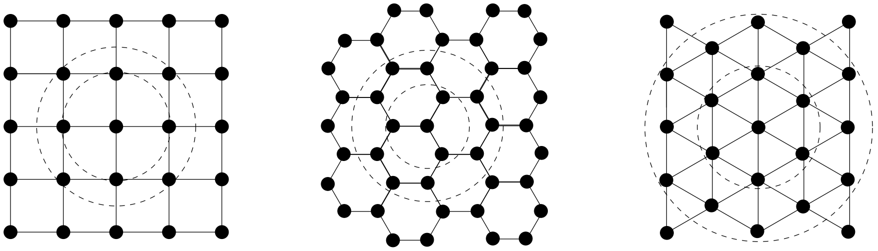


Fig. 14.2 Showing three differing lattices and how their neighbourhoods can be altered simply by varying the radii of the transmission neighbourhood which, here, is depicted as a dotted circle. All automata, shown as black dots, are able to receive messages from the automata at the centre of the circle. The three lattices depicted are (left to right) square, hexagon and triangle.

The connectivity graph defined in ad-hoc radio networks denotes those that are able to send and receive messages with certainty however in distributed algorithms the possibility of communication errors is not ignored. In such models messages may be lost, duplicated, reordered or garbled where they must be detected and corrected by supplementary mechanisms which are mostly referred to as protocols. For example the common network protocol of TCP. The Broadcasting Automata model sides with the former method of message passing. It is assumed that there will be no error in transmission, the receipt or sending of messages across the network is guaranteed, however, consideration is made to the synchronicity of the network due to the high sensitivity to the timing of message passing as will become clear later when discussing pattern formation with such protocols.

To this extent we consider two variants of Broadcasting Automata: **synchronous** and **asynchronous** (or reactive) models. In each of the variations both the transmission of a method and amount of time take for the automata to process the message is considered to be constant. It is this constant time that will afford synchronisation within the model. The two differing models are presented to show that there are possible complexity trade offs depending on how it is to be measured. If the size of the alphabet is a priority, such that it must be minimised, then the synchronous model is best, however this comes at a penalty of time complexity, where the asynchronous model fairs better.

In the **asynchronous** model upon the receipt of a message, an input symbol (or set of symbols) $\sigma \in \Sigma$, the automaton, A, becomes active. The automaton may only react to a non-empty set of messages, it may not make and epsilon transition. Once activated the automaton, A, reacts to the symbol(s), σ, according to the configuration of the automaton and responds with the transmission of an output symbol, $\lambda \in \Lambda$, to those in its transmission radius for the current state, $q \in Q$, $\tau(q)$. The processing of the input symbols by the automaton is done in one discrete time step which is the same period for all automata in the graph. Once active the automaton must wait for another message in order to change its state it is not allowed epsilon transitions.

The **synchronous** model has a singular alphabet and as such, $\Sigma = \Lambda$ with the allowance of epsilon transitions. Upon the receipt of a message, an input symbol (or set of symbols) $\sigma \in \Sigma$, the automaton, A, becomes active. Once active the automaton is synchronised with the rest of the graph by repeated epsilon transitions that are representative of a constant transition via input symbol as in the asynchronous model. As each such transition is considered a change in configuration of the automaton it must take a single time step. As such this guarantees the synchronisation of system. However the multiple alphabet can be simulated by associating different symbols to different time steps as shall be seen later.

The following outlines, informally, the methodology of message passing used by the two differing constructions of automata, synchronous and asynchronous.
In the **synchronous** model messages are passed from automaton to automaton according to the following rules:

(1) An automaton, A, receives a message from an activating source at time, t;
(2) At time $t+1$, A sends a message to all automata within its transmission radius dependent on its state, $q \in Q$, $\tau(q)$;
(3) At time $t+2$, A ignores all incoming messages for this round.

In the **asynchronous model** the following rules can be applied:

(1) An automaton, A, receives a message $\sigma_i \in \Sigma$ from an activating source at time t;
(2) The automaton, A, broadcasts a message $\sigma_{(i+1) \bmod |\Sigma|}$ to all automata within its transmission radius, dependent upon its state, $q \in Q$, $\tau(q)$ at time $t+1$;
(3) Ignore all incoming messages at time $t+2$.

In both models Step 3 prevents an automaton from receiving back the message that has just been passed to the automatons neighbours by ignoring all transmissions received the round after transmission and ensuring that the messages are always carried away from the initial source of transmission. In both cases this rejection of all messages may be modelled by the addition of a state that simply does not accept input in that state, where epsilon transitions apply, or that the transition is made to another equivalent state independent of input received, where from here it is possible to receive transmissions that once again affect the logic of the program.

Naturally, given these two models it is interesting to see if they are capable of the same computations and that indeed anything that can be done in one model can be done in the other. There are many ways with which to establish equivalence between automata and models in general. One important technique that has been used to show equivalence in connection with Turing machines is simulation [18]. Here a similar proposition is made with respects to establishing the equivalence of the two models of automata that are given here, synchronous and asynchronous.

Proposition 1. *[20] Both the synchronous and asynchronous models are able to simulate the other.*

It should also be noted that with some generalisation, to allow a variable radius neighbourhood interaction which has been explored in a limited sense [11, 29], the Cellular Automata (CA) model may also be used to simulate the Broadcasting Automata model. In a grid of CA it is possible to assign states that correlate to transmitters where each transmitter in the grid is assigned a unique state and all other CA are in the quiescent state. The initial transmitters state causes each of the automata who have the initial transmitter in their neighbourhood to change from the quiescent state to the transmitters state plus one over some modulo. This clearly simulates the transmission of some word over a modulo for all automata in the grid. Upon finding a CA with one of this initial set of states within its neighbourhood, and such that it is all encompassing in its neighbourhood, the second transmitter changes its state which triggers a second cascade of state changes that are equivalent to those from the first transmitter only this time the automata must take in to account the initial state that it is already in. This should provide an exact copy of the Broadcasting Automata model, assuming that CA is extended to allow such variable neighbourhoods. This does not mean however that there is an exact translation of Cellular Automata in to Broadcasting Automata and as such it is only possible to say that $BA \subseteq CA$.

14.3 Variable Radius Broadcasting over $\mathbb{Z}^2$

Whilst the model provided here covers many configurations for the underlying communication graph this paper is mainly considering the following case, where all automata are place in euclidean space according to a regular arrangement equivalent to that of a square lattice such that, using the Cartesian coordinate system, for any automata in the space its nearest neighbour is at distance 1 and its next nearest neighbour is at distance $\sqrt{2}$. This idea is illustrated for two dimensions in

Figure 14.3. This now leads to the model of Broadcasting Automata where (M,d) is $M = \mathbb{Z} \times \mathbb{Z}$ and where for $\rho = (\rho_0, rho_1)$ and $\rho' = (\rho'_0, \rho'_1)$ the distance function is $d(\rho,\rho') = \sqrt{(\rho_0 - \rho'_0)^2 + (\rho_0 - \rho'_0)^2}$, the Euclidean distance function for $\rho, \rho' \in M$.

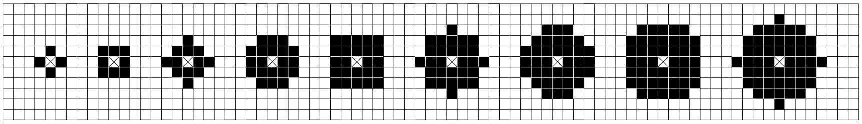


Fig. 14.3 A variety of transmission radii are shown (l-r) squared radii $r^2 = \{1,2,4,5,8,9,10,13,16\}$. Crosses represent the centre of the respective discrete disc.

Figure 14.4 shows as example the automata that receive a message when the Euclidean space and square lattice are restricted to two dimensions but the radius of broadcast, $\tau(q)$ for $q \in Q$, is varied. The automaton at point $\rho \in M$ which is the source of the transmission is shown as the circle at the centre of the surrounding automata on the plane, those that are black are within the transmission range $\tau(q)$ for $q \in Q$. Successive larger collections of automata coloured black show what happens when range can be changed to alter the automata that are included in the transmission radius, $\tau(q) \in \mathcal{R}$. If the transmission radius, $\mathcal{R}$, is equal to 1, as in Figure 14.4 diagram $a)$ then only four of the eight automata can be reached. If the radius is made slightly larger and is equal to $\sqrt{2}$, it can encompass all eight automata in its neighbourhood as shown in diagram $b)$. Such structures are identical to the well studied neighbourhoods von Neumann and Moore (i.e. chess board and city block) respectively and as such here it shall be considered that such constructions of neighbourhoods are a generalisation of these two neighbourhoods. As we will show later, iterative broadcasting within von Neumann and Moore neighbourhoods can distribute messages in the form of a diamond wave and a square wave as shown later in Figure 14.5.

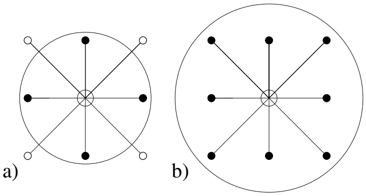


Fig. 14.4 Diagram $a)$ represents the propagation pattern for a diamond wave (Von Neumann neighbourhood) and diagram $b)$ shows the propagation pattern for a square wave (Moore neighbourhood).

The construction of distinct radii for the circles is defined by the numbers n such that $n = x^2 + y^2$ has a solution in non-negative integers x, y [26]. Here it can be seen that r^2 is given for convenience as n always has a convenient representation in $\mathbb{Z}$ whereas it becomes cumbersome to write out either the root of the integer or its decimal representation. For an explanation as to how these numbers relate to the distinct discrete discs it must be noted that distinct discs are constructed such that r^2 contains a new solution in x and y. As it can be considered that the digital disc represents all of the maximal combinations of Pythagorean triples such that $x^2+y^2 \leq r^2$ then for one disc to differ from another there must be an increase of r^2 such that there exists a new, distinct, maximal solution for $x,y \in \mathbb{Z}$. Generating these in the inverse direction by choosing $x,y \in \mathbb{Z}$ such that it is a new maximal combination not contained in any of the previous, smaller discrete discs allows the construction of the list of r^2 that defines the sequence of distinct discrete discs.

14.4 Neighbourhood and Broadcasting Sequences

The idea of propagating a pattern of square ($r^2 = 2$) or diamond waves ($r^2 = 1$), generated by repeated application of Moore or Von Neumann waves respectively, in dimension two are commonly known as Neighbourhood Sequences (see Figure 14.5). Neighbourhood Sequences are an abstraction used to study certain discrete distance metrics that are generated upon the repeated application of certain neighbourhoods to a lattice for example the Moore neighbourhood to a square lattice. As previously discussed such neighbourhoods as von Neumann and Moore are encompassed by the more general model used here whereby the transmission radius is varied on a square lattice the first two of such transmission radii to produce distinct objects equating to those of the von Neumann and Moore neighbourhoods.

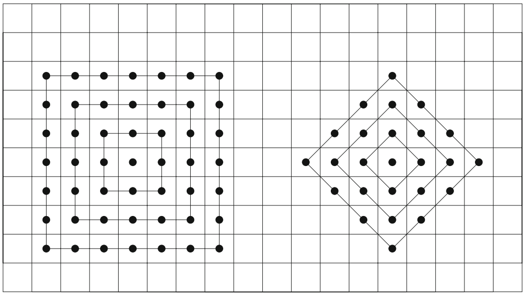


Fig. 14.5 Wave propagation with Neighbourhood Sequences on the square grid: Moore neighbourhood (left) and Von Neumann neighbourhood (right).

Definitions and notation concerning neighbourhood sequences as considered in [6], and many other works, are now given here for completeness.

Let $p \in \mathbb{Z}^n$ where $n \in \mathbb{N}$ and such that the ith coordinate of p is given by $Pr_i(p)$ for $1 \leq i \leq n$.

Definition 11. For $M \in \mathbb{Z}$ where $0 \leq M \leq n$ the points $p,q \in \mathbb{Z}^n$ are $M-Neighbours$ when the following two conditions hold:

- $|Pr_i(p) - Pr_i(q)| \leq 1$ for $(1 \leq i \leq n)$
- $\sum_{i=1}^{n} |Pr_i(p) - Pr_i(q)| \leq M$

An n-dimensional neighbourhood sequence is denoted $\mathcal{A} = (a(i))_{i=1}^{\infty}$, $\forall i \in \mathbb{N}$ where $a(i) \in 1,...,n$ denotes an *M-neighbourhood* by its value of M such that for $a(1)$ denotes that the next neighbourhood set of points in the sequence is those that differ by at most one coordinate as given by Definition 11 where $M = 1$ in this case the Von Neumann neighbourhood. If $\mathcal{A}$ is periodic then $\exists l \in \mathbb{N}$, $a(i+l) = a(i)$ $(i \in \mathbb{N})$. Such periodic sequences are given as $\mathcal{A} = (a(1), a(2), ..., a(l))$.

Definition 12. The $\mathcal{A}$-distance, $d(p,q;\mathcal{A})$, of p and q is the length of the shortest A-path(s) between them.

As the spreading of such neighbourhoods is translation invariant only an initial point of the origin need be considered w.l.o.g.

Definition 13. The region occupied after k applications of the neighbourhood sequence $\mathcal{A}$ is denoted as $\mathcal{A}_k = \{p \in \mathbb{Z}^n : d(0,p;\mathcal{A}) \leq k\}$ for $k \in \mathbb{N}$.

Also, let $H(\mathcal{A}_k)$ be the convex hull, given in Definition 14, of $\mathcal{A}_k$ in $\mathbb{Z}^n$.

Definition 14. A convex hull is the smallest set of points that form a convex set as given in Definition 15.

Definition 15. Convex set. A set $C \subset R^d$ is convex if for every two points $x,y \in C$ the whole segment xy is also contained in C. In other words, for every $t \in [0,1]$, the point $tx + (1-t)y$ belongs to C. [21]

Discrete discs are formed by the Broadcasting Automata as they are arranged on the plane at integral Cartesian coordinates, $(\mathbb{Z} \times \mathbb{Z}) \in M$, and as such a broadcast to all automata in range, for point $v \in M$, $\tau(c(v)) = r$ will cause a change in state to all those automata that form a discrete disc of radius r from the point v. In the broadcasting automata model it is assumed that there is no restriction on the radius of transmission where it is considered that as the Von Neumann and Moore neighbourhoods can be described as the first two radii in the set of distinct discrete discs, $r^2 = 1$ and $r^2 = 2$. This means that discrete discs are quite a natural extension to the basic notion of neighbourhood sequences merely relaxing the constraints on the initial definition of $M - neighbours$ in the following way.

Definition 16. Two points $p,q \in \mathbb{Z}^n$ are $r - neighbours$ if the Euclidean distance, $d(p,q) = \sqrt{\sum_{i=0}^{n}(q_i - p_i)^2}$, is less than some r, used to denote the radius of a circle, such that $d(p,q) \leq r$.

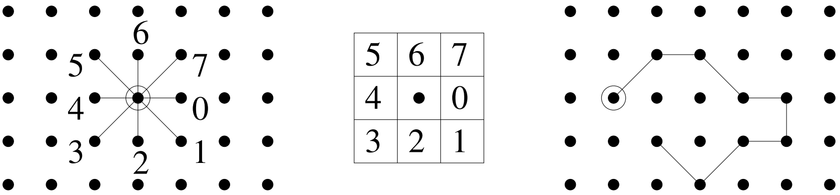
Utilising the framework supplied by the work done on neighbourhood sequences it is now possible to outline **Broadcasting sequences** which denote any sequence of radii, r, such that $R = (r_1, r_2, ..., r_l)$. Labelling those points that are reachable by some application, R_k, of the neighbourhood sequence is another extension to the notation. All points such that $p \in R_1$ are labelled 0, all points $p \in R_2 \backslash R_1$ are labelled as 1. More generally labels will be assigned b where $k \equiv b \mod m$ and $m \in \mathbb{N}$. The work conducted here will also be restricted to the study of $\mathbb{Z}^2$.

Many of the questions asked and, indeed, answered, in neighbourhood sequences, shall be useful in the exposition of broadcasting automata. These include whether a certain sequence of neighbourhoods are metrical or not [23]. Indeed when using a general form of broadcasting automata whereby alternation of the broadcasting radius is allowed at each step as suggested here. Considering that all notions of the construction of algorithms presented here in the broadcasting automata model rely on the distance from the transmitter to the node, being able to assure that this distance will be consistent for all automata is essential. Also studied in neighbourhood sequences is the possible resultant shapes which are categorised and examined for their various properties such as their use in approximation of euclidean distances where the isoperimetric ratio is used to compare the sequences based on the shapes that they form on the plane [7]. This same method is used to again show an estimation for euclidean distance as well as estimations for certain L_p distances, here the astroid which will be presented in Section 14.10. The characterisation of the shapes generated by these neighbourhood sequences yield results about the shapes of compositions of such discrete discs on the plane and the partitions that such compositions form.

14.5 Geometrical Properties of Discrete Circles

The characterization of broadcasting sequences on $\mathbb{Z}^2$ is closely related to the study of discrete circles on the square lattice and their discrete representation. One of the efficient methods to describe the discrete circle is a chain coding. The method of chain coding was first described in [10] as a way in which to encode arbitrary geometric configurations where they were initially used, as they shall be here, to facilitate their analysis and manipulation through computational means. At the time there were many questions about the encoding of shapes and the methodologies which should be used to encode such shapes and chain coding presented an schema which was simple, highly standardised and universally applicable to all continuous curves. Whilst the definition given here differs from the definition given in [10] it only differs for convenience when discussing discrete discs in the first octant allowing the use of only the numeric symbols 0 and 1 instead of 0 and 7 as Freeman suggested. This gives the following definition of chain coding that shall be used throughout.

Definition 17. From a starting point on the square lattice a **chain code** is a word from the alphabet $\{0, 1, 2, 3, 4, 5, 6, 7\}$.



5	6	7
4	●	0
3	2	1

Fig. 14.6 (Left) Showing the eight possibilities of motion from the single point of origin, circled, as they may be traced out on the plane (Centre) Shows the eight possible points of motion from a central point and (Right) gives and example of an arbitrary line traced out on the lattice with a chain code of form 70102435 originating from the circled point.

From some starting point chain codes may be used to reconstruct a shape, S, by translating the code's motion on to the lattice where 0 indicates positive movement along the $x - axis$ with increasing values moving clockwise through all 8 possible degrees of motion on the lattice, labelled from 0 to 7 respectively. An integral shape, S, can be encoded using an inverse method whereby the shape is traced from some starting point and the motion from one point to the next connected point in the shape is recorded as a chain code. For example, a straight line would be represented by the infinite repetition of a single digit such as 00000000000000..., a line that satisfies the equation, $x - y = 0$, may be encoded 77777777... or as the example given in Figure 14.6 and arbitrary line may be encoded numerically as 7010235.

The discrete disc is the basic descriptor of a transmission in broadcasting automata. It represents the total set of all automata that could be reached on the square lattice after a single broadcast of radius, r^2. In the following sections reasoning will follow only from the first octant, which can be seen in Figure 14.7, of the discrete circle which shall now be defined and gives all the information that is required to categorise the discrete disc, its perimeter, but also has a convenient method of chain code construction. The need only to observe the first octant is derived from the symmetry that occurs such that it is possible to take a solution of the first octant and by successive reflection operations about the eight octants generate a full circle [2] under the assumption, which is also made here, that the circles are centred on a integral point on the plane.

Definition 18. A discrete circle is composed of points in the $\mathbb{Z}^2$ set $\zeta = \{(x,y)|x^2 + y^2 \leq r^2,\ x,y \in \mathbb{Z},\ r \in \mathbb{R}\}$ that have in their Moore neighbourhood any point in the complement of ζ.

In Definition 18 it is shown that $r \in \mathbb{R}$ however many of these do not produce distinct discrete circles. The set of r that includes all distinct circles are those which form Pythagorean triples which consist of two positive integers $a,b \in \mathbb{N}$, such that $a^2 + b^2 = r^2$.

A simple way to draw the discrete circle is to divide it into octants as can be seen in Figure 14.8. First calculate only one octant of the circle and as the rest of the circle is "mirrored" from the first octant it can now be successively mirrored to allow the construction of the other sections of the circle. Let us divide the

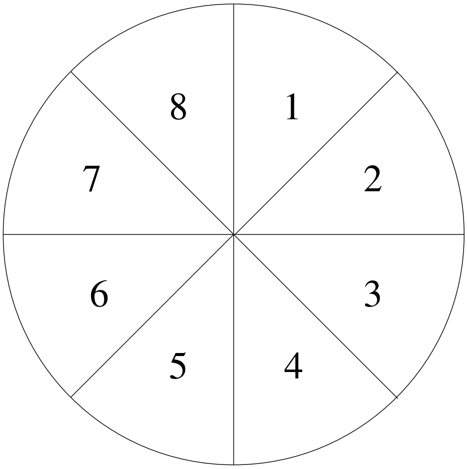


Fig. 14.7 An illustration of a circle that has been divided in to eight octants with the labelling of those octants applied accordingly.

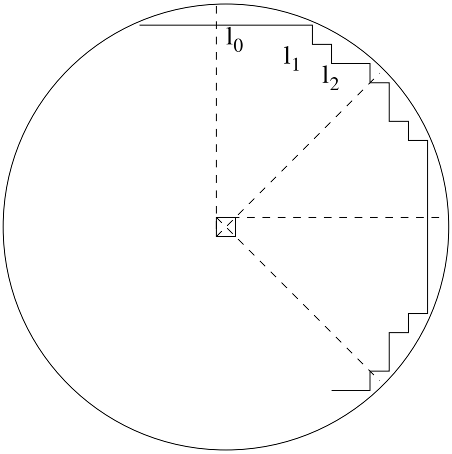


Fig. 14.8 An illustration of a discrete disc and its accompanying discrete circle of radius squared 116, where the chain code for the first octant is labelled as $l_0 l_1 l_2$.

circle into eight octants labelled 1-8 clockwise from the y-axis. The notation $Octant(i)$ is used to refer to octant sector i. If an algorithm can generate the discrete circle segment on one octant, it can be used to generate other octants by using some simple transformations, such as rotation by 90° and reflection around the x-axis, y-axis, and the $+45°$ diagonal lines.

Some basic results are now given about chain codes of such discrete circles. It is possible to show that in the first octant and given the definition of chain coding presented here only two digits, 0 and 1 are required to fully represent the discrete circle.

Lemma 1. *A Chain coding w for the circle in the first octant is a word in* $\{0,1\}^*$.

Proof. First let us show that in the first octant for any given point $p_0 = (x_0, y_0)$ on the circle at the point $p_1 = (x_0+1, y_0-1)$ must also be in the circle as $|p_1| < |p_0|$. When p_0 is in the circle and in the first octant segment such that $x < y$ it follows that the magnitude of p_1 is less than p_0 as $|p_0| - |p_1| = (x_0^2 + y_0^2) - (x_1^2 + y_1^2) = (x_0^2 - x_1^2) + (y_0^2 - y_1^2) = (x_0^2 - (x_0+1)^2) + (y_0^2 - (y_0-1)^2) = (x_0^2 - x_0^2 - 2x_0 - 1) + (y_0^2 - y_0^2 + 2y_0 - 1) = 2y_0 - 2x_0 - 2 \geq 0$. Since $p_1 = (x_0+1, y_0-1)$ should be in the circle if $p_0 = (x_0, y_0)$ is on the circle we have that any chain code for the circle in the first octant does not contain a subword "22" as it will lead to contradiction. Moreover in any subword a single symbol "2" should be followed by "0" and can be substituted by 1.

As only the first octant segment is considered the solutions can be represented using strings consisting of only 0 and 1 where 0 is positive motion along the $x-axis$ and 1 is positive motion along the $x-axis$ and negative motion along the $y-axis$.

Indeed in the generation of such chain codes the following method shall be used which is a modification of the algorithm that is given in [1]. Starting with the equation for the circle $x^2 + y^2 = r^2$ where it is known, by definition, that $r^2 \in \mathbb{Z}$. Now as solutions of the first octant are required and under the assumption that, without loss of generality, the centre of the circle is at the origin it is now possible to replace y^2 with $(r' - k)^2$, where $r' = \lfloor r \rfloor$, such that when $k = 0$ and rearranging for x^2 such that $x^2 = r^2 - r'^2 - k^2 + 2r'k$ the solution to the circle equation gives $x^2 = r^2 - r'^2$ where $r' \leq r$, as such the length of the first solution of such an equation is $\sqrt{r^2 - r'^2}$. Successive values of k give solutions to the length in the x-axis of that particular maximal Pythagorean triangle. All such Pythagorean triangles will give the final solution to the circle.

Here it is necessary to start with some terminology to say more precisely which parts of the chain code are considered to represent specific components in the resulting polygons. It will become much clearer later on why such distinctions are required as a single side of the polygon may differ in its chain code categorisation. The first of these categories of chain code, the most basic, is given here.

Definition 19. A **chain code segment** is a subword of a chain code which is either a word $0^{|s_0|}$ or $10^{|s_i|-1}$.

A discrete circle, u, is composed of chain code segments where each segment is represented as u_i, i.e $u = u_0 u_1 u_2 ... u_n$. Chain code segments can be expressed using a power notation for the number of repetitions e.g 100100 may be rewritten as $(100)^2$ and if $u = u_0 u_1 u_2 u_3 = (00)(100)(100)(10) = (00)^1(100)^2(10)^1$.

It is possible to map the chain code, u, in $Octant(1)$, to the others octants through a function that maps the alphabet of the chain code $\{0,1\}$, to a new code in another octant segment. A chain code of a circle in an $Octant(n)$ can be constructing from an $Octant(1)$ first by applying the following mapping $\{0,1\} \rightarrow \{n-1 \mod 8, n \mod 8\}$, where $0 \leq n \leq 7$. Then for the $Octant(n)$, where $n \equiv 1 \mod 2$, or n is odd, the code and the mapping itself must be reversed such that $\{0,1\} \rightarrow \{n \mod 8, n-1$

mod 8$\}$. For the rest of this paper we will only consider chain codes in the first octant.

Proposition 2. *A circle in the first octant can be chain coded as follows:*

$$0^{|s_0|}10^{|s_1|-1}10^{|s_2|-1}\dots10^{|s_i|-1}\dots10^{|s_{r'^2/2}|-1}$$

where,

$$s_i = \{x|r^2 - r'^2 + 2(i-1)r' - (i-1)^2 < x^2 \leq r^2 - r'^2 + 2ir' - i^2, x \in \mathbb{Z}\}$$

r is the radius and r' is the floor of the radius.

Proof. It follows from [1] where such a method of chain coding is introduced.

Proposition 3. *Chain code segments, following the schema*

$$0^{|s_0|}10^{|s_1|-1}10^{|s_2|-1}\dots10^{|s_i|-1}\dots10^{|s_{r'^2/2}|-1}$$

have the following constraints to their lengths in the discrete circle:

$$\lfloor\frac{s_i-1}{2}\rfloor - 1 \leq s_{i-1} \leq s_i + 1 \ .$$

Proof. It, again, follows from [1] where this property is proved.

As previously noted terminology must be introduced to distinguish the varying parts of the chain code and in order to talk about what they represent. In order to characterise the discrete circle two notions are introduced, *Line Segments* and *Gradients of Line segments*.The following definition for the gradient gives a method for extracting such information from the series of digits that construe the line as a word.

Definition 20. A **gradient**, $G(u)$, of a chain code subword in $\{0,1\}$, u, is given by $G(u) = \frac{\#_1(u)}{|u|}$, where the function $\#_1$ returns the number of $1's$ found in the chain code and $|u|$ is the length of word u.

Line segments are in reference to actual lines that these sections of chain code would represent on the plane. That is if one were to draw a line and then attempt to translate this line in to a chain, under the assumption that the gradient of the line is rational, for reasons which can be seen in the definition of a gradient, it would be done via a periodic series of chain code segments. Taking a single period of this chain all the information that is required to identify and reconstitute this line to any length is now known. In this way the name may interpreted quite literally as a segment of a line in chain coded form such that any repetition of this object produces a line. With this in mind it will naturally be of benefit to be able to know a little more about the line that has been chain coded as a line segment.

Definition 21. A **Line segment**, $l_i = u_j u_{j+1} ... u_{j+n}$, is the shortest contiguous subsequence of chain code segments which maintain an increasing gradient such that $G(l_i) < G(l_{i+1})$, $\forall i > 0$, where the last chain code segment in l_i, $u_j + n$, is contiguous to the first chain code segment in the next line segment, that is, $l_{i+1} = u_{j+n+1} u_{j+n+2} ... u_{j+n+m}$.

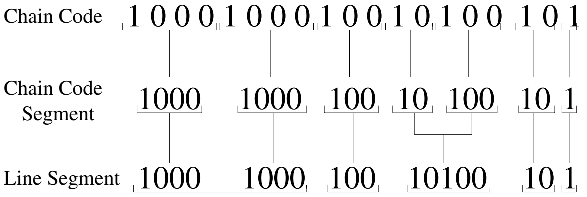


Fig. 14.9 Showing the the relationship between chain codes, chain code segments and line segments

As, ultimately, all of the discrete discs that are discussed here, and, more importantly, that are possible to construct, are polygons, such polygons are composed of line segments which represent the sides of the convex shape on the plane. Further, when categorising the repeated broadcast of such discrete discs, later discussed as composition, the shapes that are generated are known to be similar, in the strict geometric sense. A definition for chain codes, translating from the geometry of the Reals, is given.

Definition 22. For any two chain codes, u, v, comprising line segments such that, $u = l_0 l_1 ... l_m$ and $v = l'_0 l'_1 ... l'_n$, for $m = n$, given by Definition 21, are similar (geometrically) if there is a constant $k \in \mathbb{N}$ such that $u = l'^k_0 l'^k_1 ... l'^k_n$.

In this way it is possible to say that two shapes are similar in a geometric sense such that they contain the same number of distinct line segments but the number of each of these line segments is different by some constant.

With these definitions stated it is possible to begin the categorisation of the set of discrete circles with the first observation which extends an already known notion about the chain code of the discrete disc. Following [1] it is known that in the chain code u any chain code segments with increasing lengths, such that $|u_i| < |u_{i+1}| < ... < |u_{i+n}|$, may increase it by at most 1 from the length of a previous segment, i.e. $|u_{i+1}| \leq |u_i| + 1$ for all i.

By direct construction it is easy to check that a line segment can be of the form 10^n or $10^n 10^{n+1}$. However we can show that no line segment may be composed of more than two chain codes which increase in length by one.

Lemma 2. *No line segment can be of the form* $10^n 10^{n+1} 10^{n+2} \dots 10^{n+i}$, *where* $i > 1$.

Proof. Let us first show that any line segment which is part of the discrete circle cannot be of the form $10^n 10^{n+1} 10^{n+2}$ for any $n > 1$. We will prove it by contradiction. Assume that the line segment $l_i = 10^n 10^{n+1} 10^{n+2}$ with the gradient $1/n + 1$ starting

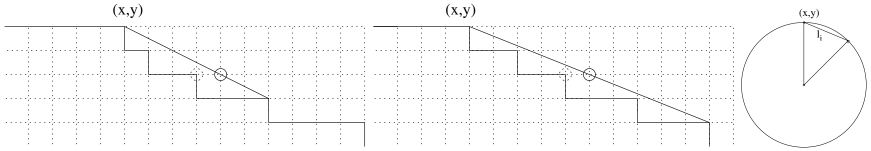


Fig. 14.10 (Left and Centre) Showing the initial point (x,y) the chord and the point on the chord, the solid circle. (Right) Showing a chord on a circle from (x,y)

at a point p with coordinates (x,y) and finishing at $(x+3n+6, y-3)$. Therefore a point $(x+2(n+2), y-2)$, i.e. a point which can be reached by a code $10^n 10^{n+2}$ from a point (x,y), is above the discrete circle. By the definition of the chordal property of the circle any line that joins two points on the circle must bound, within the triangle formed from the two points on the circle and its centre, points which are again within the circle. On the other hand a point $(x+2(n+2), y-2)$, shown circled in Figure 14.10, belongs to a chord between end points of a chord (x,y) and $(x+3n+6, y-3)$ and therefore should be within a discrete circle. So it is not possible to have a line segment with the following chain code $10^n 10^{n+1} 10^{n+2}$. The same argument holds for the general case $10^n 10^{n+1} 10^{n+2} \dots 10^{n+i}$, where $i > 1$ since the extension of the line segment still keep the point $(x-2, y+2(n+2))$ above the discrete circle and on the other hand this point will be in the triangle formed by a centre of the circle and two end points of the line segment since its gradient is equal to $\frac{i+1}{n(i+1)+(i+1)+i(i+1)/2} = \frac{1}{n+1+i/2} \leq \frac{1}{n+2}$, where $i \geq 2$.

Further to this weak restriction, that the chain codes may not be monotonically increasing or indeed non-decreasing, it is possible to give a more definite generalisation of the structure of the chain codes. This structure, which must be adhered to for all discrete discs, is given here.

Theorem 1. *Any line segments on the discrete circle with non-negative gradients should be in one of the following forms* $(10^n)^*$, $(10^n)(10^{n+1})^*$, $(10^n)^*(10^{n+1})$.

Proof. Following Lemma 2 we can restrict number of cases since any concatenations of more than two chain codes which increase in length by one may not be part of a line segment. The line segments are sub-words of a chain code with increasing gradients, so if a subword $(10^n)^m$ is surrounded by any chain code segments 10^{n_1} and 10^{n_2}, where $n_1, n_2 \neq n \pm 1$, then $(10^n)^m$ is a line segment. In the case were $n_1, n_2 = n \pm 1$ a subword $(10^n)^m (10^{n+1})^k$ can be surrounded by only chain codes 10^{n_3} and 10^{n_4}, where $n_3 \neq n-1$ and $n_4 \neq n+2$ (by Lemma 2). We will show now that the only line segments of the form $(10^n)^m (10^{n+1})^k$ can be where either $m = 1$ or $k = 1$. The proof is similar to Lemma 2. Let us first show that the line segment from a point (x,y) cannot have the following chain code $(10^n)^2 (10^{n+1})^2$. If we assume that such chain code may correspond to a line segment such that a point $(x-2, y+2n+3)$, shown in the centre diagram in Figure 14.10, is above the chain code (i.e. our of a circle) and at the same time is on a chord between points (x,y) and $(x-4, y+4n+6)$, so should be in a discrete circle. Extending the chain code $(10^n)^2(10^{n+1})^2$ from the left by $(10^n)^{m_1}$ or from the right by $(10^{n+1})^{m_2}$ for any $m_1, m_2 > 0$ will not change the property of the point $(x-4, y+4n+6)$. So it can be in one of the following forms either $(10^n)^m 10^{n+1}$ or $10^n(10^{n+1})^k$.

14.6 Iterative Composition

Iterative composition of discrete circles may create different polygons, which later can also be used for forming non-convex shapes. The following definitions, and further characterisations, shall be required in order to discuss the properties of such composition and develop the proofs.

Definition 23. A discrete disc is composed of points in the $\mathbb{Z}^2$ set $\zeta^{r^2} = \{(x,y)|x^2 + y^2 \leq r^2,\ x,y \in \mathbb{Z},\ r \in \mathbb{R}\}$.

Definition 24. The composition of l digital discs is defined as $\zeta^{r_0} \circ \zeta^{r_1} \circ \ldots \circ \zeta^{r_l}$ for $r_i \in \mathbb{N}$, where, $\zeta^r \circ \zeta^{r'} = \{a+b | a \in \zeta^r,\ b \in \zeta^{r'}\}$ and is equivalent to $H(\mathcal{A}_l)$ where $\mathcal{A} = r_0, r_1, \ldots, r_l$ as given in Definition 14 and Definition 16 respectively.

Definition 25. The composition $u \circ v = w$ represents the composition of the chain codes in the first octant for their respective discs, ζ^u and ζ^v, which results in w the chain code for the first octant of the resultant disc of $\zeta^u \circ \zeta^v$. This method can be seen in Figure 14.11.

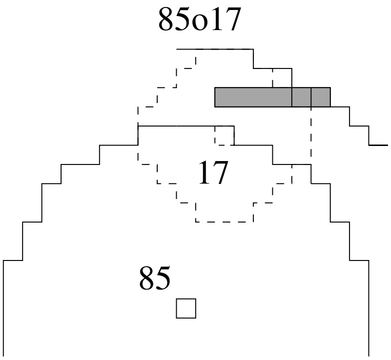


Fig. 14.11 Showing the composition of squared radii $Disc(85) \circ Disc(17)$ where $Disc(85)$ is shown as a solid line and 17's first composition is shown dashed. The hatched area represents all contributions from the previous compositions.

Example 1. Iterative composition of discrete circles may create different polygons. For example, in the following picture, Figure 14.12, there are two differing cases, in one case we iteratively apply the digital circle with squared radius 9 resulting in the equilateral octagon. In the second case squared radii 16 and 26 are periodically applied starting with 16.

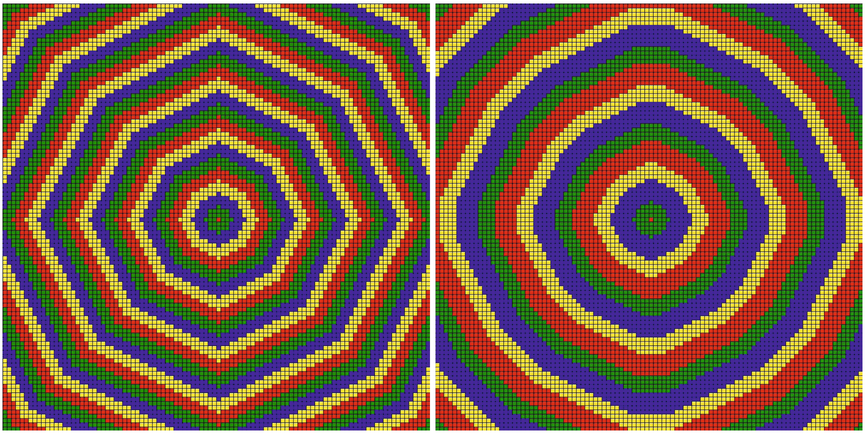


Fig. 14.12 The above figures illustrate (left) the constant iteration of the discrete disc of squared radius 9 and (right) the alternation between the transmission of squared radii 16 then 26.

From these definitions it is now possible to describe a naive algorithm, and derive its correctness, for the composition of any number of chain codes that represent discrete discs. The result is again a convex polygon that represents the combination of the constituent parts. Such an algorithm merely systematically checks all possible combinations of the chain code and reports the largest that is found for that step. Later it shall be seen that if the chain code of the discrete disc is categorised further it is possible to give a simple linear time algorithm with a proof of correctness that is derived from the Minkowski sum along with a combinatorial proof.

Lemma 3. *Given two chain codes, $u = u_0u_1...u_n$ and $v = v_0v_1...v_m$, in form $\{0,1\}^*$ then $u \circ v = w = 0^{|w_0|}10^{|w_1|-1}...10^{|w_{m+n-1}|-1}$, such that*

$$|w_k| = max(\{\sum_{i=0}^{n} |u_i| + \sum_{j=0}^{k-n} |v_j| \mid 0 \leq n \leq k\}).$$

Proof. The naive way of generating the composition $u \circ v$ is for all points on the chain code of the disc $u = 0^{|u_0|}10^{|u_1|-1}...10^{|u_i|-1}...10^{|u_n|-1}$, to be the centre of the disc $v = 0^{v_0}10^{|v_1|-1}...10^{|v_j|-1}...10^{|v_m|-1}$. This is equivalent to placing the centre of the chain code v at every point defined by the chain code u generating w. The maximal point in each chain code segment (i.e if $u_i = 100$ then the coordinate of the final 0) need only be considered, clearly covering all others. Let $u_i \circ' v_j = w_{i+j}$ denote the centring v at the coordinate reached by the maximal point of u_i in which we consider the maximal point attained by chain code segment v_j and represent a possible length of the chain code for w at w_{i+j}. The maximal of all such combinations of lengths, those for which the sum of i, j are equivalent, is required and is defined as the longest contiguous subsequence of length k from $v = v_0 v_1 ... v_j$ and $u = u_0 u_1 ... u_i$ for all i, j such that $i + j = k$ which represents w_k for $0 \leq k < m+n$ i.e:

$$max\left(|u_0| + |v_0|\right) = |w_0| \qquad max\begin{pmatrix}|u_0| + |v_0| + |v_1| \\ |u_0| + |u_1| + |v_0|\end{pmatrix} = |w_1|$$

$$max\begin{pmatrix}|u_0| + |v_0| + |v_1| + |v_2| \\ |u_0| + |u_1| + |v_0| + |v_1| \\ |u_0| + |u_1| + |u_2| + |v_0|\end{pmatrix} = |w_2|$$

$$...$$

$$max\begin{pmatrix}|u_0| + |v_0| + |v_1| + ... + |v_k| \\ |u_0| + |u_1| + |v_0| + ... + |v_{k-1}| \\ ... \\ |u_0| + |u_1| + ...|u_i| + |v_0| + ... + |v_{k-i}| \\ ... \\ |u_0| + |u_1| + ... + |u_k| + |v_0|\end{pmatrix} = |w_k|$$

$$...$$

$$|u_0| + |u_1| + ... + |u_n| + |v_0| + |v_1| + ... + |v_m| = |w_{m+n}|.$$

Such an algorithm allows a characterisation of the composition of discrete discs. Indeed, here, it can be shown that the composition of chain codes, which have been shown to be a word $W \in \{0,1\}^*$, is, again, a word, $W' \in \{0,1\}^*$.

Corollary 1. *Given two chain codes u, $v \in \{0,1\}^*$. The composition $u \circ v \in \{0,1\}^*$*

Proof. Each segment in u, v, is non-zero $\forall i \ |u_i| \neq 0, |v_i| \neq 0$. Its follows from Lemma 3 that by enlarging k, for $l(k+1)$, the previous set of solutions is extended by a chain code from u or v so $l(k) < l(k+1)$. Thus the composition will be in $\{0,1\}^*$.

Also as a derivation from the algorithm an observation about the properties of the composition function $\circ$.

Theorem 2. *The composition $\circ$ of two chain codes, u and v is commutative, i.e. $u \circ v = v \circ u$.*

Proof. Following Lemma 3 it is possible to exchange u and v and still obtain the same result, showing commutativity of composition of chain codes u and v. The following two sets representing the k-th level of a new chain code are equal as well as their maximums:

$$\{S|S=\sum_{i=0}^{n}|u_i|+\sum_{j=0}^{k-n}|v_j|,0\le n\le k\}=\{S|S=\sum_{i=0}^{n}|v_i|+\sum_{j=0}^{k-n}|u_j|,0\le n\le k\}$$

It is also notable that having shown commutativity chain code composition it may be possible that it is an Abelian group, where the identity element is simply the circle of radius 0, associativity is derived from the algorithms ordering of line segments by gradient, which is naturally the same irrespective of the order it is carried out in, the inverse is simply the removal of one set of line segments from a chain code. However, it is currently unknown whether or not the operation is closed.

Further, a proof of the ordering of the line segments in the discrete circle is here required in order to validate the proof of the composition theorem that is given as one of the main tools and results of this section.

Lemma 4. *Given line segments, l_i, of a form produced by the digitisation of a circle, which are composed of chain code segments of its first octant $a = 10^{m-1}$ and $b = 10^m$, then the ordering of their gradient for combinations of a and b is*

$$G(b) < ... < G(ab^*) < ... < G(ab) < ... < G(a^*b) < G(a)\ .$$

Proof. The proof is a simple comparison of the gradients which shows, where $|a| = m$, $\frac{1}{m+1} < \frac{n}{(n-1)(m+1)+m} < \frac{n-i}{(n-1-i)(m+1)+m} < \frac{n-i}{(n-1-i)(m)+m+1} <$
$< \frac{n}{(n-1)(m)+m+1} < \frac{n}{(n-1)m+m+1}$, where $i < n-1$, for $G(b) < G(ab^{n-1}) < G(ab^{n-1-i})$
$< G(a^{n-1-i}b) < G(a^{n-1}b) < G(a)$ respectively.

Given the preceding validation about the convexity of the discrete discs it is now possible to employ the Minkowski sum to conclude and validate the composition of any number of discrete discs using the composition function.

Theorem 3. Composition Theorem *Given two chain codes u and v which contain line segments $l_1^u l_2^u \dots l_t^u$ and $l_1^v l_2^v \dots l_{t'}^v$ with strictly increasing gradients. The chain code of a composition $u \circ v$ can be constructed by combining line segments of u and v and ordering them by increasing gradient.*

Proof. One of the ways to prove the above statement is to use a similar result about the Minkowski sum [25] of convex polygons in $\mathbb{R}^2$ and then to prove that it holds for the ordering of the segments of the digital circles in $\mathbb{Z}^2$. Here, a purely combinatorial proof is given in terms of chain codes, chain code segments and line segments. The above statement is proved by induction. The fact that the base case for the composition of the first two line segments holds can be seen by directly checking the expression from Lemma 3.

Assume that the statement of the lemma holds and the first z line segments of $u \circ v$ were composed from a set $LS = \{l_1^u, l_2^u, \dots, l_i^u, l_1^v, l_2^v, \dots, l_j^v\}$ and contains x chain code segments. Let us also denote a set with other line segments from u and v as $L\bar{S}$. Without loss of generality suppose that the last line segment that has been added is l_j^v, so $G(l_j^v) \ge G(l')$, where $l' \in LS$ and $G(l_j^v) \le G(l'')$, where $l'' \in L\bar{S}$. By adding the next line segment l''' which will have a gradient larger or equal than all other line segments in LS and smaller or equal than gradients of $\bar{LS}$ it must be shown that the extended chain code of $u \circ v$ is still correct, i.e. it satisfies to Lemma 3.

First of all note that adding a new part of the chain code would not change the previous x layers. If a current maximum is in the form

$$|u_p| + |u_{p-1}| + ... + |u_0| + |v_0| + |v_1| + ... + |v_q|$$

then the next $\#_1(l''')$ sums will be extended by chain codes from l'''. If l''' is a line segment in u the gradient of $G(l^v_j)$ is greater then $G(l''')$ and, taking into account Lemma 4 and Theorem 1, it can be seen that

$$|u_{p+1}| + |u_p| + |u_{p-1}| + ... + |u_0| + |v_0| + |v_1| + ... + |v_q|$$

is a maximum within the following set

$$\{|u_{p+1+shift}| + ... + |u_p| + |u_{p-1}| + ... + |u_0| + |v_0| + |v_1| + ... + |v_{q-shift}|, shift \in \mathbb{N}\}$$

since the shift will represent removal of a larger value from the v component and appending the smaller value from the u component.

Repeating the procedure and extending the sum by one value it can be seen that, again,

$$|u_{p+2}| + |u_{p+1}| + |u_p| + |u_{p-1}| + ... + |u_0| + |v_0| + |v_1| + ... + |v_q|$$

is a maximum within a set

$$\{|u_{p+2+shift}| + ... + |u_p| + |u_{p-1}| + ... + |u_0| + |v_0| + |v_1| + ... + |v_{q-shift}|, shift \in \mathbb{N}\}$$

following the same reasoning. Similar arguments can be applied for the case were $l''' \in v$.

Example 2. Let us illustrate the composition of two chain codes corresponding to digital circles with squared radii 45 and 9 in the first octant. The chain code of the first octant is $\zeta^{45} = 0001$ and for $\zeta^9 = 01$ under composition this yields $\zeta^{45} \circ \zeta^9 = 000101$.

From the preceding notions it is now possible to describe a linear time algorithm, which vastly out performs the exhaustive search of all combinations that has previously been given, which is able to form the composition of the chain codes of any two discrete circles. As Lemma 4 shows that all of the line segments will be in a non-increasing order it is possible to construct such line segments by counting the number of $0's$ after each delimiting 1 combining those that increase in size with the preceding smaller chain code segment. Having found all line segments for both chain codes it is a simple case of, starting with the chain code with the line segment with the largest gradient, adding that element to a new chain code which is the composition of the initial two. Continue choosing the line segment with the largest gradient from either of the initial chain codes until there are no elements left in either of the original circles. The result of the algorithm is the composition of the two circles. The following proposition gives a proof of the algorithm and the notion that it is linear time computable.

Proposition 4. *Two discrete circles chain codes can be composed into a single chain code in a linear time.*

Proof. As Lemma 4 shows that all of the line segments will be in a non-increasing order it is possible to construct such line segments by counting the number of $0's$ after each delimiting 1 combining those that increase in size with the preceding smaller line segment. Having found all line segments for both chain codes it is a simple case of starting with the chain code with the line segment with the largest gradient adding that to a new chain code which is the composition of the initial two.

It is also now possible to state a more formal definition of 'similar' such that it is possible to mathematically determine whether two objects are 'similar', geometrically. The following algorithm is given as one application of such an algorithm and also hints at the notion of composing one discrete disc from other discrete discs which allows an insight in to the primality of discrete discs those that cannot be composed from any other set of discrete discs.

Theorem 4. *Given a finite broadcasting sequence of radii $R = (r_1, r_2, ..., r_l)$ and a convex polygon P, it is decidable whether there are radii such that the chain coding of the composition of is* **similar** *to P.*

Proof. The algorithm computes all line segments, and their corresponding gradients, for the chain codes of the set of digital disks with radii in, $\mathbb{R}$, and the convex polygon, P. Each digital disk $r_i \in R$ can be represented as a vector with k values, where k is the cardinality of the set of gradients and each vector component corresponds to a particular gradient with an integer value standing for the number of line segments with this gradient in R. Finally we would need to solve a system of linear Diophantine equation over positive integers to check whether there is a set of factors for defined vectors that may match a vector for P with another unknown factor.

14.7 Broadcasting Sequences and Their Limitations

Although the composition of circles gives us a large range of polygonal shapes, an analysis of broadcasting sequences shows that there certain limitations for such composition. It's seen that by Lemma 2 there are only three possible forms which the lines that comprise the discrete circle may take. Translating these in to their respective gradients gives the following three possible gradient forms

$$G_1 = G((10^{n-1})^m) = \frac{m}{mn} = \frac{1}{n}$$

$$G_2 = G((10^{n-1})(10^n)^m) = \frac{m+1}{n+m(n+1)}$$

$$G_3 = G((10^{n-1})^m(10^n)) = \frac{m+1}{nm+n+1} \,.$$

The above gradients are reduced to a minimal form in order to elucidate the exclusivity of the gradients.

Proposition 5. *It is only possible to express gradients in the first octant of a circle in a reduced form* $\frac{1}{n}$, $\frac{a}{a(n+1)-1}$ *or* $\frac{a}{an+1}$ *for* $a, n \in \mathbb{N}$.

Proof. As G_1 is already in a minimal form it is obvious that this can only express those gradients, g, of the form $\frac{1}{n}$. For G_2 and G_3 the following method is employed, $G_2 = \frac{m+1}{n+m(n+1)} = \frac{a}{b}$, where $a \perp b$ from which it follows that $m = a-1$ such that through substitution of m in to denominator, $\frac{a}{a(n+1)-1}$ is arrived at. Similarly for G_3 by substitution the equation $\frac{a}{an+1}$ arises.

It now becomes more clear of what cannot be expressed and the limitations inherent in the hulls of the broadcasting sequences.

Proposition 6. *Not all rational gradients,* $0 \leq g \leq 1$, *are expressed by the lines that comprise the discrete circle in the first octant.*

Proof. A counter example is given as proof. It is impossible to express any such rational of the form $\frac{5}{8}$. It is clear that it is not possible for G_1 to express such a fraction. For G_2 and G_3 the following suffices. $G_2 = \frac{a}{a(n+1)-1}$ where $a = 5$ such that $\frac{5}{5n+4}$ where there is no such $n \in \mathbb{N}$ such that $G_2 = \frac{5}{8}$. Similarly for $G_3 = \frac{a}{an+1}$ where $a = 5$ and $\frac{5}{5n+1}$ there is no such $n \in \mathbb{N}$ such that $G_3 = \frac{5}{8}$

It also becomes clear from the composition theorem (Theorem 3) itself that it is not possible to generate any further gradients or new line segments through composition and in turn successive broadcasts of radii may not generate any polygons with chain codes that include gradients not previously present.

Corollary 2. *The set of line segments, and as such gradients, that compose any discrete circle are closed under composition.*

As noted here the shapes that are generatable are limited by their convexity as well as by the gradients of the lines that compose them. In the next section some of these limitations will be removed or relaxed using multiple broadcast sequences and an aggregation function with which to map all values to a single value.

14.8 Reducing Restrictions through Aggregation

This section makes uses of the notion of aggregation to reduce the restrictions that are imposed by the continual composition, or simple construction of discrete discs. It can be shown that it is possible in certain cases, which shall be exposited both here and in further sections where it is employed, to show the approximation of the astroid metric.

As all points may be labelled by their distance over some arbitrary modulo value an extension to the current work is proposed whereby two differing $r-neighbourhood$ sequences, A and A' are used to label the $\mathbb{Z}^2$ lattice from the same point p. At any point, p', there are now two labels such that $p' = (i, j)$ where by $k \equiv i \mod m$ for the sequence A and $k' \equiv j \mod m$ for the sequence A'. Here two differing functions for the aggregation of values of i and j which define new shapes on the lattice and in turn new metrics. The new shapes defined by the lattice are not necessarily convex.

As all of the discrete disks which make up the labellings of the lattice are composed of discrete lines it is possible to analyse the effects of the combinations of disks by considering only the intersections of the lines that make up the disks. Consider a series of parallel lines expressed in the form $y_0 = m_0x_0 + c_0$ where $c_i \in \mathbb{Z}$ is an arbitrary constant or offset, $m_i \in \mathbb{Q}$ is any gradient permitted by the line segments of a discrete circle and $x, y \in \mathbb{Z}$ are the usual Euclidean coordinates. These lines are such that each successive line is of a distance w_i from the last which shall imitate the width between iterations of the discrete circles, for the first line segment this is equivalent to $\lfloor\sqrt{r^2}\rfloor$, and the discrete lines that they generate. All coordinates such that $m_0x_0 + c_0 + kw_0 \le y_0 < m_0x_0 + c_0 + (k+1)w_0$ are labelled as k_0. A second set of parallel lines differing from the first are defined as $y_1 = m_1x_1 + c_1$ where all coordinates such that $m_1x_1 + c_1 + k_1w_1 \le y_1 < m_1x_1 + c_1 + (k_1+1)w_1$ are labelled k_1. The intersection of these two areas is thus (k, k'). Increasing the offset of k_0 and k_1 to $k_0 + 1$ and $k_1 + 1$ naturally results in the labelling of the area of their intersection as $(k_0 + 1, k_1 + 1)$. The ordering of the tuple is not relevant to the functions that aggregate them.

The first of the functions here have previously been studied in [19] with regards to geometric computations on the lattice. It is of particular interest in the Broadcasting Automata model where the notion of waves as observed in nature are used to control a large number of distributed automata. Being able to predict the resultant shapes that are formed by the transmission of waves is, naturally, largely advantageous in that it affords the ability to predict and manipulate the formations on the plane. Such formations can then be used for a variety of computational duties such as partitioning and geometric computation.

Two functions for aggregation will now be introduced with the relevant equations for resultant patterning of the lattice, where $m = 4$ and the values of $i, j \in \{0, 1, 2, 3\}$.

Definition 26. The **moiré** aggregation function is given in the table below.

moiré(i,j) =

$\oplus$	0	1	2	3
0	a	b	c	b
1	b	a	b	c
2	c	b	a	b
3	b	c	b	a

Definition 27. The **Anti-moiré** aggregation function can be expressed simply as addition over modulo 4 and is given in the table below.

Anti-moiré(i,j) =

$\oplus$	0	1	2	3
0	a	b	c	d
1	b	c	d	a
2	c	d	a	b
3	d	a	b	c

Proposition 7. *The gradients of the new lines formed by the moiré equation can be predicted using the equation*

$$\frac{w_o \cdot m_1 - w_1 \cdot m_0}{w_0 - w_1}\,.$$

Variables refer to the, line gradient, m_i, and the width of the line, w_i.

Proof. The form of the moiré function is such that if the intersection of one area is labelled as (k,k') then the next area that is labelled similarly will be of the form $(k+1,k'+1)$. Encoding this as the intersections of lines $y = m_0x + c_0 + k \cdot w_0$ with $y = m_1x + c_1 + k' \cdot w_1$ and $y = m_0x + c_0 + (k+1) \cdot w_0$ with $y = m_1x + c_1 + (k'+1) \cdot w_1$. All that is left is to find the gradient of the two intersections. Solving the first case, $x_0 = \frac{c_1-c_0-k\cdot w_0-k'\cdot w_1}{m_0-m_1}$ and $y_0 = m_0(\frac{c_1-c_0-k\cdot w_0-k'\cdot w_1}{m_0-m_1}) + c_0 + k \cdot w_0$. The second, $x_1 = \frac{c_1-c_0-(k+1)\cdot w_0-(k'+1)\cdot w_1}{m_0-m_1}$ and $y_1 = m_0(\frac{c_1-c_0-(k+1)\cdot w_0-(k'+1)\cdot w_1}{m_0-m_1}) + c_0 + (k+1) \cdot w_0$. The gradient of the two points can be found, $\frac{y_1-y_0}{x_1-x_0}$, which results in the gradient of the labelling applied to the lattice.

Proposition 8. *The gradients of the new lines formed by the Anti-moiré equation can be predicted using the equation*

$$\frac{w_o \cdot m_1 + w_1 \cdot m_0}{w_0 + w_1}\,.$$

Variables refer to, line gradient, m_i, and the width of the line, w_i.

Proof. The form of the moiré function is such that if the intersection of one area is labelled as (k,k') then the next area that is labelled similarly will be of the form $(k+1,k'+1)$. Encoding this as the intersections of lines $y = m_0x + c_0 + k \cdot w_0$ with $y = m_1x + c_1 + (k'+1) \cdot w_1$ and $y = m_0x + c_0 + (k+1) \cdot w_0$ with $y = m_1x + c_1 + k' \cdot w_1$. All that is left is to find the gradient of the two intersections. Solving the first case, $x_0 = \frac{c_1-c_0-k\cdot w_0-(k'+1)\cdot w_1}{m_0-m_1}$ and $y_0 = m_0(\frac{c_1-c_0-k\cdot w_0-(k'+1)\cdot w_1}{m_0-m_1}) + c_0 + k \cdot w_0$. The second, $x_1 = \frac{c_1-c_0-(k+1)\cdot w_0-k'\cdot w_1}{m_0-m_1}$ and $y_1 = m_0(\frac{c_1-c_0-(k+1)\cdot w_0-k'\cdot w_1}{m_0-m_1}) + c_0 + (k+1) \cdot w_0$. The gradient of the two points can be found, $\frac{y_1-y_0}{x_1-x_0}$, which results in the gradient of the labelling applied to the lattice.

Composition of this form is given as example in Figure 14.13. Here the two lines are shown on the lattice correspond to the gradients of the two differing aggregation functions. The following proposition is now noted.

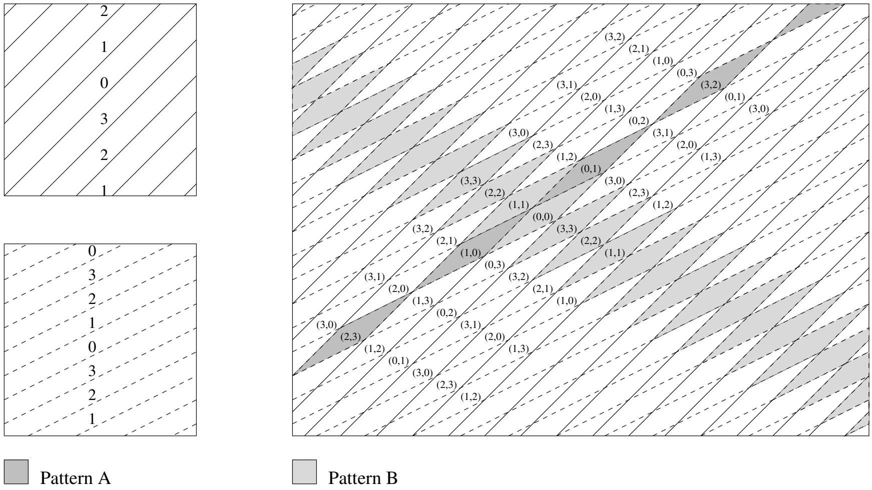


Fig. 14.13 The two forms of aggregation via the above functions. Here pattern A gives the line formed by the anti-moire function and the pattern b gives the line as formed by the moire function.

Proposition 9. *For any two discrete lines on the $\mathbb{Z}^2$ lattice it is possible to vary the observed gradients for resultant from aggregate functions by varying the width of the lines.*

Proof. The proof is an observation of the equations for the gradients of the lines resultant from aggregation. Observing the results of two aggregating functions, $\frac{w_o \cdot m_1 + w_1 \cdot m_0}{w_0 + w_1}$ and $\frac{w_o \cdot m_1 - w_1 \cdot m_0}{w_0 - w_1}$, any alteration to the width of the line which represents results in a direct change in the gradient.

It is now possible to formulate the following statement which shows the increase in expressivity that comes from using the composition of two broadcast sequences.

The number of lines formed by aggregation are not limited by the restrictions that are demonstrated in Proposition. 5 where it is now possible to construct new lines through the varying of widths and indeed new polygons may now be formed with this technique such that they are non-convex.

14.9 Formation of Polygons through Aggregation

Having shown that it is possible to construct different gradients from line sections it is natural to now observe what happens when whole circles are intersected and the relationships that are formed by the aggregation functions that have been introduced. It is noted that a diagram for the formation of polygons in case of the moiré aggregation function is shown in Figure 14.14. Two digital discs, those with squared radii two and 25 such that the discs are ζ^2 and ζ^{25}, are composed gen-

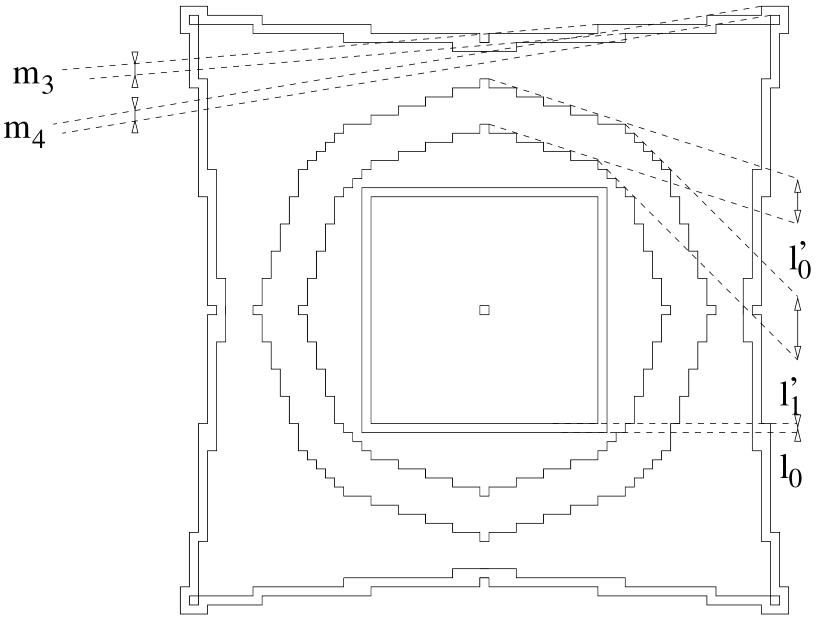


Fig. 14.14 The above schematic depicts the broadcast of two discrete circles. The first, inner, circle (of squared radius two where $\zeta^2 = l_0$ with gradient m_0) and the second discrete circle (of squared radius 25 where $\zeta^{25} = l'_0 l'_1$ with gradients m_1 and m_2 respectively) the outer construction shows the resulting moiré lines here labelled m_3 and m_4. Arrowed lines show line width measurements, the respective w_i, here, $w_0 = 1, w_1 = 5, w_2 = 7, w_3 = \frac{5}{4}$ and $w_4 = \frac{7}{6}$.

erating, from the two convex polygons, and new, previously unreachable, polygon which is *non − convex*.

The following examples given in Figure 14.15 also elucidate the differences between the two aggregation functions showing the non-convex polygon generated by the two digital discs that are represented by, ζ^2 and ζ^9.

It is also possible to gain a better picture of the overall shapes produced by the anti-moire equation by reducing the number of labels, and so the colours, further. This is done by a process of merging certain values or reducing the values over some modulo which in this example, Figure 14.16, is two. The reduction is given by the following aggregation function:

Anti-moiré-Mod2(i,j) =

⊕	0	1	2	3
0	a	b	b	a
1	b	b	a	a
2	b	a	a	b
3	a	a	b	b

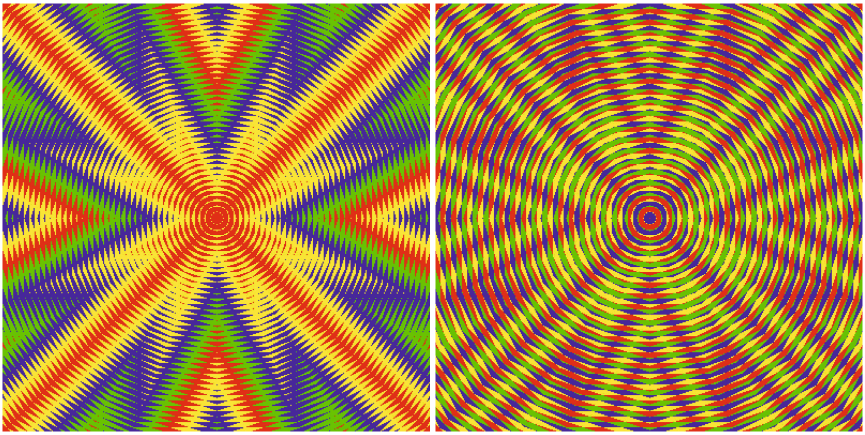


Fig. 14.15 Above gives an example of aggregation of two broadcast sequences, one of the discrete disc ζ^{32} and the other ζ^{36}, from a central point on the diagram and with the two aggregation functions (left) moiré and (right) Anti-moiré.

14.10 Approximating L^p Metrics with Broadcasting Automata

Previously, von Neumann and Moore neighbourhoods have been used to achieve an approximation of the Euclidean metric through some mixing of the neighbourhoods. This has been done in many different settings such as periodic, non-periodic combinations of the two neighbourhoods, regular and non-regular, i.e, hexagonal, triangular grids to which different sequences, i.e applying different definitions of neighbourhood, are applied, as well as a variety of methods for defining just how an approximation of Euclidean distance by neighbourhood sequences should be defined and measured, where most of these techniques discuss notions of digital circularity such as the isoperimetric ratio, perimeter comparisons, etc. In short this has been one of the main studies with regards to neighbourhood sequences. There is ultimately, as has been previously discussed, a large barrier to the extension of this body of work in the approximation of the more general L^p metrics due to the impossibility of constructing any non-convex polygon from the composition of the two convex polygons which represent the Moore and von Neumann neighbourhoods. The astroid is part of the 'family' of L^p metrics of which the Moore neighbourhood, L^∞, the von Neumann neighbourhood, L^1, and the Euclidean metric, L^2 are the most well known. In general an L^p space is defined by the formula $\| x \|_p = (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{\frac{1}{p}}$ such that $\| x \|_p$ is the p-norm.

It is in this section that a new method for the generalisation of metric approximation with broadcasting neighbourhoods shall be given and, using only two broadcasting radii, here given in terms of r^2, and the moiré aggregating function, explore the ability of this model to approximate the astroid and, as such, a new class of metrics outside of the reach of neighbourhood sequences.

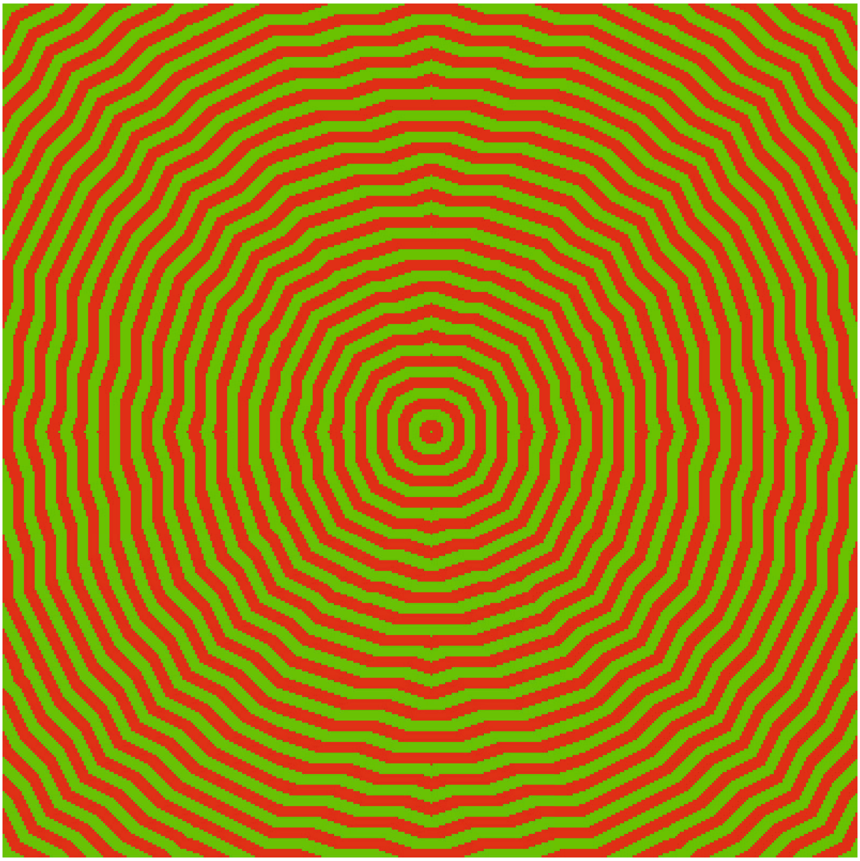


Fig. 14.16 The above figure depicts the anti moire pattern after the merging of two sets of two colours and initially generated by the discrete discs of squared radius 32 and 36 from the central point on the plane.

It has previously been seen that the methodology of combining two broadcasting sequences using an aggregating function can be used to extend the possible resulting polygons. In this section it shall be seen that this can be employed in the solution to a practical problem. The problem, in this case, is the approximation of L^p metric, $L^{2/3}$, which forms an *astroid* [30]. The astroid can be expressed by the equation, $x^{2/3} + y^{2/3} = r^{2/3}$, where x and y are those of the Cartesian plane and, as with the equation for the circle from which this equation is generalised, the r is the 'radius' of the astroid.

A few preliminaries with regards to the astroid which are taken from [30] will be required to understand the methods proposed for comparing the approximation, via broadcasting sequence and aggregation, and the actual astroid. The astroid was discovered in Roemer in 1674 whilst searching for the best form of gear teeth. It is a hypercycloid of four cups and can be described by a point on a circle of radius $\frac{3}{4} \cdot a$

rolling on the inside of a fixed circle of radius a. The resultant shape has a perimeter, L, of $L = 6a$ where a is the radius of the outer fixed circle and is the maximal point reached by the astroid. The area, A, encapsulated is given by $A = \frac{3}{8}\pi a^2$. Finally, the point at which the x and y coordinates are equivalent on the perimeter of the astroid can be expressed as $\frac{r}{2}$. The equation is known to have applications in magnetism where the Stoner-Wohlfarth astroid curve separates regions with two minima of free energy density from those with only one energy minimum and is a geometric representation of the model of the same name [28].

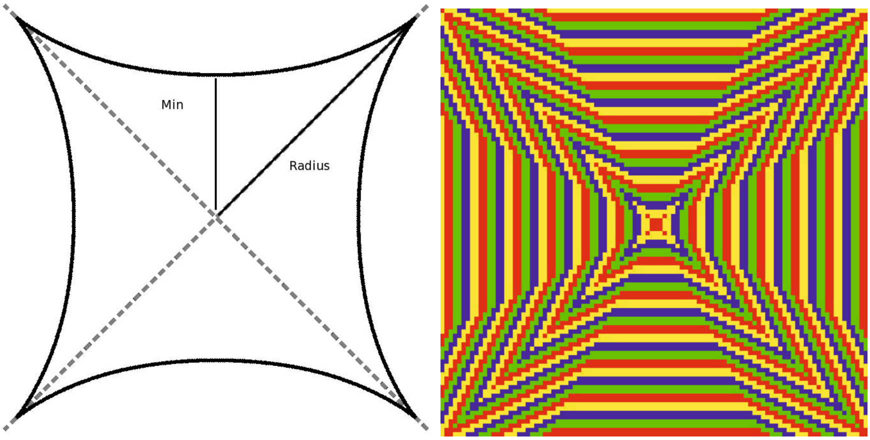


Fig. 14.17 The above figure depicts the astroid, here, rotated by 45° (left) compared to its best, found, approximation with broadcasting sequences (right), an aggregation of the discrete discs of squared radius, 2 and 5.

In [14, 24] a number of methods for how well a particular neighbourhood sequence approximates the euclidean distance are given. The authors of [24] consider any sequence, including those sequences which are non-periodic, of neighbourhoods with which to approximate the euclidean distance. In order to show how well, or poorly, the sequence approximates the euclidean distance it is compared to the euclidean circle through a variety of methods. One such method is through the use of the isoperimetric ratio or the noncompactness ratio. Here, the attempt is to measure $\frac{P^2}{A}$ where, P, is the perimeter, and, A, is the area. This measure is conjectured to be minimal for the circle where it is 4π, however, this does not help when looking at non-convex shapes, such as those produced when approximating the an astroid, due to the measure being optimised for convex and symmetrical shapes.

Other ways of approximating the euclidean circle are suggested such as a perimeter based approximation and area based approximation. With respects to are based approximation there are two techniques used. The first is that of the inscribed circle based approximation. This method attempts to find a sequence that generates the polygon, generated by the neighbourhood sequence, which is closest to the polygon having the given circle as the inscribed circle. The second of these methods is the covering circle based approximation such that polygon must be covered by the circle. More methods are given but rely on properties distinct to the circle and so are not discussed here.

Further, in [14], another measure of circularity is considered using three differing methods. They formulate three approximation problems whereby the aim is to find the neighbourhood sequence that minimises the error in each formulation. The problems may be informally described as the following: problem one requires finding the neighbourhood sequence that best minimises the size of the symmetric difference between some neighbourhood sequence at step k, given as A_k and the Euclidean circle of radius k; the second problem attempts to find the sequence that best minimises the complement of the neighbourhood generated by A_k with it's smallest inscribed circle; the third problem is a discretisation of the second problem where an A_k must be found such that it minimises the complement with the largest discrete disc that can be inscribed within.

It is the second of these problems from [14] that shall be explored as a method for showing a best approximation of the astroid with combined broadcasting sequences. In this case the method is the most simplistic to calculate and also the least dependent on any of the properties that are only present in the circle. As such a more formal representation of the problem can now be posed, here, given as, $Area(H(f_k(A,B))\backslash G'_k \leq Area(H(f_{k'}(A,B'))\backslash G'_{k'})$, where, $f_k(A,B)$ is the kth polygon generated by the aggregation of the broadcasting sequences, A, which here will be the discrete disc, $r^2 = 2$, and the broadcasting sequence, B,B', here a variable which is to be optimised to find the best approximation. There is a simple change to G_k, originally used in [14] to represent the circle of radius k, to convert it to G'_k in that $G'_k = \{q \in \mathbb{R}^2 : L^{\frac{2}{3}}(0,q) \leq k\}$ as simple conversion of the metric from that of the euclidean distance, L^2, to the astroid distance, $L^{\frac{2}{3}}$.

The complexity of calculating these compositions for the purposes of optimisation means that only experimental data shall be given here as a proof of validity of the approximation of the concave metrics, which the astroid represents. As the astroids require a rotation by 45° in order to match the polygon that is generated by the aggregation, $f(A,B)$, as defined before. In matching the point that is at the largest euclidean distance from the origin, which for simplicity, and without loss of generality, is considered the initial point from which all broadcasting occurs. This point is matched to the same point on some polygon on some composition, $f(A,B)$, and the complement of the areas compared. The following table is produced with this method.

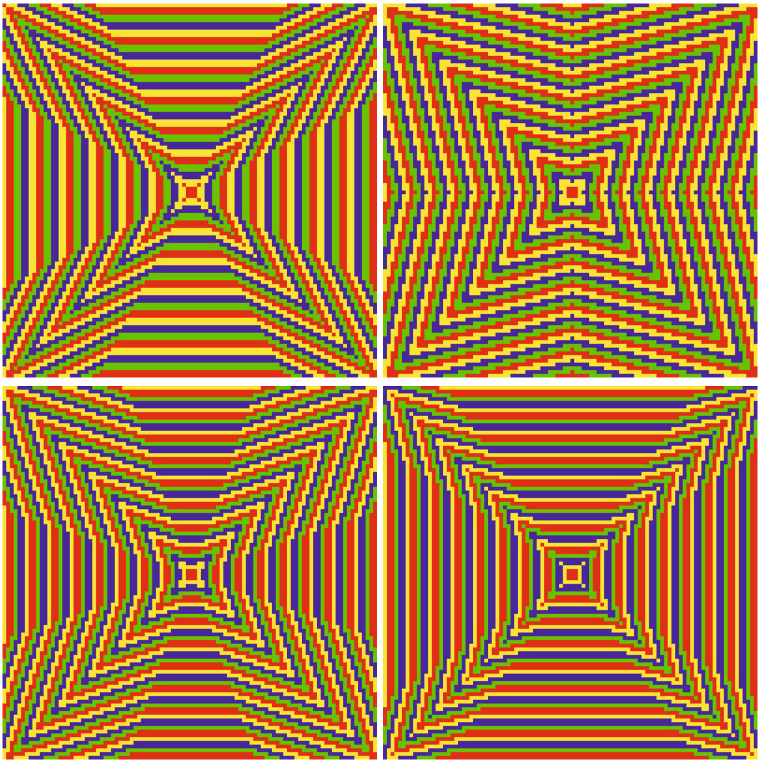
The function of aggregation must also be defined. Here the choice is moiré aggregation without the modulo restriction, although, such a restriction is retained in the images for simplicity. Such a function can now be defined as, $f(A_i,B_j) = |i-j|$ for the ith and jth iteration of the sequence A and B respectively. For the function $f_k(A,B)$ where there exists some A_i and B_j such that $|i-j| = k$.

The min point for the $L^{\frac{2}{3}}$ is calculated by $\frac{r}{2}$ where r is the radius of the astroid, the r in $x^{\frac{2}{3}} + y^{\frac{2}{3}} = r^{\frac{2}{3}}$. The B is the second broadcasting sequence, here, given as

Table 14.1 Illustrating the experiments done with regards to the approximation of the astroid

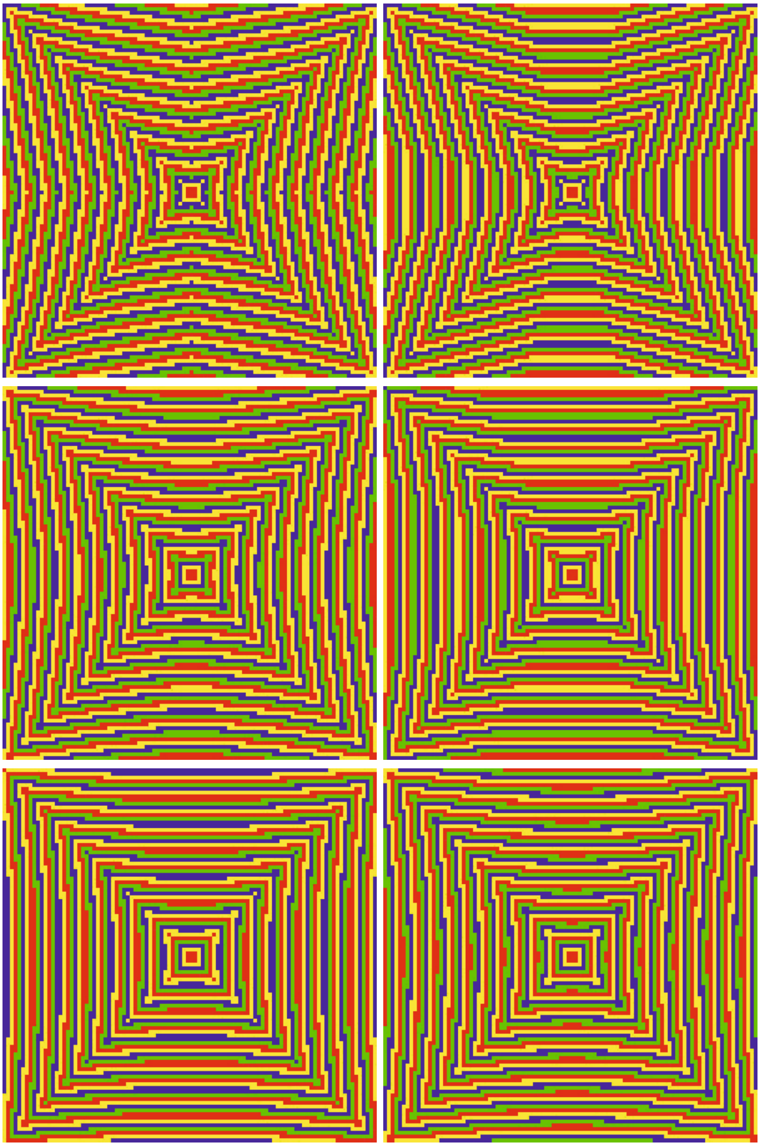
Astroid - $L^{\frac{2}{3}}$			moiré - $f_k(A,B)$					
Radius	Min	Area	Radius	Min	Area	B	k	Complement
17	8.5	340	17	11	461	5	5	121
17	8.5	340	17	13	753	9	8	413
17	8.5	340	17	13	873	10	9	533
18	9	382	18	16	1121	13	11	739
18	9	382	18	14	1033	16	11	651
18	9	382	18	14	1001	17	11	619
17	8.5	340	17	15	1041	37	13	701
17	8.5	340	17	16	1141	45	14	801
17	8.5	340	17	16	1093	61	14	753
17	8.5	340	17	16	1181	82	15	841

Table 14.2 The above diagrams show the patterns generated by the aggregation of the two discs, where one is the disc of squared radius 2 and the other is varied, in these diagrams (from left to right and line by line) there are discs of squared radius, 5, 9, 10 and 13



the r^2 of the disc. Images for each of the resultant, aggregated images are given in Table 14.2 and Table 14.3.

Table 14.3 The above diagrams show the patterns generated by the aggregation of the two discs, where one is the disc of squared radius 2 and the other is varied, in these diagrams (from left to right and line by line) there are discs of squared radius, 16, 17, 37, 45, 61 and 82



Tables 14.1 and 14.4 give a series of comparisons for $Area(H(f_k(A,B)))\backslash G'_k$. From this table it is possible to observe that the best approximation, from those constructed, though there is a trend towards worsening approximations as B increases, that the simplest composition yields the best results. In this case this value for B is

Table 14.4 Illustrating the experiments done with regards to the approximation of the astroid where both A and B are fixed

Astroid - $L^{\frac{2}{3}}$			moiré - $f_k(A,B)$					
Radius	Min	Area	Radius	Min	Area	B	k	Complement
2	1	5	2	1	13	5	1	8
5	2.5	30	5	3	65	5	2	35
8	4	75	8	5	157	5	3	82
11	5.59	143	11	7	289	5	4	146
18	9	382	18	9	461	5	5	79
21	10.5	520	21	11	673	5	6	153
24	12	679	24	13	925	5	7	246
27	13.5	859	27	15	1217	5	8	358
30	15	1060	30	17	1549	5	9	489
33	16.5	1283	33	19	1921	5	10	628

the disc generated by the squared radius r^2 of 5, this disc being constructed by the next smallest squared radius that constructs a new discrete disc. The following table now looks, again, experimentally, at how the approximation changes as k increases. The table notes that the approximation weakens as it increases perhaps indicating a divergence between the astroid and the polygon generated by $f_K(A,B)$.

14.11 Pattern Formations and Periodic Structures in $\mathbb{Z}^2$

It is natural after observing the variety of effects that are the result of the application of an aggregation function to look at functions which are themselves shapes of some form. Here, functions of this form and their resultant patterns, imposed on the grid according to their application, are given. Such functions are only restricted by their symmetry, a result of the unordered nature of the tuples to be aggregated.

The first function, depicted in Figure 14.18 (Left) takes the form of a discrete disc itself, in this case one which is also represented by the discrete disc of squared radius five. The functions here have also been increased in size and the size of the modulo for the labels has been increased. The use of discrete discs of squared radius 8 is important here as it constructs, in some sections of the lattice, a perfect reproduction of the shape given in the aggregating function. The following table describes the aggregating function and the details of the image.

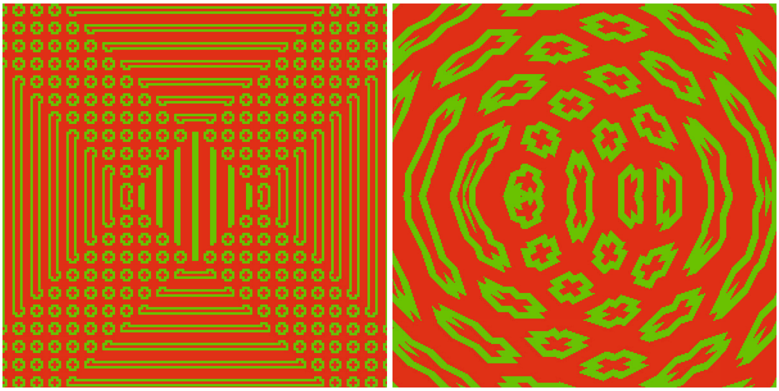


Fig. 14.18 (Left) Patterns generated by the aggregating function representing the discrete disc of squared radius five and the labelling of the lattice given by two broadcasting sequences of squared radius eight. (Right) Patterns generated by the aggregating function representing the discrete disc of squared radius five and the labelling of the lattice given by two broadcasting sequences, one of squared radius 26 and the other of squared radius 36

Array Size: 300
Centre 1: (100,150)
Centre 2: (200,150)
Radius 1: 8
Radius 2: 8
Modular Labelling: (0,1,2,3,4,5,6)
Aggregation Function: Shown right.
Figure 14.18 (Left)

$\oplus$	0	1	2	3	4	5	6
0	a	a	a	a	a	a	a
1	a	a	b	b	b	a	a
2	a	b	b	a	b	b	a
3	a	b	a	a	a	b	a
4	a	b	b	a	b	b	a
5	a	a	b	b	b	a	a
6	a	a	a	a	a	a	a

This reproduction of the aggregating function is not always exact. It is possible to skew and deform the representation of the image described by altering the radii of the circles that are used to form the underlying labelling of the lattice. The following image, Figure 14.18 (right) gives an example of such a deformation.

Array Size: 300
Centre 1: (100,150)
Centre 2: (200,150)
Radius 1: 26
Radius 2: 36
Modular Labelling: (0,1,2,3,4,5,6)
Aggregation Function: Shown right.
Figure 14.18 (Right)

$\oplus$	0	1	2	3	4	5	6
0	a	a	a	a	a	a	a
1	a	a	b	b	b	a	a
2	a	b	b	a	b	b	a
3	a	b	a	a	a	b	a
4	a	b	b	a	b	b	a
5	a	a	b	b	b	a	a
6	a	a	a	a	a	a	a

The two following figures, Figure 14.19, demonstrates changes that occur when manipulating the number of colours, changing the aggregation function and altering the underlying tuples that generate the overall shape of the colourings.

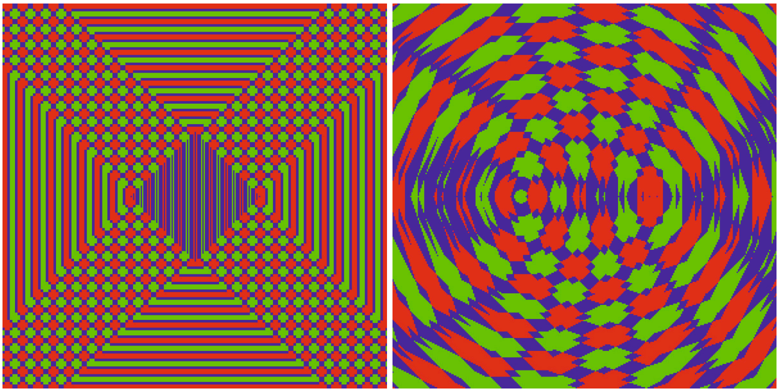


Fig. 14.19 Patterns generated by the aggregating function represented by the table and the labelling of the lattice given by two broadcasting sequences, (right) both of squared radius eight (left) one of squared radius 26 and the other of squared radius 36.

Array Size: 300
Centre 1: (100,150)
Centre 2: (200,150)
Radius 1: 8
Radius 2: 8
Modular Labelling: (0,1,2,3,4,5)
Aggregation Function: Shown right.
Figure 14.19 (left)

⊕	0	1	2	3	4	5
0	b	b	c	c	b	b
1	b	c	a	a	c	b
2	c	a	a	a	a	c
3	c	a	a	a	a	c
4	b	c	a	a	c	b
5	b	b	c	c	b	b

Array Size: 300
Centre 1: (100,150)
Centre 2: (200,150)
Radius 1: 26
Radius 2: 36
Modular Labelling: (0,1,2,3,4,5)
Aggregation Function: Shown right.
Figure 14.19 (rigth)

⊕	0	1	2	3	4	5
0	b	b	c	c	b	b
1	b	c	a	a	c	b
2	c	a	a	a	a	c
3	c	a	a	a	a	c
4	b	c	a	a	c	b
5	b	b	c	c	b	b

Altering the aggregating function is clearly a powerful tool in pattern and polygon formation. The alteration of the discs that are used as the basis of the aggregation also show that any underlying shape generate by an aggregating function can be skewed and otherwise altered whilst retaining the gestalt representation. Methods of manipulating the hew and scale of the shapes that are generated through some aggregating function may be useful to pattern recognition and detection methods that

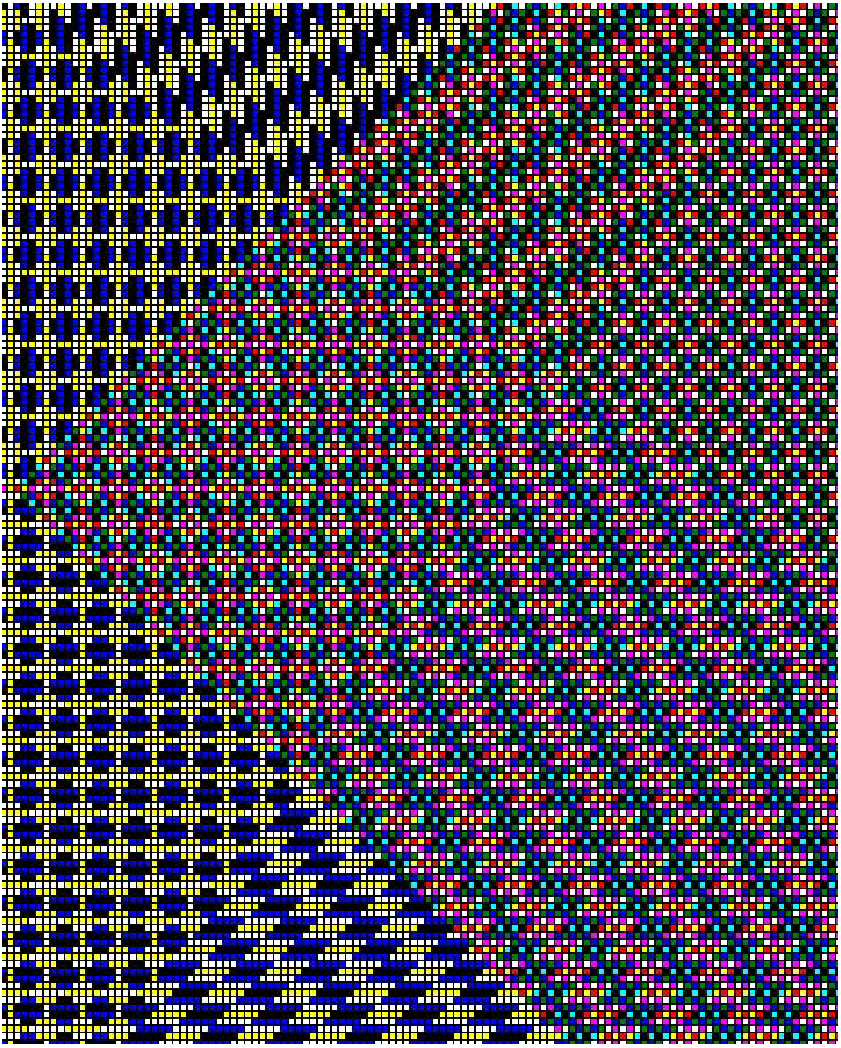


Fig. 14.20 The above figure shows a number of distinct patterns that are here generated by four different transmissions where two, one of squared radius 2 and the other of squared radius 12, are placed at two separate points

are part of the Swarm Robotics cannon among others. Whilst altering the aggregation function is one an interesting concept, for the purpose of pattern formation, there is still a lot of possibility that remains when only considering the standard moiré function. Such as can be seen in the variety of patterns that are formed in Figure 14.20.

References

1. Bhowmick, P., Bhattacharya, B.B.: Number-theoretic interpretation and construction of a digital circle. Discrete Applied Mathematics 156(2), 2381–2399 (2008), http://www.sciencedirect.com/science/article/pii/S0166218X07004817, doi:10.1016/j.dam.2007.10.022, ISSN 0166-218X
2. Bresenham, J.: A linear algorithm for incremental digital display of circular arcs. Commun. ACM 20(2), 100–106 (1977), http://doi.acm.org/10.1145/359423.359432, doi:10.1145/359423.359432, ISSN 0001-0782
3. Chatterji, B.N., Das, P.P., Chakrabarti, P.P.: Generalized distances in digital geometry. Information Sciences 42, 51–67 (1987)
4. Conway, J.H.: Regular Algebra and Finite Machines. Dover Books on Mathematics Series. Dover Publications (2012), http://books.google.co.uk/books?id=1KAXc5TpEV8C, ISBN 9780486485836
5. Duncan, R.: A survey of parallel computer architectures. Computer 23(2), 5–16 (1990), doi:10.1109/2.44900, ISSN 0018-9162
6. Farkas, S., Bajk, J., Nagy, B.: Approximating the Euclidean circle in the square grid using neighbourhood sequences. Pure Math. Appl (PU.M.A.) 17, 309–322 (2006), ISSN 1218-4586
7. Farkas, S., Bajak, J., Nagy, B.: Approximating the Euclidean circle in the square grid using neighbourhood sequences. ArXiv e-prints (June 2010)
8. Feng, T.: A survey of interconnection networks. Computer 14(12), 12–27 (1981), doi:10.1109/C-M.1981.220290, ISSN 0018-9162
9. Feynman, R.P., Leighton, R.B., Sands, M.L.: The Feynman lectures on physics. Addison-Wesley World Student Series, vol. 1. Addison-Wesley Pub. Co. (1963), http://books.google.co.uk/books?id=_ZUfAQAAMAAJ
10. Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Transactions on Electronic Computers, EC-10, 260–268 (1961), doi:10.1109/TEC.1961.5219197, ISSN 0367-9950
11. Gerhardt, M., Schuster, H., Tyson, J.J.: A cellular automaton model of excitable media: Ii. curvature, dispersion, rotating waves and meandering waves. Physica D: Nonlinear Phenomena 46(3), 392–415 (1990), http://www.sciencedirect.com/science/article/pii/016727899090101T, doi:10.1016/0167-2789(90)90101-T, ISSN 0167-2789
12. Hella, L., Järvisalo, M., Kuusisto, A., Laurinharju, J., Lempiäinen, T., Luosto, K., Suomela, J., Virtema, J.: Weak models of distributed computing, with connections to modal logic. CoRR, abs/1205.2051 (2012)
13. Hajdu, A.: Geometry of neighbourhood sequences. Pattern Recognition Letters 24(15), 2597–2606 (2003)
14. Hajdu, A., Hajdu, L.: Approximating the euclidean distance using non-periodic neighbourhood sequences. Discrete Mathematics 283(1-3), 101–111 (2004), http://www.sciencedirect.com/science/article/pii/S0012365X04001116, doi:10.1016/j.disc.2003.12.016, ISSN 0012-365X

15. Kari, J.: Theory of cellular automata: a survey. Theor. Comput. Sci. 334, 3–33 (2005), `http://dl.acm.org/citation.cfm?id=1083031.1083033`, doi:10.1016/j.tcs.2004.11.021, ISSN 0304-3975
16. Lee, G., Chong, N.Y.: A geometric approach to deploying robot swarms. Annals of Mathematics and Artificial Intelligence 52 257–280 (2008), `http://dl.acm.org/citation.cfm?id=1527581.1527606`, doi: 10.1007/s10472-009-9125-x, ISSN 1012-2443
17. Lee, G., Yoon, S.: A mobile sensor network forming concentric circles through local interaction and consensus building. Journal of Robotics and Mechatronics 21, 469–477 (2009), ISSN 1883-8049
18. Linz, P.: An Introduction to Formal Languages and Automata. Theory of Computation Series. Jones and Bartlett (2001), `http://books.google.co.uk/books?id=Cgooanwdo9AC`, ISBN 9780763714222
19. Martin, R., Nickson, T., Potapov, I.: Geometric computations by broadcasting automata on the integer grid. In: Calude, C.S., Kari, J., Petre, I., Rozenberg, G. (eds.) UC 2011. LNCS, vol. 6714, pp. 138–151. Springer, Heidelberg (2011)
20. Martin, R., Nickson, T., Potapov, I.: Geometric computations by broadcasting automata. Natural Computing, 1–13 (2012), `http://dx.doi.org/10.1007/s11047-012-9330-0`, doi:10.1007/s11047-012-9330-0, ISSN 1567-7818
21. Matoušek, J.: Lectures on Discrete Geometry. Graduate Texts in Mathematics. Springer (2002), `http://books.google.co.uk/books?id=MzFzCZAAk8MC` ISBN 9780387953731
22. Mazoyer, J.: An overview of the firing squad synchronization problem. In: Choffrut, C. (ed.) Automata Networks. LNCS, vol. 316, pp. 82–94. Springer, Heidelberg (1988), `http://dx.doi.org/10.1007/3-540-19444-4_16`, doi:10.1007/3-540-19444-4_16, ISBN 978-3-540-19444-6
23. Nagy, B.: Metric and non-metric distances on zn by generalized neighbourhood sequences. In: Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, ISPA 2005, pp. 215–220 (September 2005), doi:10.1109/ISPA.2005.195412
24. Nagy, B., Strand, R.: Approximating euclidean distance using distances based on neighbourhood sequences in non-standard three-dimensional grids. In: Reulke, R., Eckardt, U., Flach, B., Knauer, U., Polthier, K. (eds.) IWCIA 2006. LNCS, vol. 4040, pp. 89–100. Springer, Heidelberg (2006)
25. Schneider, R.: Convex Bodies: The Brunn-Minkowski Theory. In: Encyclopedia of Mathematics and Its Applications. Cambridge University Press (1993), `http://books.google.co.uk/books?id=2QhT8UCKx2kC`
26. Sloane, N.J.A.: The On-Line Encyclopedia of Integer Sequences (1995), `http://oeis.org/A001481A001481`; Numbers that are the sum of 2 nonnegative squares
27. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM Journal on Computing 28, 1347–1363 (1999)

28. Thiaville, A.: Extensions of the geometric solution of the two dimensional coherent magnetization rotation model. Journal of Magnetism and Magnetic Materials 182(1-2), 5–18 (1998), http://www.sciencedirect.com/science/article/pii/S0304885397010147, doi:10.1016/S0304-8853(97)01014-7, ISSN 0304-8853
29. Wolfram, S.: Universality and complexity in cellular automata. Physica D: Nonlinear Phenomena 10(1-2), 1–35 (1984), http://www.sciencedirect.com/science/article/pii/0167278984902458, doi:10.1016/0167-2789(84)90245-8, ISSN 0167-2789
30. Yates, R.C.: Curves and their properties. Classics in mathematics education. National Council of Teachers of Mathematics (1974), http://books.google.co.uk/books?id=UPs-AAAAIAAJ

Chapter 15
Real-Time Prime Generators Implemented on Small-State Cellular Automata

Hiroshi Umeo, Kunio Miyamoto, and Yasuyuki Abe

Abstract. For a long time there was little use of prime numbers in practical applications. But nowadays, it has been known that large-scale prime numbers play a very important role in encryption in computer security networks. In this paper, we explore the prime generation problem on cellular automata consisting of infinitely many cells each with finite state memory and present two implementations of real-time prime generators on cellular automata having smallest number of internal states, known at present. It is shown that there exists a real-time prime generator on a 1-bit inter-cell communication cellular automaton with 25-states, which is an improvement over a 34-state implementation given in Umeo and Kamikawa [2003]. In addition, we show that an infinite prime sequence can be generated in real-time by an eight-state cellular automaton with constant-bit communications. Both the algorithms presented are based on the classical sieve of Eratosthenes, and our eight-state implementation is an improvement over a nine-state prime generator developed by Korec [1998]. Those two implementations on cellular automata with different communication models are the smallest realizations in the number of states, at present.

15.1 Introduction

Cellular automata (CA) are considered to be a useful model of complex systems in which an infinite one-dimensional array of finite state machines (cells) updates itself in a synchronous manner according to a uniform local rule. In the present paper, we study the prime sequence generation problem on cellular automata. In recent years,

Hiroshi Umeo · Yasuyuki Abe
Univ. of Osaka Electro-Communication,
Neyagawa-shi, Hatsu-cho, 18-8, Osaka, 572-8530, Japan
e-mail: umeo@cyt.osakac.ac.jp

Kunio Miyamoto
Japan Advanced Institute of Science and Technology,
1-1, Asahidai, Nomi, Ishikawa, 923-1292, Japan

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_15

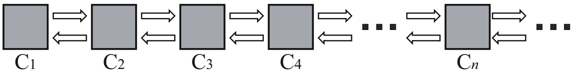


Fig. 15.1 One-dimensional cellular automaton connected with 1-bit inter-cell communication links

it has been known that large-scale prime numbers play a very important role in encryption in computer security networks.

First, we introduce the sequence generation problem on the cellular automaton with 1-bit inter-cell communication, where each cell can communicate with its nearest neighbor cells by sending and receiving a 1-bit information, and present a 25-state real-time prime generator on the model. Then, we consider at each step the same problem on an extended communication model, where each cell can send and receive constant-bits more than one in local communications. We present an eight-state real-time prime generation algorithm on the CA. Both algorithms are based on the sieve of Eratosthenes. Due to the space available we only give an outline of our two constructions.

15.2 Prime Generator on One-Bit Communication Cellular Automata

15.2.1 One-Bit Communication Cellular Automata

A one-dimensional 1-bit inter-cell communication cellular automaton consists of an infinite array of identical finite state automata, each located at a positive integer point. Each automaton is referred to as a cell. The cell at point i is denoted by C_i where $i \geq 1$. Each C_i, except for C_1, is connected with its left and right neighbor cells via a left and right one-way communication link, where those communication links are indicated by right- and left-going arrows, respectively, as shown in Fig. 15.1. Each one-way communication link can transmit only one bit at each step in each direction.

A cellular automaton with 1-bit inter-cell communication (abbreviated as $\mathrm{CA}_{1-\mathrm{bit}}$) consists of an infinite array of finite state automaton $A = (Q, \delta)$, where

(1) Q is a finite set of internal states.
(2) δ is a function that defines the next state of any cell and its binary outputs to its left and right neighbor cells such that δ: $Q \times \{0,1\} \times \{0,1\} \to Q \times \{0,1\} \times \{0,1\}$, where $\delta(p,x,y) = (q,x',y')$, p, $q \in Q$, $x,x',y,y' \in \{0,1\}$, has the following meaning: We assume that, at step t, the cell C_i is in state p and receives binary inputs x and y from its left and right communication links, respectively. Then, at the next step t+1, C_i takes a state q and outputs x' and y' to its left and right communication links, respectively. Note that the binary inputs to

C_i at step t are also outputs of C_{i-1} and C_{i+1} at step t. A quiescent state $q \in Q$ has a property such that $\delta(q,0,0) = (q,0,0)$.

Thus, the $CA_{1-\mathrm{bit}}$ is a special subclass of *normal* (i.e., *conventional*) cellular automata. Let N be any normal cellular automaton with a set of states Q and a transition function $\delta : Q^3 \to Q$. The state of each cell on N depends on the previous state of the cell and states of its nearest neighbor cells. This means that the amount of information exchanged per step between neighboring cells is $O(1)$ bits. Each state in Q can be encoded with a binary sequence of length $\lceil \log_2 |Q| \rceil$ and then transmitted by sending the binary sequences sequentially bit-by-bit in each direction via each one-way communication link. The sequences are then received bit-by-bit and decoded into their corresponding states in Q. Thus, the $CA_{1-\mathrm{bit}}$ can simulate one step of N in $\lceil \log_2 |Q| \rceil$ steps. This observation gives the following computational relation between the normal CA and the $CA_{1-\mathrm{bit}}$.

Themrem 1 Mazoyer [1996], Umeo and Kamikawa [2002] Let N be any *normal* cellular automaton operating in $T(n)$ steps with internal state set Q. Then, there exists a $CA_{1-\mathrm{bit}}$ that can simulate N in $kT(n)$ steps, where k is a positive integer such that $k = \lceil \log_2 |Q| \rceil$.

15.2.2 Sequence Generation Problem

We now define the sequence generation problem on the $CA_{1-\mathrm{bit}}$. Let M be a $CA_{1-\mathrm{bit}}$, and let $\{t_n | n = 1,2,3,...\}$ be an infinite monotonically increasing positive integer sequence defined on natural numbers, such that $t_n \geq n$ for any $n \geq 1$. We then have a semi-infinite array of cells, as shown in Fig. 15.1, and all cells, except for C_1, are in the quiescent state at time $t = 0$. In order to define the sequence generation problem, we assume that the state set of the communication cell C_1 is $\{0,1\}$. The communication cell C_1 assumes a special state "0" (zero) in Q and outputs 1 to its right communication link at time $t = 0$ for initiation of the sequence generator. We say that M generates a sequence $\{t_n | n = 1,2,3,...\}$ in k *linear-time* if and only if the leftmost end cell of M falls into a special state "1" (one) in Q at time $t = kt_n$, where k is a positive integer. We call M a *real-time* generator when $k = 1$. In designing algorithms for $CA_{1-\mathrm{bit}}$, for ease of understanding and description, we often use *signals* or *waves*, instead of transition tables. A signal (wave) is an information flow that is described as a straight line in the space-time diagram. Note that any signal cannot propagate at speed more than one cell per one step. Arisawa [1971], Fischer [1965], Korec [1997, 1998] and Mazoyer and Terrier [1999] have considered the sequence generation problem on the cellular automata model. Umeo and Kamikawa [2002] showed that infinite non-regular sequences such as $\{2^n | n = 1,2,3,..\}$, $\{n^2 | n = 1,2,3,..\}$ and Fibonacci sequences can be generated in real-time and the prime sequence in twice real-time by $CA_{1-\mathrm{bit}}$. Umeo and Kamikawa [2003] showed that the prime sequence can be generated in real-time by a $CA_{1-\mathrm{bit}}$ having 34 internal states and 107 transition rules.

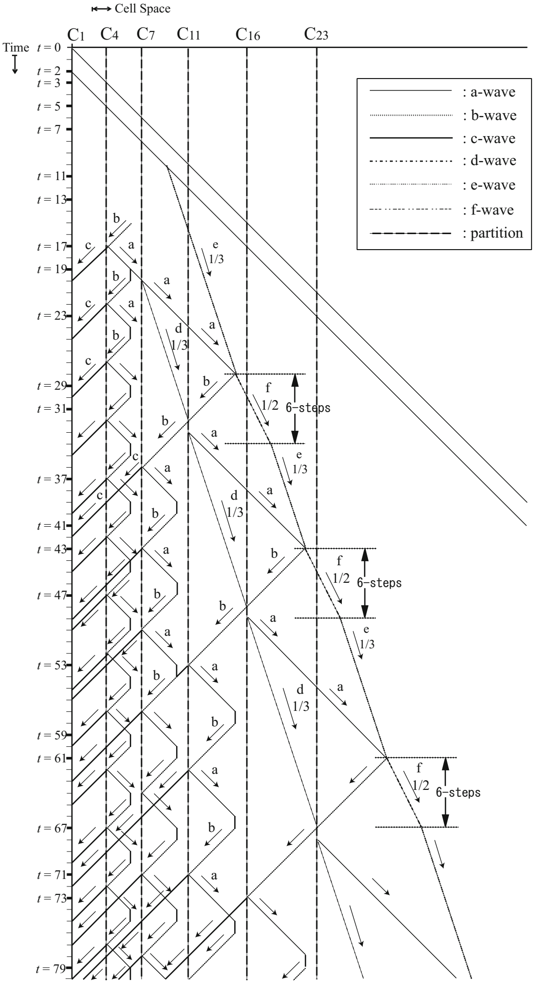


Fig. 15.2 Space-time diagram for Korec's 9-state real-time prime generator on constant-bit communication model

15.2.3 Korec's Prime Generator

Our prime generation algorithm is based on the well-known sieve of Eratosthenes. Imagine a list of all integers greater than 2. The first member, 2, becomes a prime and every second member of the list is crossed out. Then, the next member of the remainder of the list, 3, is a prime and every third member is crossed out. In Eratosthenes' sieve, the procedure continues with 5, 7, 11, and so on. In our procedure, given below, for any odd integer $k \geq 3$, every $2k$-th member of the list beginning with k^2 will be crossed out, since the k-th members less than k^2 (that is, $\{i \cdot k | 2 \leq i \leq k-1\}$) and $2k$-th members beginning with $k^2 + k$ (that is, it is an even number such that $\{(k+2i-1) \cdot k | i = 1,2,3,...\}$) should have been crossed out in the previous stages. Thus, every k-th member beginning with k^2 is successfully crossed out in our procedure. Those integers never being crossed out are the primes.

Figure 15.2 is a space-time diagram that shows a real-time detection of every multiples of odds. Each cross-out operation is performed by C_1. We assume that the cellular space is initially divided by the partitions such that a special mark "*w*" is printed on cell $\mathrm{C}_{(i^2+3i+4)/2}$, for any positive integer $i \geq 2$. Those partitions will be used to generate reciprocating signals for the detection of every multiples of odds, for example, three, five, and seven, ..., . See Fig. 15.2. We denote the sub-cellular space sandwiched by C_k and C_ℓ as S_i, where $k = (i^2+3i+4)/2$, $\ell = (i^2+7i+14)/2$, for any $i \geq 2$, and call it the i-th partition. Note that S_i contains $(2i+3)$ cells, excluding both ends. How to set up the partitions in terms of 1-bit communication is omitted. One can observe that: Let M be a $\mathrm{CA}_{1-\mathrm{bit}}$. We assume that the initial array of M is partitioned into finitely many blocks such that a special symbol "*w*" is marked on cells $\mathrm{C}_{(i^2+3i+4)/2}$, for any positive integer $i \geq 2$. The array M given above can generate the i-th prime at time $t = i$.

15.2.4 A 25-State Real-Time Prime Generator on $\mathrm{CA}_{1-\mathrm{bit}}$

Based on the space-time diagram shown in Fig. 15.2, we construct a 25-state $\mathrm{CA}_{1-\mathrm{bit}}$ that can generate primes in real-time. Figure 15.3 shows its transition table consisting of 25 states and 86 local rules. In Figure 15.4 some snapshots on 34 cells for real-time generation of primes are given. Small right- and left-facing black triangles, ► and ◄, in the figure, indicate a 1-bit signal transfer in the right or left direction between neighbor cells. The symbol in each cell shows its internal state.

Theorem 2. There exists a $\mathrm{CA}_{1-\mathrm{bit}}$ with 25-states and 86-rules that can generate prime sequence in real-time.

15.3 Prime Generator on Constant-Bit Communication Cellular Automata

15.3.1 Cellular Automaton with Constant-Bit Communication

A cellular automaton with constant-bit communication is a usual CA which consists of an infinite array of finite state automaton $A = (Q, \delta)$, where

(1) Q is a finite set of internal states.
(2) δ is a function that defines the next state of any cell such that δ: $Q^3 \rightarrow Q$ where $\delta(p,q,r) = q'$, $p,q,r,p' \in Q$, has the following meaning: We assume that, at step t, the cell C_{i-1}, C_i, and C_{i+1} are in state p,q,r, respectively. Then, at the next step t+1, C_i takes a state q'.

Current state	Input from right link
Input from left link	(next state, left output, right output)

1

0	R = 0	R = 1
L = 0	(0,0,1)	(1,0,1)
L = 1	(1,0,1)	(1,1,1)

2

1	R = 0	R = 1
L = 0	--	(1,0,1)
L = 1	(E3,0,1)	(E3,0,1)

3

.	R = 0	R = 1
L = 0	(.,0,0)	(F4,1,0)
L = 1	(H1,1,1)	--

4

F3	R = 0	R = 1
L = 0	(-,0,0)	(C4,1,0)
L = 1	--	(F4,1,1)

5

F5	R = 0	R = 1
L = 0	(C3,0,1)	(C2,1,0)
L = 1	--	(E2,1,0)

6

B2	R = 0	R = 1
L = 0	--	(H1,0,1)
L = 1	(B1,1,0)	(B1,0,0)

7

B1	R = 0	R = 1
L = 0	--	(E2,1,1)
L = 1	(B2,0,0)	(C3,1,0)

8

E1	R = 0	R = 1
L = 0	(E2,0,1)	(E2,0,0)
L = 1	--	(E4,0,1)

9

E2	R = 0	R = 1
L = 0	(-,1,0)	(E3,0,1)
L = 1	(-,1,1)	(E2,0,1)

10

E3	R = 0	R = 1
L = 0	(E1,0,1)	(-,0,0)
L = 1	--	--

11

E4	R = 0	R = 1
L = 0	(-,0,0)	(-,0,0)
L = 1	(E4,1,1)	(E4,0,1)

12

-	R = 0	R = 1
L = 0	(-,0,0)	(-,1,0)
L = 1	(^,0,1)	(E1,0,0)

13

F1	R = 0	R = 1
L = 0	(F2,0,0)	(F4,1,0)
L = 1	(F2,1,0)	--

14

F2	R = 0	R = 1
L = 0	(F3,0,1)	(F3,0,0)
L = 1	(D3,1,1)	(F5,0,1)

15

F4	R = 0	R = 1
L = 0	(D1,0,1)	(F5,1,1)
L = 1	(-,1,1)	--

16

^	R = 0	R = 1
L = 0	(^,0,0)	(-,1,0)
L = 1	(F1,0,0)	--

17

H1	R = 0	R = 1
L = 0	(F2,1,1)	(C1,1,0)
L = 1	(B2,1,0)	(E2,1,1)

18

C3	R = 0	R = 1
L = 0	(C4,0,1)	(C4,0,1)
L = 1	(E3,0,1)	(C2,1,1)

19

C4	R = 0	R = 1
L = 0	(C3,0,1)	(C3,0,1)
L = 1	(B2,0,1)	(D3,0,0)

20

C1	R = 0	R = 1
L = 0	(C1,1,1)	(C1,0,1)
L = 1	(C1,0,1)	(C2,1,1)

21

C2	R = 0	R = 1
L = 0	(C2,1,1)	(C2,0,1)
L = 1	(D4,1,0)	(D3,0,1)

22

D1	R = 0	R = 1
L = 0	(.,0,1)	(D2,0,1)
L = 1	(D2,1,0)	(D2,1,1)

23

D2	R = 0	R = 1
L = 0	--	--
L = 1	(D1,1,0)	(D1,0,0)

24

D3	R = 0	R = 1
L = 0	(D4,1,1)	(D4,1,1)
L = 1	(D4,1,0)	(D4,1,1)

25

D4	R = 0	R = 1
L = 0	(D4,1,0)	(C1,1,1)
L = 1	(D3,1,0)	(D1,1,0)

Fig. 15.3 Transition rule set for the 25-state real-time prime generator on $\mathrm{CA}_{1-\mathrm{bit}}$

Figure 15.5 shows a semi-infinite one-dimensional cellular array (CA) consisting of cells, denoted by C_1, C_2, ..., C_i, ..., $i \geq 1$. Each cell is an identical (except the border cells) finite-state automaton. The array operates in lock-step mode in such a way that the next state of each cell (except border cells) is determined by both its own present state and the present states of its left and right neighbors. All cells, except the left end cell, are initially in the quiescent state at time $t = 0$ with the property that the next state of a quiescent cell with quiescent neighbors is the quiescent state again. At time $t = 0$, the left end cell C_1 is in a special state acting as an initiation signal for the array. Let s_i^t be state of the cell C_i at time t, where $t \geq 0, i \geq 1$.

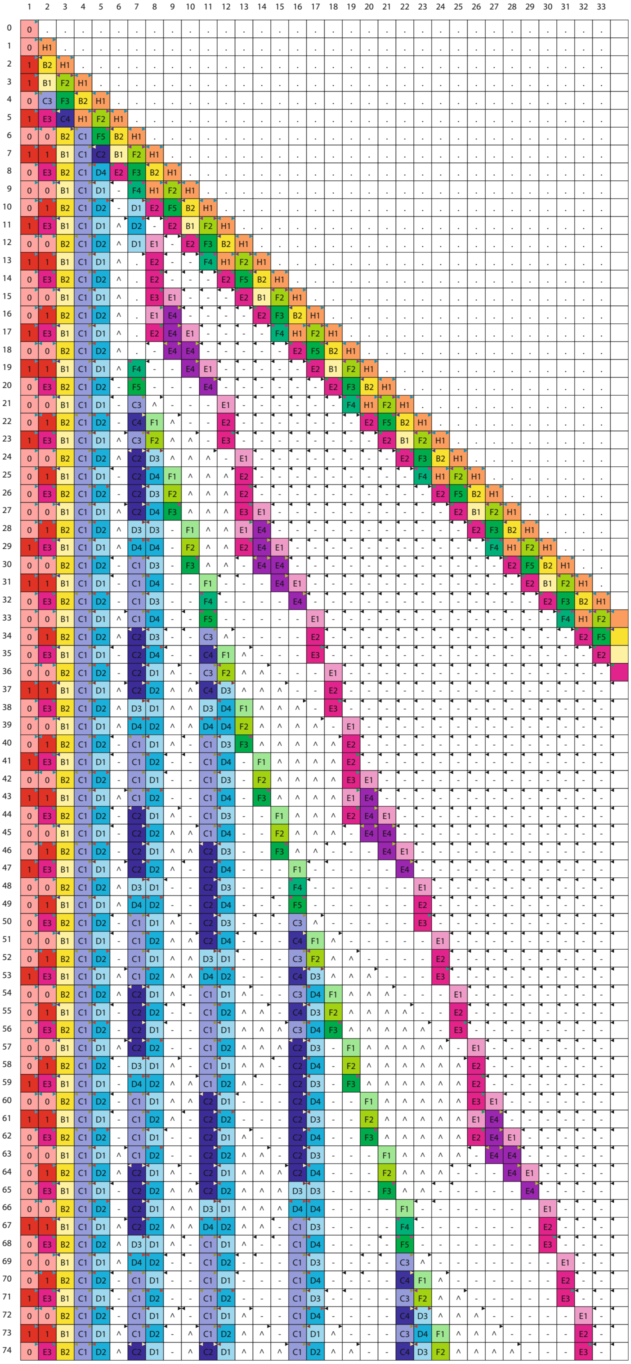


Fig. 15.4 Snapshots of configurations for real-time prime generation on the 25-state $\mathrm{CA}_{1-\mathrm{bit}}$

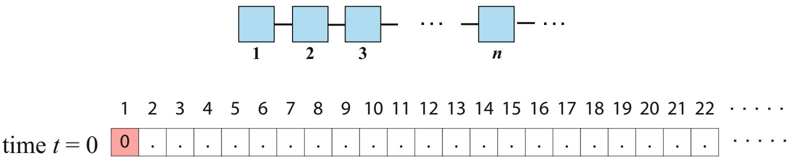


Fig. 15.5 A one-dimensional cellular automaton with constant-bit communication (above) and its initial configuration at time $t = 0$ (below)

15.3.2 Eight-State Real-Time Prime Generator on CA with O(1)-Bit Communication

In this section, we present an eight-state real-time prime generation algorithm on CA. The algorithm is implemented on an eight-state CA using 305 transition rules. Our prime generation algorithm is also based on the sieve of Eratosthenes employed in the previous section. Its cross out technique is different from the one mentioned above. Imagine a list of all integers greater than 2. The first member, 2, becomes a prime and every second member of the list is crossed out. Then, the next member of the remainder of the list, 3, is a prime and every third member is crossed out. In Eratosthenes' sieve, the procedure continues with 5, 7, 11, and so on. In our procedure, given below, for any odd integer $k \geq 3$, every $2k$-th member of the list beginning with k^2 will be crossed out, since the k-th members less than k^2 (that is, $\{i \cdot k | 2 \leq i \leq k-1\}$) and $2k$-th members beginning with $k^2 + k$ (that is, it is an even number such that $\{(k+2i-1) \cdot k | i = 1,2,3,...\}$) should have been crossed out in the previous stages. Thus, every k-th member beginning with k^2 is successfully crossed out in our procedure. Those integers never being crossed out are the primes.

Figure 15.6 is a space-time diagram that shows a real-time detection of odd multiples of odds.

We now outline the algorithm. Each cross-out operation is performed by C_1. We assume that the cellular space is initially divided by the partitions such that a special mark "w" is printed on cell C_{i^2}, for any positive integer $i \geq 1$. Those partitions will be used to generate reciprocating signals for the detection of odd multiples of, for example, three, five, and seven (See Fig. 15.6). We denote a sub-cellular space sandwiched by C_k and C_ℓ as S_i, where $k = i^2$, $\ell = (i+1)^2$ for any $i \geq 1$, and call it the i-th partition. Note that S_i contains $(2i+2)$ cells, including both ends.

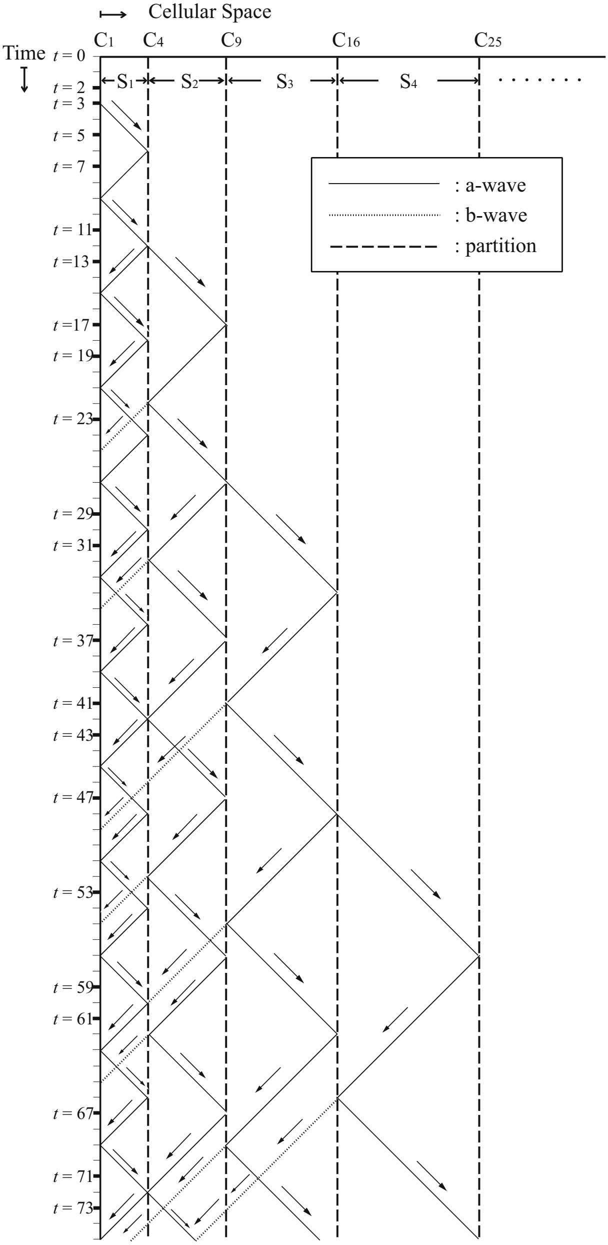


Fig. 15.6 Space-time diagram for real-time detection of odd multiples of odds

.		Right State								
		.	0	1	V	R	L	r	/	*
Left State	.	.	.	/	.	.	L	/	/	.
	0	/	0			/	.			
	1	.	.			.	L	/	/	
	V	.	1			.	L	/	/	
	R	R	R	r	R	R	R		r	
	L	.	.	/	.	0			/	
	r					L			r	
	/	.		/	.	.	L		/	
	*									

0		Right State								
		.	0	1	V	R	L	r	/	*
Left State	.	.	V	0	R	0	r	.	R	
	0	0				1			R	
	1	R						1	0	
	V	1							r	
	R	0		/					V	
	L	V	R	V					R	
	r	0							/	
	/	0	L							
	*	0	0	0	1	1	0	0	1	

1		Right State								
		.	0	1	V	R	L	r	/	*
Left State	.	V				V	1	1	1	
	0	R		R	V	L	L	L	0	
	1		.	.	/	L		L		
	V	1	1	1	1	1	1	.	1	
	R	V				V	1		1	
	L	V				V	1	1	1	
	r	V				V	1		1	
	/	V		.		V	1		1	
	*		1	0	0	0	0			

V		Right State								
		.	0	1	V	R	L	r	/	*
Left State	.	V	0	V	V	V	1		1	
	0		L				L			
	1	R	1			L			1	
	V		V					1		
	R	V		0			1		1	
	L	V					1		1	
	r	V					1		1	
	/	V					1	1	1	
	*									

R		Right State								
		.	0	1	V	R	L	r	/	*
Left State	.	.	.	L	L	.	.	.	/	.
	0	R	V			V	1		1	
	1	.	.	L	L	0		.	/	
	V	.	.			L				
	R	R	r			r			V	R
	L		L	L	L	/				
	r		r	L	L	.				
	/	.		L	L	r			/	
	*									

L		Right State								
		.	0	1	V	R	L	r	/	*
Left State	.	.	.	/	.	L	.		/	
	0	R				R	r	r		
	1	R	R	r	R	R		r		
	V	R	R	r	R		R		r	
	R		L	r	R			V		
	L		V	r	R	R				
	r			r	R					
	/	.	.		.		.			
	*									

r		Right State								
		.	0	1	V	R	L	r	/	*
Left State	.		.	L					/	
	0		R			V	1	1		
	1	.	0	L	L	R			/	
	V		V					R		
	R	0	R							
	L		V	L	L					
	r		V	L	L	r				
	/			L					/	
	*									

/		Right State								
		.	0	1	V	R	L	r	/	*
Left State	.	.	L	/	.	.	L	/	/	
	0	0				V				
	1	.				.	L		/	
	V	.	1			.	L	/	/	
	R	R	0	r	R	r			r	
	L	.		/	.				/	
	r	R			R					
	/	.		/	.	.	L		/	
	*									

Fig. 15.7 Transition table for the eight-state real-time prime generator

Based on the space-time diagram shown in Fig. 15.6, we construct an 8-state CA that can generate primes in real-time. Figure 15.7 shows its transition table consisting of 8 states and 305 local rules. In Figure 15.8 some snapshots on 50 cells are given for real-time generation of primes.

Theorem 3. There exists a cellular automaton having 8-states and 305-rules that can generate prime sequence in real-time.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
0	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	1	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	1	R	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	0	/	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
5	1	V	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6	0	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
7	1	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
8	0	0	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
9	0	1	V	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
10	0	V	L	L	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
11	1	L	R	R	/	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
12	0	R	/	V	r	r	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
13	1	1	R	1	R	r	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
14	0	L	L	V	.	0	L	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
15	0	r	R	V	1	r	R	0	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
16	0	V	L	0	.	R	r	0	/	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
17	1	L	R	V	/	.	R	/	0	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
18	0	R	L	1	.	.	/	0	0	/	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
19	1	1	r	V	.	/	L	L	R	0	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
20	0	L	L	V	/	L	.	R	L	0	/	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
21	0	r	R	1	L	.	0	.	L	R	0	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
22	0	V	L	1	R	.	.	.	L	L	0	/	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
23	1	L	r	V	.	R	.	L	.	V	R	0	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
24	0	r	L	V	.	.	R	.	.	V	.	0	/	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
25	0	1	R	V	.	.	.	R	.	V	1	R	0	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
26	0	L	L	V	.	.	.	.	R	V	1	.	0	/	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
27	0	r	R	V	.	.	.	.	L	0	1	.	R	0	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
28	0	V	L	V	.	.	.	L	.	V	R	.	.	0	/	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
29	1	L	R	V	.	.	L	.	.	V	.	R	.	R	0	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
30	0	R	L	V	.	L	.	.	.	V	.	.	R	.	0	/	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
31	1	1	R	V	L	.	.	.	.	V	.	.	.	R	R	0	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
32	0	L	L	1	R	.	.	.	.	V	.	.	.	.	r	0	/	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
33	0	r	r	V	.	R	.	.	.	V	.	.	.	/	.	/	0	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
34	0	1	L	V	.	.	R	.	.	V	.	.	/	.	/	L	0	/	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
35	0	L	R	V	.	.	.	R	.	V	.	/	.	/	L	.	R	0	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
36	0	R	L	V	.	.	.	.	R	V	/	.	/	L	.	0	.	0	/	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
37	1	1	R	V	.	.	.	.	L	1	.	/	L	.	.	.	0	R	0	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	.
38	0	L	L	V	.	.	.	L	/	V	/	L	.	.	.	.	0	V	0	/	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.
39	0	r	R	V	.	.	L	/	.	1	L	.	.	.	.	.	R	L	r	0	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.
40	0	V	L	V	.	L	/	.	/	1	R	.	.	.	.	.	.	V	V	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.	.
41	1	L	R	V	L	/	.	/	/	V	.	R	.	.	.	.	.	V	V	r	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.	.
42	0	R	L	1	r	.	/	/	.	V	.	.	R	.	.	.	.	V	1	V	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.	.
43	1	1	r	1	.	r	/	.	.	V	.	.	.	R	.	.	.	V	1	1	r	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.	.
44	0	L	L	V	/	/	R	.	.	V	.	.	.	.	R	.	.	V	1	L	0	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.	.
45	0	r	R	1	/	.	.	R	.	V	.	.	.	.	.	R	.	V	1	R	R	R	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.	.
46	0	V	L	1	.	.	.	.	R	V	.	.	.	.	.	.	R	V	1	0	r	r	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.	.
47	1	L	r	V	.	.	.	.	L	V	.	.	.	.	.	.	L	0	1	1	1	V	/	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.	.
48	0	r	L	V	.	.	.	L	.	V	.	.	.	.	.	L	.	V	R	.	/	1	1	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.	.
49	0	1	R	V	.	.	L	.	.	V	.	.	.	.	L	.	.	V	.	r	/	.	.	0	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.	.
50	0	L	L	V	.	L	.	.	.	V	.	.	.	L	.	.	.	V	/	/	R	.	.	V	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R	.
51	0	r	R	V	L	.	.	.	.	V	.	.	L	.	.	.	.	1	/	.	.	R	.	0	r	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.	R
52	0	V	L	1	R	.	.	.	.	V	.	L	.	.	.	.	/	1	.	.	.	.	R	.	R	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.	.
53	1	L	r	V	.	R	.	.	.	V	L	.	.	.	.	/	/	V	.	.	.	.	.	R	.	V	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R	.
54	0	r	L	V	.	.	R	.	.	1	R	.	.	.	/	/	.	V	.	.	.	.	.	.	R	0	1	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R	R
55	0	1	R	V	.	.	.	R	/	V	.	R	.	/	/	.	.	V	.	.	.	.	.	.	.	/	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	R
56	0	L	L	V	.	.	.	/	R	V	.	.	r	/	.	.	.	V	.	.	.	.	.	.	/	L	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
57	0	r	R	V	.	.	/	.	L	V	.	/	/	R	.	.	.	V	.	.	.	.	.	/	L	.	R	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
58	0	V	L	V	.	/	.	L	.	V	/	/	.	.	R	.	.	V	.	.	.	.	/	L	.	0	.	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
59	1	L	R	V	/	.	L	.	.	1	/	.	.	.	.	R	.	V	.	.	.	/	L	.	.	.	0	R	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
60	0	R	L	1	.	L	.	.	/	1	.	.	.	.	.	.	R	V	.	.	/	L	.	.	.	.	0	V	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
61	1	1	r	V	L	.	.	/	/	V	.	.	.	.	.	.	L	V	.	/	L	.	.	.	.	.	R	L	r	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
62	0	L	L	1	R	.	/	/	.	V	.	.	.	.	.	L	.	V	/	L	.	.	.	.	.	.	.	V	V	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
63	0	r	r	V	.	r	/	.	.	V	.	.	.	.	L	.	.	1	L	.	.	.	.	.	.	.	.	V	V	r	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
64	0	1	L	V	/	/	R	.	.	V	.	.	.	L	.	.	/	1	R	.	.	.	.	.	.	.	.	V	1	V	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
65	0	L	R	1	/	.	.	R	.	V	.	.	L	.	.	/	/	V	.	R	.	.	.	.	.	.	.	V	1	1	r	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
66	0	R	L	1	.	.	.	.	R	V	.	L	.	.	/	/	.	V	.	.	R	.	.	.	.	.	.	V	1	L	0	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
67	1	1	r	V	.	.	.	.	L	V	L	.	.	/	/	.	.	V	.	.	.	R	.	.	.	.	.	V	1	R	R	R	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
68	0	L	L	V	.	.	.	L	.	1	R	.	/	/	.	.	.	V	.	.	.	.	R	.	.	.	.	V	1	0	r	r	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
69	0	r	R	V	.	.	L	.	/	V	.	r	/	.	.	.	.	V	.	.	.	.	.	R	.	.	.	V	1	1	1	V	/	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
70	0	V	L	V	.	L	.	/	.	V	/	/	R	.	.	.	.	V	.	.	.	.	.	.	R	.	.	V	1	.	/	1	1	0	/	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Fig. 15.8 Snapshots for the eight-state real-time prime generator on 50 cells

15.4 Conclusions

We have studied the prime generation problem on cellular automata and presented two constructions of real-time prime generators on cellular automata having the smallest number of internal states, known at present. It is shown that there exists a real-time prime generator on 1-bit inter-cell communication cellular automaton with 25-states, which is an improvement over a 34-state implementation given in Umeo and Kamikawa [2003].

In addition, we show that the infinite prime sequence can be generated in real-time by an eight-state cellular automaton with constant-bit communications. Our eight-state implementation is an improvement over a nine-state prime generator presented in Korec [1998]. Those two constructions on cellular automata with different communication models are the smallest realizations in the number of states, at present.

References

1. Arisawa, M.: On the generation of integer series by the one-dimensional iterative arrays of finite state machines. Trans. of IECE, 71/8 54-C(8), 759–766 (1971)
2. Dyer, C.R.: One-way bounded cellular automata. Information and Control 44, 261–281 (1980)
3. Fischer, P.C.: Generation of primes by a one-dimensional real-time iterative array. J. of ACM 12(3), 388–394 (1965)
4. Kamikawa, N., Umeo, H.: A study on sequence generation powers of small cellular automata. SICE Journal of Control, Measurement, and System Integration 11(1), 1–8 (2011)
5. Korec, I.: Real-time generation of primes by a one-dimensional cellular automaton with 11 states. In: Privara, I., Ružička, P. (eds.) MFCS 1997. LNCS, vol. 1295, pp. 358–367. Springer, Heidelberg (1997)
6. Korec, I.: Real-time generation of primes by a one-dimensional cellular automaton with 9 states. In: Margenstern, M. (ed.) Proc. of International Colloquim on Universal Machines and Computation, MCU 1998, vol. I, pp. 101–116. IUT Metz (1998)
7. Mazoyer, J.: On optimal solutions to the firing squad synchronization problem. Theoretical Computer Science 168, 367–404 (1996)
8. Mazoyer, J., Terrier, V.: Signals in one-dimensional cellular automata. Theoretical Computer Science 217, 53–80 (1999)
9. Smith, A.R.: Real-time language recognition by one-dimensional cellular automata. J. of Computer and System Sciences 6, 233–253 (1972)
10. Umeo, H.: Problem solving on one-bit-communication cellular automata. In: Hoekstra, A.G., Kroc, J., Sloot, P.M.A. (eds.) Simulating Complex Systems by Cellular Automata, ch. 6, pp. 117–144. Springer, Heidelberg (2010)
11. Umeo, H., Kamikawa, N.: A design of real-time non-regular sequence generation algorithms and their implementations on cellular automata with 1-bit inter-cell communications. Fundamenta Informaticae 52(1-3), 255–273 (2002)
12. Umeo, H., Kamikawa, N.: Real-time generation of primes by a 1-bit-communication cellular automaton. Fundamenta Informaticae 58(3, 4), 421–435 (2003)

Chapter 16
Phyllosilicate Automata

Andrew Adamatzky

Abstract. The chapter is an overview of our finding on a novel class of regular automata networks, the phyllosilicate automata. Phyllosilicate is a sheet of silicate tetrahedra bound by basal oxygens. A phyllosilicate automaton is a regular network of finite state machines, which mimics structure of the phyllosilicate. A node of a binary state phyllosilicate automaton takes states 0 and 1. A node updates its state in discrete time depending on a sum of states of its three (silicon nodes) or six (oxygen nodes) closest neighbours. By extensive sampling of the node state transition rule space we classify rules by main types of patterns generated by them based on the patterns shape (convex and concave hulls, almost circularly growing patterns, octagonal patterns, dendritic growth); and, the patterns interior (disordered, solid, labyrinthine). We also present rules exhibiting travelling localizations attributed to Conway's Game of Life: gliders, oscillators, still lifes, and a glider gun.

16.1 Introduction

Phyllosilicates are parallel sheets of silicate tetrahedra, they are widely present in nature and typically found in clay-related minerals on the Earth surface [15,16,23]. Phyllosilicates are used in the development of nano-materials, nano-wires and patterned surfaces for nano-biological interfaces [18,34,41]. They are also employed, in a form of cation-exchanged sheet silicates, as catalysts in chemical reactions [10], e.g. nickel phyllosilicate catalysts [25,33,40].

We continue lines of enquiry into space-time dynamics of cellular automata on non-orthogonal and aperiodic lattices, including triangular tessellations and Penrose tilings and hyperbolic planes [14,21,28–32,35]. We introduce and study an automaton network — phyllosilicate automata — where connections between finite automata are inspired by simplified structure of silicate sheets, lattices of con-

Andrew Adamatzky
Unconventional Computing Centre, University of the West of England, Bristol, UK
e-mail: andrew.adamatzmy@uwe.ac.uk

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_16

nected tetrahedra, and investigate the dynamics of perturbations on these automata networks. In the chapter we overview results of our studies of phenomenological automata, thus summarising our line of research published in [7–9].

Automata models studied in the paper are abstractions of silicon sheets. The automata models do not aim to compete with mainstream computational chemistry models [38], however they do bring certain benefits. Namely, an array of finite state machines is a fast prototyping tool for an express evaluation of a space-time dynamics of a spatially-extended active nonlinear medium for different local transitions rules, and prototyping of unconventional computing devices based on the nonlinear medium. Compare, for example, three-state cellular automata and excitable chemical media. Their complexity differ by orders. A cellular automaton's behaviour can be expressed in a short table or set of rules while proper computer representation of an excitable medium requires dozens of partial differential equations. However, high degree of descriptional complexity does not always imply high degree of behavioural. All types of travelling waves, including spiral and target waves in excitable mode and wave-fragment observed mode are sufficiently imitated in Greenberg-Hasting automata [22]. The automaton model of Belousov-Zhabotinsky medium is phenomenologically equivalent to more accurate, close to physical and chemical reality, Oregonator model [20]. By analogy, studies of phyllosilicate automata may help us to get an insight into dynamics of localizations, defects and excitations, in the lattices of silicon tetrahedra. Thus, by developing automata models of excitation in tetrahedra sheets we contribute towards future developments of silicate sheet based computing circuits where quanta of information are transferred and processed by stationary and travelling localised excitations [3,6].

16.2 Phillosilicate Automata

Phyllosilicates are sheets of coordinated $(SiO_4)^{4-}$ tetrahedra units. Each tetrahedron shares its three corner basal oxygens of neighbouring tetrahedra [16,26,36] (Fig. 16.1). Vacant oxygens of all tetrahedra point outside the lattice, away of tetrahedra; these apical oxygens are not taken into account in our present simulation. A phyllosilicate automaton $\mathcal{A}$ is a two-dimensional regular network — as shown in Fig. 16.1 — of finite state machines, or automata [7–9]. There are two types of automata in the network: silicon automata s (centre vertex of each tetrahedron) and oxygen automata o (corner vertices of each tetrahedron). A silicon automaton has three neighbours. An oxygen automaton has six neighbours (Fig. 16.1).

The automata take two states, 0 (resting state) and 1 (non-resting state), and update their states s^t and o^t simultaneously and in discrete time t depending on the states of their neighbours. Let $\sigma^t(a)$ be a number of 1-state neighbours of automaton a, $0 \leq \sigma^t(s) \leq 3$ and $0 \leq \sigma^t(o) \leq 6$. Let o and s be instances of oxygen and silicon automata, and $\mathbf{o}$ and $\mathbf{s}$ be binary strings of seven and four symbols each; $\mathbf{a}[i]$ is a symbol at ith position of string $\mathbf{a}$. An automaton a, $a = o$ or s, updates its state by the rule:

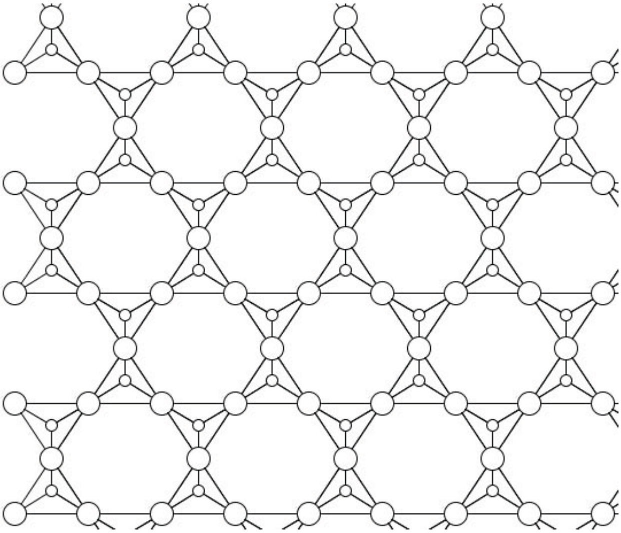


Fig. 16.1 Phillosilicate automata. Automata representing silicon are small discs, oxygen automata are large discs. Unshared oxygen atoms of the tetrahedra are not shown.

$$a^{t+1} = \mathbf{a}_{a^t}[\sigma(a)^t], \tag{16.1}$$

where $a = s$ or o and $\mathbf{a} = \mathbf{s}$ or $\mathbf{o}$. When referring to any particular rule $(\mathbf{s}_0, \mathbf{s}_1, \mathbf{o}_0, \mathbf{o}_1)$, where string $\mathbf{a}_i$ determines next state of automaton being in state i, we will be using decimal representations of strings as $\mathcal{R}(dec(\mathbf{s}_0), dec(\mathbf{s}_1), dec(\mathbf{o}_0), dec(\mathbf{s}_1))$. We assume state 0 is a quiescent state: a node in state 0 will not take state 1 if all its neighbours are 0. Sometimes we will be calling nodes in state 1 resting nodes. To avoid rules which do not produce any patterns at all we do not study rules $\mathbf{a}_0$ where $\mathbf{a}_0[i] = 0$ for all $0 \leq i \leq 6$.

For example, consider automaton $\mathcal{A}$ which nodes update their states as follows. A silicon node in state 0 takes state 1 if one, two or three of its neighbours are in state 1. The node remains in state 0 otherwise. Thus string $\mathbf{s}_0$ is (0111) and its binary representation is 7. A silicon node in state 1 takes state 1 if it has two or three of its neighbours in state, the node takes 0 otherwise: $\mathbf{s}_1 = (0011)$, $dec(\mathbf{s}_1) = 3$. An oxygen node in state 0 takes state 1 if it has two, five or six neighbours in state 1; the node remains in state 0 otherwise: $\mathbf{o}_0 = (0010011)$, $dec(\mathbf{o}_0) = 19$. An oxygen node in state 1 takes state 0 if one, two or six of its neighbours are in state 1; the node remains in state 1 otherwise: $\mathbf{o}_0 = (1001110)$, $dec(\mathbf{o}_0) = 78$. This automaton is represented by the rule $R(7, 3, 19, 78)$.

Automata are tested by random perturbations. In a rectangular array covering 39 nodes, resting nodes are assigned state 1 with probability 0.1. Statistics is collected on a rectangular lattice of 157 × 157 nodes. 100K rules are generated at random. For each rule we randomly perturb the automaton and evolve it for 100 steps, and then evaluate configurations generated visually and using few integral parameters.

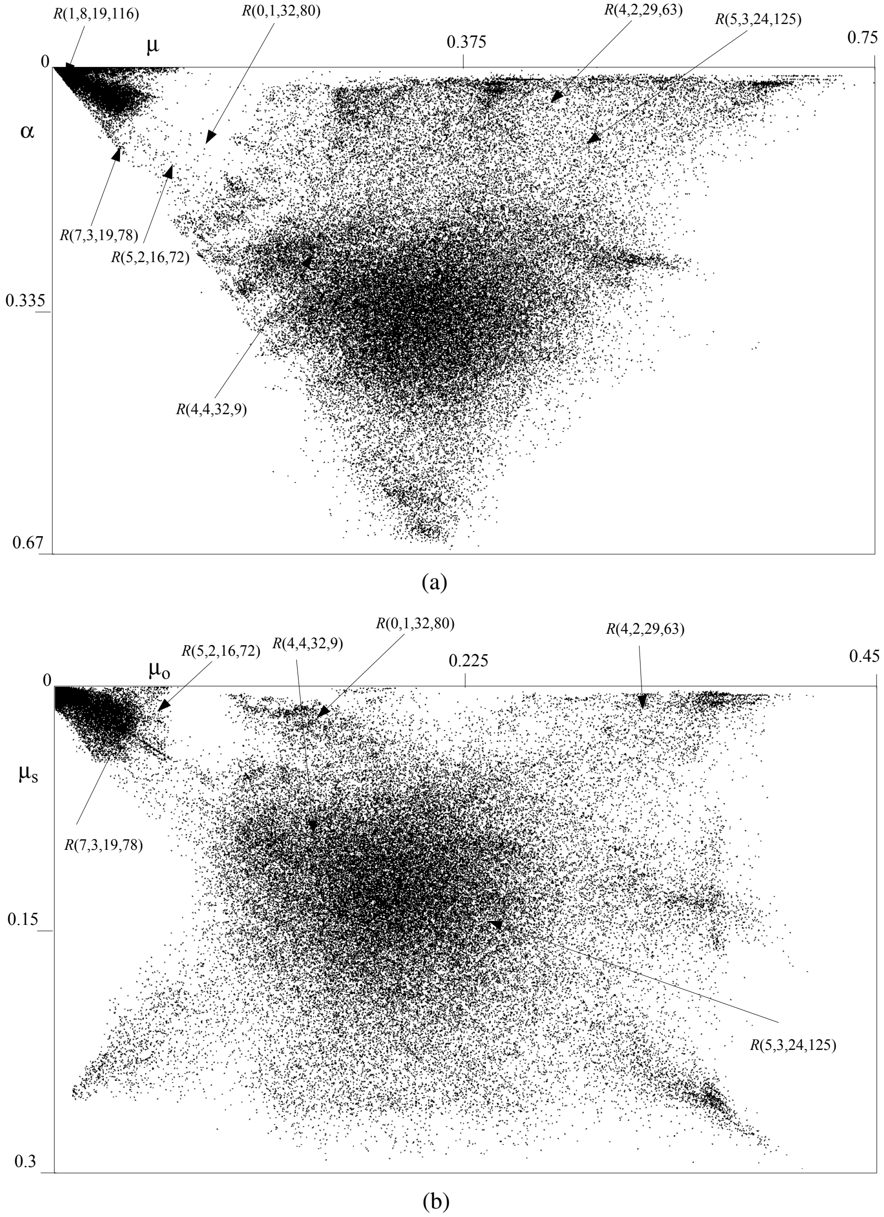


Fig. 16.2 100K randomly chosen rules are mapped onto μ vs α (a) and μ_O vs μ_S (b) map. Locations of some rules illustrated in the paper are shown by arrows.

Density μ_a is the ratio of a number of nodes of type a in state 1 in a disc radius 100 to a total number of nodes covered by the disc; total density $\mu = \mu_s + \mu_o$. Activity α is a ratio of nodes a that $a^{t+1} \neq a^t$, $t = 100$. The rules mapped on parametric spaces are shown in Fig. 16.2.

For each class $\mathcal{C}$ of rules we calculate a likelihood structure $L(\mathcal{C}) = P(\mathbf{p}^s_0, \mathbf{p}^s_1, \mathbf{p}^o_0, \mathbf{p}^o_1)$ as follows. Given a sample of m strings, $m = 200$, for $0 \leq \mathbf{p}^f_z[i] \leq 1$ is calculated as a number of times ’1’ appears in position i of strings $\mathbf{f}_z$ ($\mathbf{f} = \mathbf{o}$ or $\mathbf{s}$ and $z = 0$ or 1) divided by m. The higher is value $0 \leq \mathbf{p}^f_z[i] \leq 1$ the more likely randomly chosen function of a given class will have 1 in position i of string $\mathbf{f}_z$.

16.3 Principal Morphologies

In the paper we present principle types of patterns generated by rule (16.1), classified based on their shape internal morphology. There are five types of rules based on shapes of patterns generated

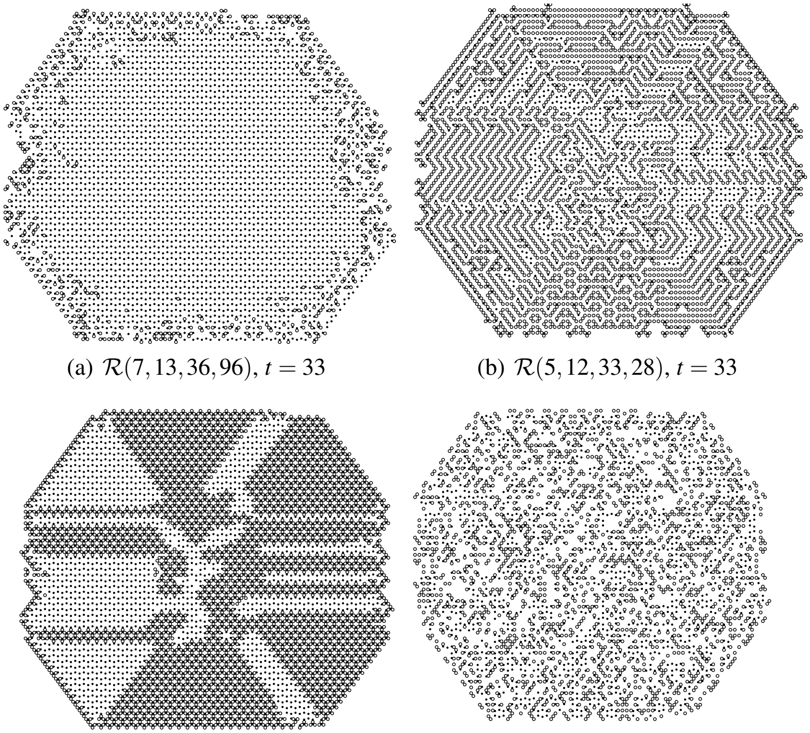
- $\mathcal{C}_1$: octagonally shaped growth patterns,
- $\mathcal{C}_2$: almost circularly shaped growth patterns,
- $\mathcal{C}_3$: stationary patterns shaped as convex and concave hulls,
- $\mathcal{C}_4$: patterns exhibiting dendritic growth
- $\mathcal{C}_5$: patterns showing still localizations and oscillators.

and five classes of rules based on internal morphology of patterns generated

- $\mathcal{M}_1$: solid patterns,
- $\mathcal{M}_2$: labyrinthine patterns,
- $\mathcal{M}_3$: wave-like patterns,
- $\mathcal{M}_4$: disordered patterns,
- $\mathcal{M}_5$: still, travelling and propagating localizations.

16.3.1 Class $\mathcal{C}_1$

Patterns (Fig. 16.3) generated by rules of class $\mathcal{C}_1$ are bounded by eight half-planes parallel to principle axes of the phyllosilicate lattice (Fig. 16.1). The pattern grow unlimitedly on in ’infinite’ lattice. Boundaries of the patterns may be well defined, e.g. configurations generated by rules $\mathcal{R}(7,13,36,96)$ (Fig. 16.3a), $\mathcal{R}(5,12,33,28)$ (Fig. 16.3b) and $\mathcal{R}(6,13,63,76)$ (Fig. 16.3c) or slightly ‘fuzzified’, as e.g. those produced by rule $\mathcal{R}(3,4,41,70)$ (Fig. 16.3d) . Rules of this class are usually found in the right bottom quadrant of μ_o–μ_s map (Fig. 16.2b), especially when solid domains are internal morphologies. The rules occupy right top quadrant of μ–α map in case of solid growing domains and tends towards centre of the map when patterns generated exhibit wave-like internal morphologies (Fig. 16.2a). The rules of class $\mathcal{C}_1$



(a) $\mathcal{R}(7,13,36,96)$, $t = 33$

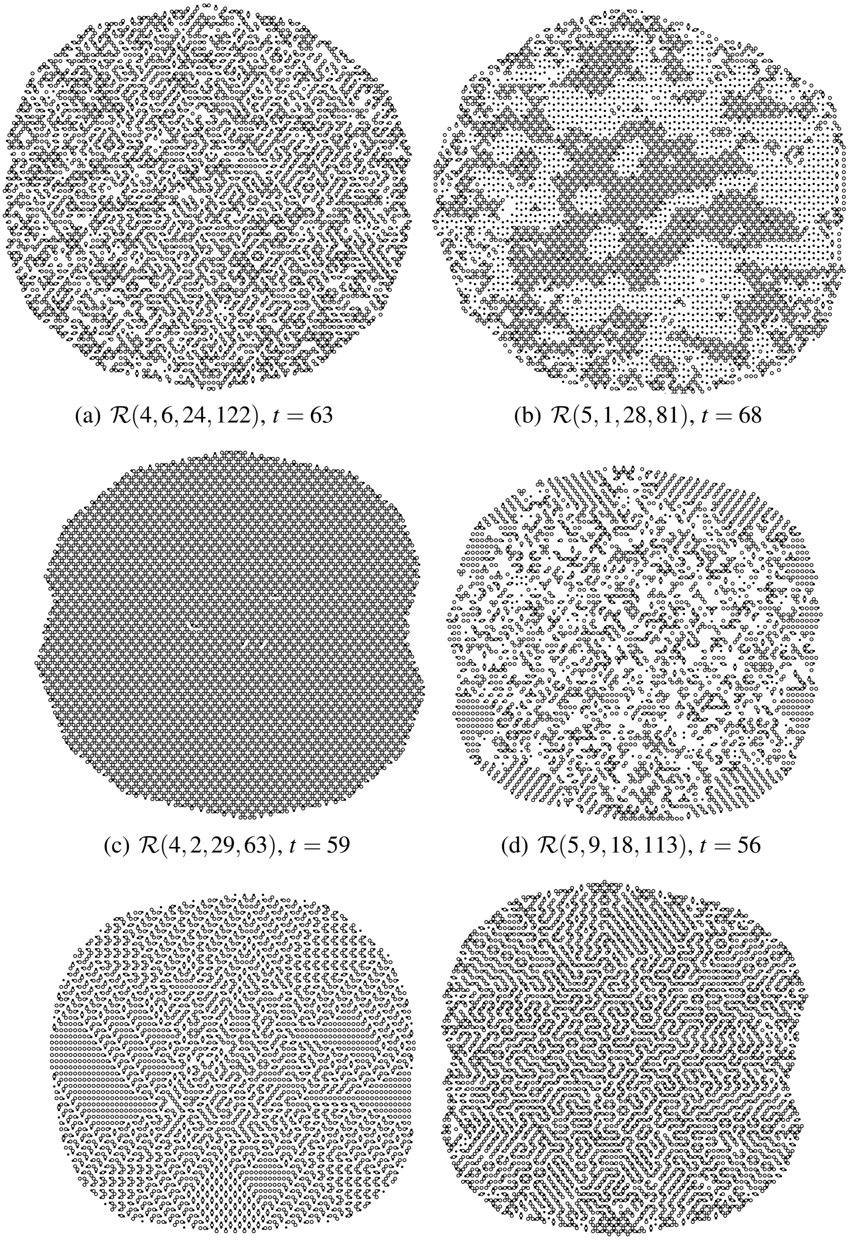
(b) $\mathcal{R}(5,12,33,28)$, $t = 33$

(c) $\mathcal{R}(6,13,63,76)$, $t = 31$

(d) $\mathcal{R}(3,4,41,70)$, $t = 29$

Fig. 16.3 Configurations of $\mathcal{A}$ generated by rules of class $\mathcal{C}_1$. Snapshots of the configurations are taken at time step t, exact values of t are shown. Silicon automata in state 1 are shown by small black discs, oxygen automata in state 1 are shown by circles.

appear with probability 0.14 in random samplings of rules. Likelihood of the functions $L(\mathcal{C}) = ((0.0,0.67,0.55,0.69), (0.53,0.42,0.58,0.47), (0.0,1.0,0.92,0.36,0.42,0.42,0.61), (0.69,0.53,0.39,0.47,0.55,0.58,0.55))$ shows that to form octagonal pattern nodes of automaton $\mathcal{A}$ will have the following transition functions. A resting silicon node most likely to take state 1 if it has one or three neighbours in state 1; a resting oxygen node very likely to take state 1 if it has two or six neighbours in state 1. A silicon node remains in state 1 with high probability if it has two neighbours. An oxygen node remains in state 1 more likely if it has no neighbours in state 1.



(a) $\mathcal{R}(4,6,24,122)$, $t = 63$

(b) $\mathcal{R}(5,1,28,81)$, $t = 68$

(c) $\mathcal{R}(4,2,29,63)$, $t = 59$

(d) $\mathcal{R}(5,9,18,113)$, $t = 56$

(e) $\mathcal{R}(6,10,31,66)$, $t = 66$

(f) $\mathcal{R}(5,3,24,125)$, $t = 54$

Fig. 16.4 Configurations of $\mathcal{A}$ generated by rules of class $\mathcal{C}_2$. Snapshots of the configurations are taken at time step t, exact values of t are shown. Silicon automata in state 1 are shown by small black discs, oxygen automata in state 1 are shown by circles.

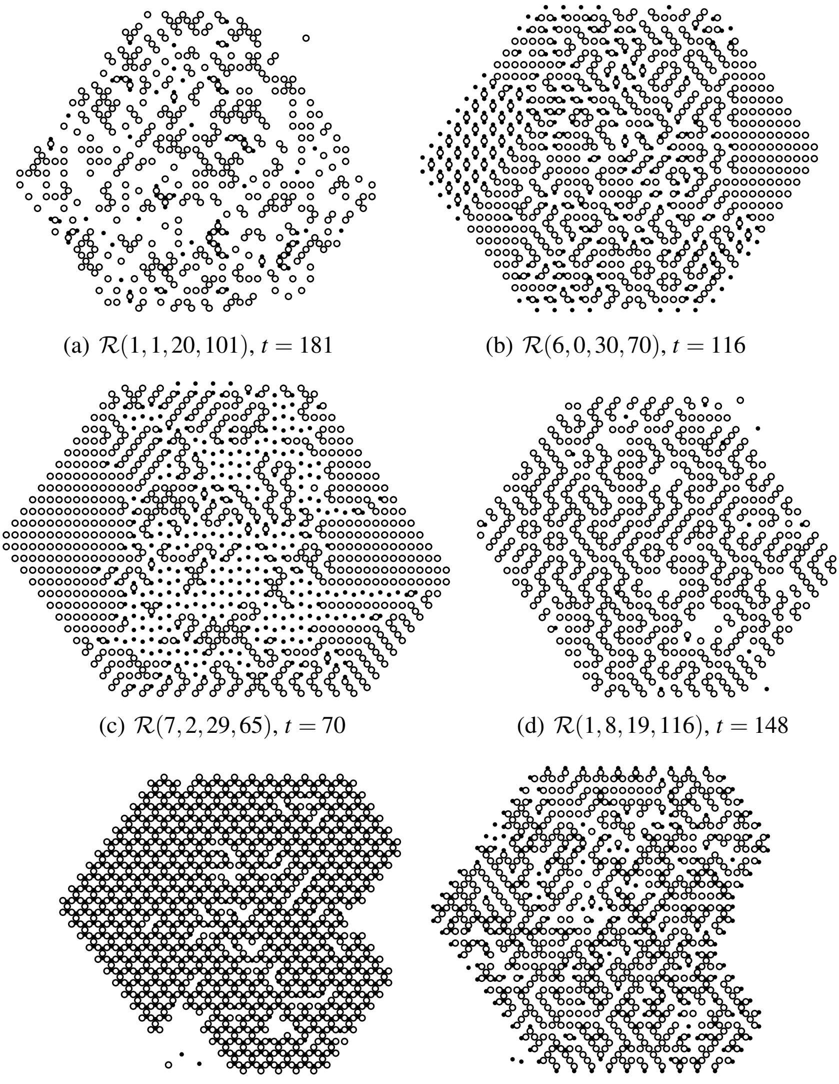
16.3.2 Class $\mathcal{C}_2$

Boundaries of configurations generated by rules of class $\mathcal{C}_2$ by time step t are close, in shape and position, to a circle of radius t (Fig. 16.4). Rules of $\mathcal{C}_2$ appear with probability 0.21 in a random sampling of rules. The likelihood representation of the rules $L(\mathcal{C}) = ((0.0, 0.83, 0.57, 0.54), (0.51, 0.6, 0.46, 0.4), (0.0, 0.4, 0.89, 0.51, 0.46, 0.57, 0.57), (0.68, 0.57, 0.49, 0.57, 0.63, 0.51, 0.46))$ indicates that a silicon atom is more likely to become unresting, undergo state transition $0 \to 1$, when it has one neighbour in state 1. Oxygen atoms undergo transition $0 \to 1$ when they have two neighbours in state 1. Automata remain in state 1 more likely when they have none or four neighbours in state 1.

Typically patterns of class $\mathcal{C}_2$ show a low level of activity and above average mass, see e.g. positions of rules $R(4, 2, 29, 63)$ and $R(5, 3, 24, 125)$ on μ–α map (Fig. 16.2a). Ratios of non-resting silicon and oxygen automata depends on internal morphology of patterns generated (Fig. 16.2b). For example, a pattern produced by rule $R(4, 2, 29, 63)$ has almost no silicon nodes in state 1, prevailing majority of non-resting nodes are oxygen automata (Fig. 16.4b). Moreover, nodes taken state 1 remain in this state forever, thus low level of activity, and the rule's positioning very close to the beginning of μ_s axis. Labyrinthine pattern grown by rule $R(5, 3, 24, 125)$ is composed of chains and rings of oxygen and silicon nodes in state 1, where a non-resting oxygen node is always followed by a non-resting silicon atom (Fig. 16.4f). This is why rule $R(5, 3, 24, 125)$ is positioned near a centre, i.e. average values, of the plane μ_o–μ_s (Fig. 16.2b).

16.3.3 Class $\mathcal{C}_3$

Boundaries of the configurations generated by rules of class $\mathcal{C}_3$ resemble discrete convex (Fig. 16.5a–d) and concave (Fig. 16.5ef) hulls [1, 19, 43]. The boundaries are stationary, the patterns do not propagate further when concave or convex hulls are formed. The rules are characterised by low density μ and activity level α, see e.g. position of rule $R(1, 8, 19, 116)$ in (Fig. 16.2a). The rules of $\mathcal{C}_3$ are quite rate, they appear with probability 0.051 in random samplings. As we can see in $L(\mathcal{C}) = ((0.0, 0.24, 0.76, 0.43), (0.57, 0.71, 0.48, 0.33), (0, 0.05, 0.81, 0.62, 0.67, 0.29, 0.52), (0.91, 0.67, 0.71, 0.71, 0.33, 0.48, 0.57))$, resting silicon automata more likely to take state 1 if they have two neighbours in state 1, they remain in state 1 with high likelihood if they have non or one neighbour in state 1. An oxygen automaton takes state 1 more likely if it has two, three or four neighbours in state 1, and it remains in state one if there are less than four neighbours in state 1.



(a) $\mathcal{R}(1,1,20,101)$, $t = 181$

(b) $\mathcal{R}(6,0,30,70)$, $t = 116$

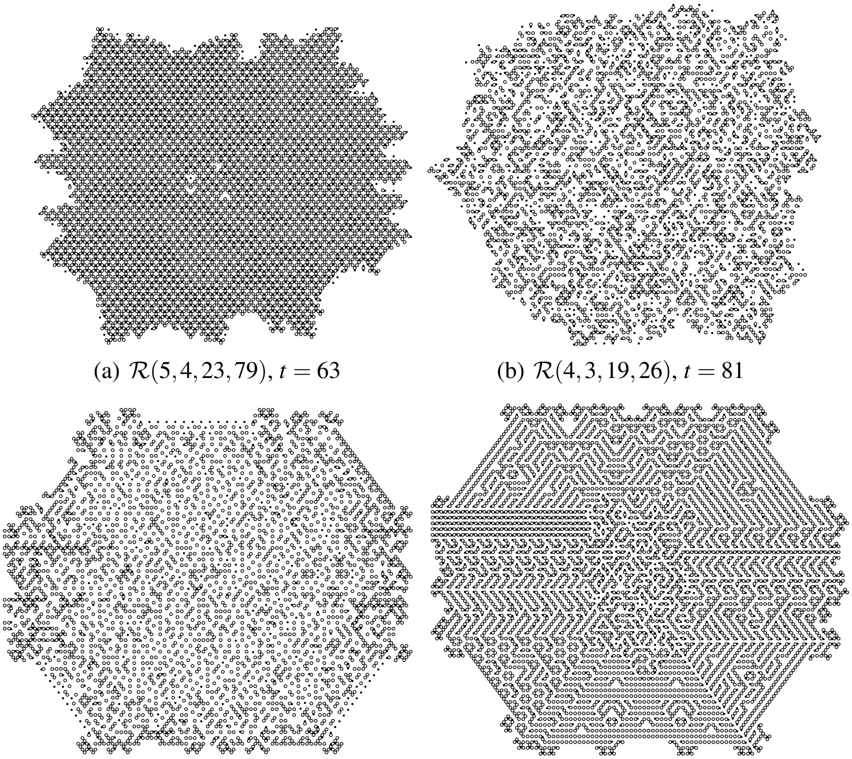
(c) $\mathcal{R}(7,2,29,65)$, $t = 70$

(d) $\mathcal{R}(1,8,19,116)$, $t = 148$

(e) $\mathcal{R}(3,9,17,127)$, $t = 67$

(f) $\mathcal{R}(5,12,11,29)$, $t = 129$

Fig. 16.5 Configurations of $\mathcal{A}$ generated by rules of class $\mathcal{C}_3$. Boundaries of the patterns are stationary. Snapshots of the configurations are taken at time step t, exact values of t are shown. Silicon automata in state 1 are shown by small black discs, oxygen automata in state 1 are shown by circles.



(a) $\mathcal{R}(5,4,23,79)$, $t = 63$

(b) $\mathcal{R}(4,3,19,26)$, $t = 81$

(c) $\mathcal{R}(17,9,36,119)$, $t = 35$

(d) $\mathcal{R}(4,14,32,60)$, $t = 36$

Fig. 16.6 Configurations of $\mathcal{A}$ generated by rules of class $\mathcal{C}_4$. Snapshots of the configurations are taken at time step t, exact values of t are shown. Silicon automata in state 1 are shown by small black discs, oxygen automata in state 1 are shown by circles.

16.3.4 Class $\mathcal{C}_4$

The rules are rare, they occur with frequency 0.04 in random samplings. Patterns generated by rules of this class show non-uniform growth rate along its wave-fronts, reflected in formation of growth-cones or small dendritic protrusions. These growth-cones grow faster than the rest of the boundary. The growth-cones can be positioned irregularly along the boundary, see e.g. Fig. 16.6ab, or more or less symmetrically, e.g. Fig. 16.6cd. The non-uniformity of wave-front propagation is due to high degree of 'non-linearity' of cell-state transition function: numbers of 1-state neighbours responsible for a state transition can differ as much as four: $L(\mathcal{C}) = ((0.0, 0.75, 0.45, 0.7), (0.3, 0.6, 0.5, 0.25), (0.0, 0.7, 0.3, 0.25, 0.5, 0.6, 0.5), (0.7, 0.55, 0.5, 0.55, 0.7, 0.5, 0.45))$. A silicon automaton makes the state transition $0 \rightarrow 1$ when it has one or three neighbours in state 1, and the automaton keeps its state 1 if there is only one non-resting neighbour. A resting oxygen automaton switches to state 1 if there are one or five non-resting neighbours; the automaton stays in state 1 if it has none or four 1-state neighbours.

16.3.5 Class $\mathcal{C}_5$

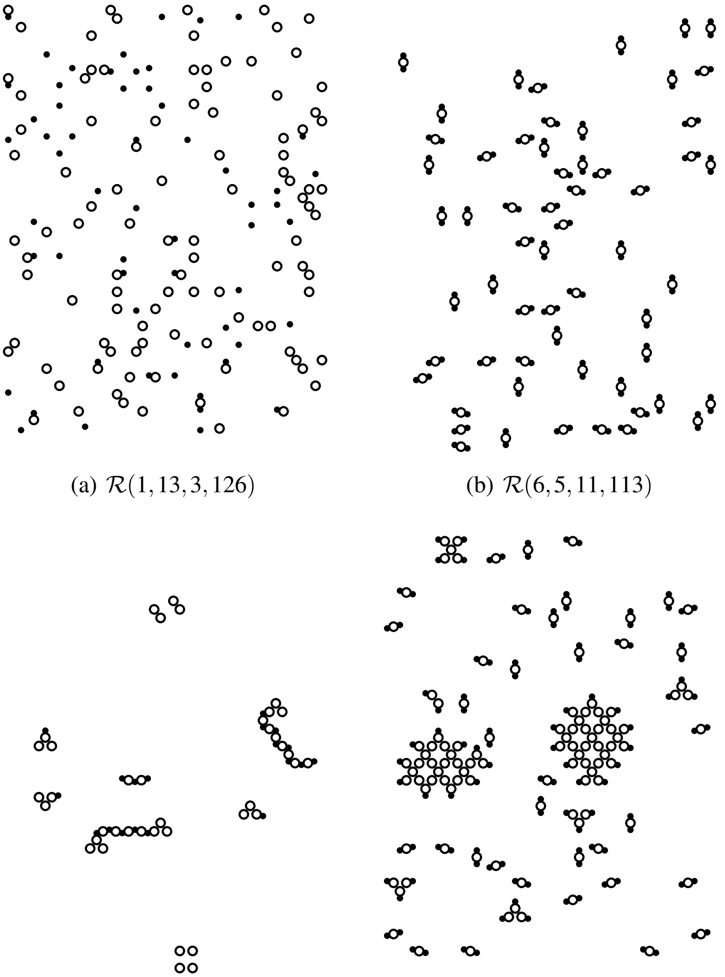
Initial random configuration of 1-states shrinks to few still localisations and oscillators in automata of class $\mathcal{C}_5$. Typical still localisations are singletons of oxygen or silicon automata in non-resting state (Fig. 16.7a) or tiny clusters of oxygen and silicon automata (Fig. 16.7b). The still and oscillating localizations do sometimes assemble into chains or rings e.g. rule $\mathcal{R}(1,3,17,45)$ (Fig. 16.7c) or into regular two-dimensional structures, e.g. rule $\mathcal{R}(4,6,11,95)$ (Fig. 16.7d). The rules of the class are relatively common and appear with probability 0.23 in random samplings. Statistics of functions from $\mathcal{C}_5$, $L(\mathcal{C}) = ((0.0, 0.52, 0.661, 0.6), (0.43, 0.38, 0.52, 0.51), (0.0, 0.01, 0.15, 0.46, 0.55, 0.49, 0.49), (0.69, 0.48, 0.54, 0.49, 0.43, 0.37, 0.58))$, shows that silicon automata more likely switches from state 0 to 1 and back when they have one, two or three neighbours in state 1. Thus transitions between 0 and 1 would be equiprobable, however the transition $1 \to 0$ also occurs when silicon automaton has no neighbours in non-resting state. Thus, in general, initially random pattern of non-resting automata reduces to few localizations.

16.3.6 Class $\mathcal{M}_1$

Node state transition rules of this morphological class can be found in all shape based classes. Thus, rule $\mathcal{R}(7,13,36,96)$ generates a growing octagonal domain of a regular lattice of oxygen automata in state 1 (Fig. 16.3a). Octagonal (Fig. 16.3a) and near-circular (Fig. 16.4b) patterns with separate domains of non-resting oxygen and silicon automata are produced by rules $\mathcal{R}(6,13,63,76)$ and $\mathcal{R}(5,1,28,81)$. A regular lattice of non-resting oxygen automata grows as a near-circular pattern from a random configuration in automata governed by rule $\mathcal{R}(4,2,29,63)$ (Fig. 16.4c). Regular patterns of silicon and oxygen are produced in a concave hull shaped patterns, rule $\mathcal{R}(3,9,17,127)$ Fig. 16.5e, and dendritic patterns, rule $\mathcal{R}(5,4,23,79)$ Fig. 16.6a.

16.3.7 Class $\mathcal{M}_2$

Labyrinthine patterns made of chains and rings of non-resting oxygen node with passages partly filled with non-resting silicon nodes are observed in octagonal configurations generated by rule $\mathcal{R}(5,12,33,28)$ (Fig. 16.3b). Rule $\mathcal{R}(5,3,24,125)$ transforms an initially random configuration into a growing labyrinthine pattern



(a) $\mathcal{R}(1,13,3,126)$

(b) $\mathcal{R}(6,5,11,113)$

(c) $\mathcal{R}(1,3,17,45)$

(d) $\mathcal{R}(4,6,11,95)$

Fig. 16.7 Configurations of $\mathcal{A}$ generated by rules of class $\mathcal{C}_5$. Snapshots of the configurations are taken at time step t, exact values of t are shown. Silicon automata in state 1 are shown by small black discs, oxygen automata in state 1 are shown by circles.

made of chains of non-resting silicon-oxygen pairs (Fig. 16.4f). Convex and concave halls grown by rules $\mathcal{R}(1,8,19,116)$ (Fig. 16.5d) and $\mathcal{R}(5,12,11,29)$ (Fig. 16.5f) give a combination of oxygen only and silicon-oxygen chains. Averaging the rules, $L(\mathcal{C}) = ((0.0,0.89, 0.44,0.78), (0.44,0.56,0.56,0.67), (0.0,0.67,0.67,0.11,0.44, 0.22, 0.44), (0.89, 0.89,0.78,1.0,0.78,0.44,0.44))$, we find that resting silicon becomes non-resting if it has one or three neighbours and resting oxygen takes state 1 if its two or three neighbours in state 1. A node of a chain have two neighbours and an end-node of chain has just neighbour. This is why most labyrinthine patterns are made of oxygen nodes.

16.3.8 Class $\mathcal{M}_3$

Wave-like patters are analogues of target and spiral wave-fronts in excitable nonlinear media. The wave-fronts may be travelling or stationary. The wave-like patterns are represented by chains of automaton in non-resting states arrange circularly or semi-circularly around initial source of perturbation. Likelihood representation of class $\mathcal{M}_3$ rules is $L(\mathcal{C}) = ((0.0,0.5,0.3,0.6), (0.3,0.6,0.4,0.7), (0.0,1.0,0.9,0.7, 0.6, 0.3,0.6), (0.6,0.4, 0.3,0.5,0.7,0.2,0.5))$. A resting silicon automaton is more likely to take state 1 when two or three neighbours are in state 1; it is more likely to remain in state 1 when three or one of its neighbours are non-resting.

16.3.9 Class $\mathcal{M}_4$

Rules generating disordered patterns are quite common, they appear with frequency 0.34 in random samplings. A frequency representation of the functions is $L(\mathcal{C})$ =((0.0,0.6,0.77,0.45), (0.55,0.55,0.45, 0.37), (0.0,0.7,0.55,0.57,0.45,0.62, 0.52), (0.7,0.52,0.47,0.5,0.5,0.4,0.5)). The disordered interior can be observed in patterns generated by rule $\mathcal{R}(3,4,41,70)$ (Fig. 16.3d), $\mathcal{R}(5,9,18,113)$ (Fig. 16.4d), $\mathcal{R}(1,9,36,119)$ (Fig. 16.6c).

16.3.10 Class $\mathcal{M}_5$

Rules of this class exhibit still and travelling localizations and generators of localizations. Class $\mathcal{M}_5$ overlaps with class $\mathcal{C}_5$ because localizations do not have internal morphology, their ‘internal morphology’ is their shape. There are rules in $\mathcal{M}_5$ which could not be found in any of the classes $\mathcal{C}$. The rules of class $\mathcal{M}_5$ and patterns generated by them are discussed in details in Sect. 16.4.

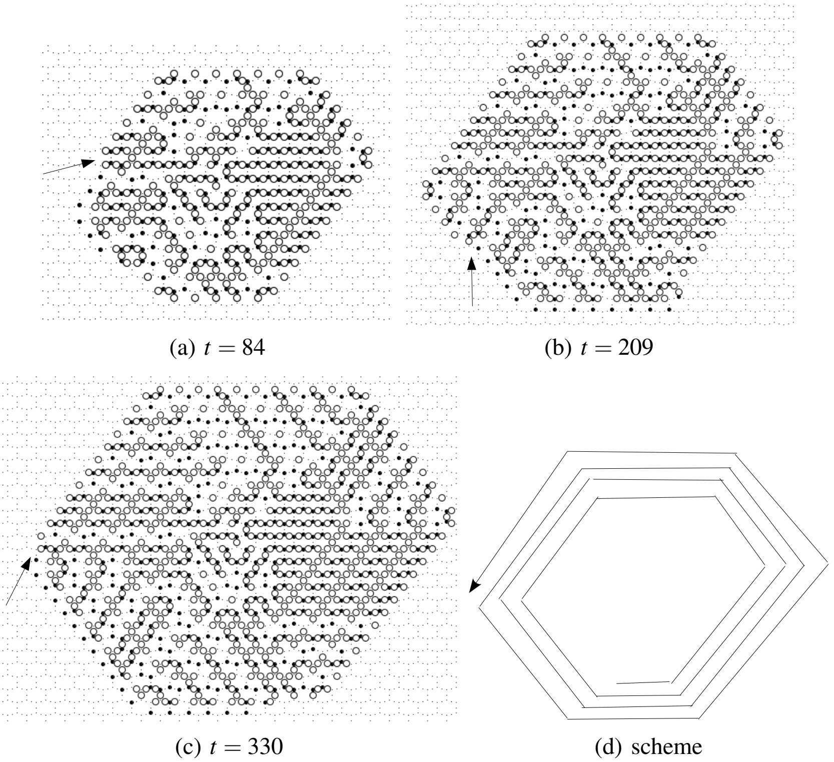


Fig. 16.8 The slowest growing configuration generated by rule $\mathcal{R}(7,5,31,33)$. (a–c) Snapshots of the configuration, growing from initial random rectangular domain of 39×39 nodes, recorded at 84th, 209th and 330th steps of evolution. Growth points are shown by arrows. (d) a Scheme of the growth. Silicon automata in state 1 are shown by small black discs, oxygen automata in state 1 are shown by circles. Automata in state 0 are shown by dots.

16.3.11 Sub-linearly Growing Patterns

Wave-like patterns and gliders are fastest growing configurations. Their linear size increases linearly with time. What would be a slowest expanding pattern? The slowest we found so far emerges in automata governed by rule $\mathcal{R}(7,5,31,33)$, when an initial random configuration of states 1 is transformed to a convex hull shaped pattern (Fig. 16.8a). The convex patterns emerged in rule $\mathcal{R}(7,5,31,33)$, in general, do not grow anymore. Boundary of the pattern consists of chains of automata in state 1. Oxygen and silicon automata take state 1 in turns. In some situations, such configuration of non-resting oxygen and silicon atoms occurs (Fig. 16.8abc, shown by arrow) that propagates along chains of silicon atoms and build a chain of non-resting oxygen atoms along. This solitary configuration, a growth point, always propagate along the boundary of the convex hull pattern (Fig. 16.8c). Therefore a linear size (along of the axis) of the pattern increases by two nodes only when the growth

Table 16.1 Parameters of localizations. For each localisation L we indicate its period p, minimum $w_{\min}$ and maximum $w_{\max}$ numbers of non-resting nodes, minimum $w^o_{\min}$ and maximum $w^o_{\max}$ numbers of non-resting oxygen nodes, and minimum $w^s_{\min}$ and maximum $w^s_{\max}$ numbers of non-resting oxygen nodes. Minimal values in min columns and maximum values in max columns are underlined.

L	p	$w_{\min}$	$w_{\max}$	$w^o_{\min}$	$w^o_{\max}$	$w^s_{\min}$	$w^s_{\max}$
G^{16}_{65}	$\underline{16}$	6	13	2	9	2	8
G^{12}_{68}	12	7	13	2	6	2	7
G^{4}_{72}	4	4	11	2	6	2	6
O^{5}_{65}	5	6	10	2	8	4	8
O^{6}_{65}	6	4	9	2	5	2	4
O^{6A}_{65}	6	3	5	$\underline{1}$	3	2	2
O^{6}_{68}	6	3	6	$\underline{1}$	3	2	3
O^{12}_{68}	12	8	$\underline{26}$	4	$\underline{22}$	0	$\underline{16}$
O^{2}_{72}	2	3	4	3	3	4	4
O^{4}_{72}	4	6	10	3	6	4	9
O^{12}_{72}	12	$\underline{2}$	5	$\underline{1}$	3	$\underline{1}$	3

point made a full turn along boundary of the pattern. Thus the size of the pattern is bounded as $O(\frac{1}{t})$.

16.4 Game of Life Phyllosilicate Automata

Exploring the 'activity-density' space in the loci with high activity and low density we discovered three minimal — minimality in number of '1's in strings **s** and **o** — rules that support gliders, $\mathcal{R}_{65} = \mathcal{R}(5,2,16,65)$, $\mathcal{R}_{68} = \mathcal{R}(5,2,16,68)$, and $\mathcal{R}_{72} = \mathcal{R}(5,2,16,72)$. The rules belong to class $\mathcal{M}_5$.

The rules have the same rules for state transitions of silicon nodes, $\mathbf{s}_0 = (0101)$ and $\mathbf{s}_1 = (0010)$, and oxygen nodes in resting states, $\mathbf{o}_0 = (0010000)$. Only behaviour of oxygen nodes in non-resting state differs: $\mathbf{o}_1 = (1000001)$ in rule $\mathcal{R}_{65}$, (1000100) in $\mathcal{R}_{68}$, and (1001000) in $\mathcal{R}_{72}$. Each of thee three rules $\mathcal{R}_{65}$, $\mathcal{R}_{68}$ and $\mathcal{R}_{72}$. generate its own unique glider and several oscillators. Parameters of gliders and oscillators are shown in Tab. 16.1. We have also discovered one still life, observed in rule $\mathcal{R}_{68}$ and a glider gun, found in rule $\mathcal{R}_{72}$.

Gliders, oscillators and still lifes are characteristic patterns of Conway's Game of Life cellular automata. Phyllosilicate analogs of the Game of Life, or Life-like automata, obey the following rules. Silicon node in state 0 takes state 1 only if it has one or three neighbours in state 1. Silicon node in state 1 remains in state 1 if it has two neighbours in state 1. Oxygen node in state 0 takes state 1 only if it has two neighbours in state 1. An oxygen node in state 1 remains in state 1 if it has no neighbours in state 1 or six in rule $\mathcal{R}_{65}$, four in rule $\mathcal{R}_{68}$ and three in rule $\mathcal{R}_{72}$ neighbours are in state 1.

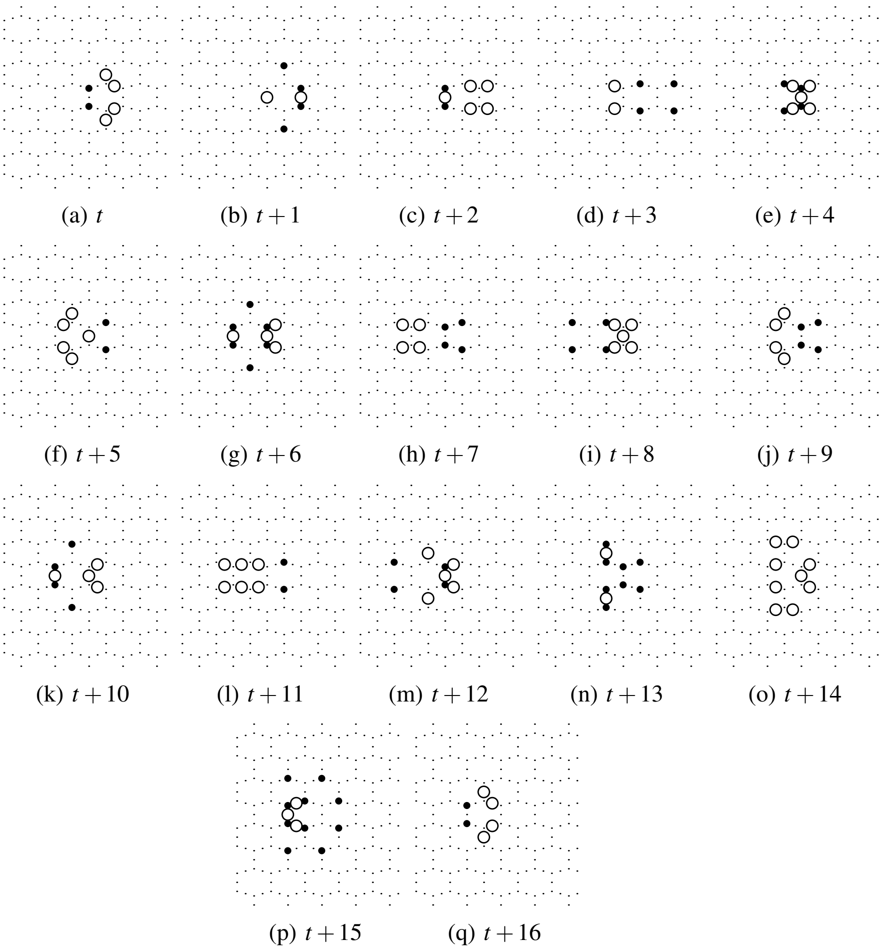
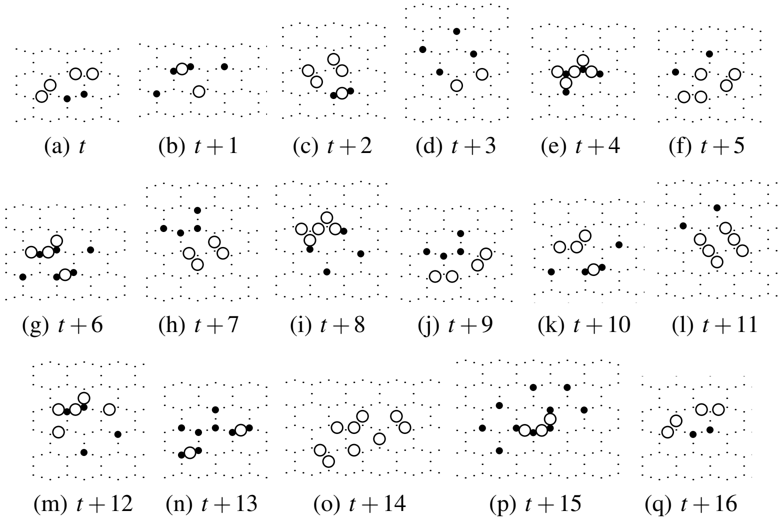


Fig. 16.9 Glider G^{16}_{65} in rule $\mathcal{R}_{65}$. Glider moves west. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

16.4.1 Gliders

Glider G^{16}_{65} is observed in configurations produced by rule $\mathcal{R}_{65}$, see examples of gliders travelling west and south-east in Figs. 16.9 and 16.10. Glider G^{16}_{65} is theaviest amongst gliders discovered (Tab. 16.1). It has longest period — 16 time steps — and maximum weight of 13 non-resting nodes (Figs. 16.9p and 16.10p). The glider undergoes thickening and lengthening phases. In thickening phase the glider is larger, in terms of maximum between non-resting nodes, along a normal to its velocity vector, see e.g. Figs. 16.9abfjko. In lengthening phase glider is larger along

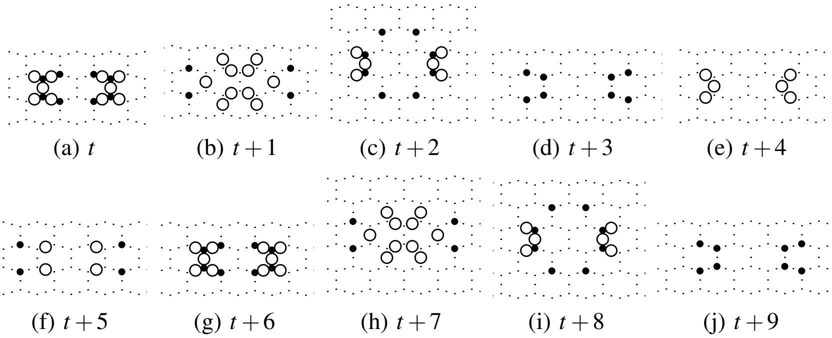


(a) t (b) $t+1$ (c) $t+2$ (d) $t+3$ (e) $t+4$ (f) $t+5$

(g) $t+6$ (h) $t+7$ (i) $t+8$ (j) $t+9$ (k) $t+10$ (l) $t+11$

(m) $t+12$ (n) $t+13$ (o) $t+14$ (p) $t+15$ (q) $t+16$

Fig. 16.10 Glider G_{65}^{16} in rule $\mathcal{R}_{65}$. Glider moves south-east. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.



(a) t (b) $t+1$ (c) $t+2$ (d) $t+3$ (e) $t+4$

(f) $t+5$ (g) $t+6$ (h) $t+7$ (i) $t+8$ (j) $t+9$

Fig. 16.11 Head on collision of gliders in rule $\mathcal{R}_{65}$, zero vertical shift. Glider travelling east collides with glider travelling west. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

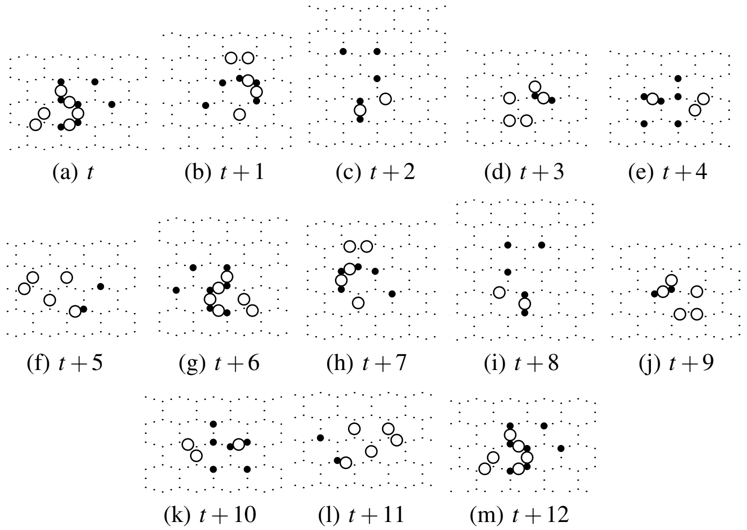

(a) t (b) $t+1$ (c) $t+2$ (d) $t+3$ (e) $t+4$ (f) $t+5$ (g) $t+6$ (h) $t+7$ (i) $t+8$ (j) $t+9$ (k) $t+10$ (l) $t+11$ (m) $t+12$

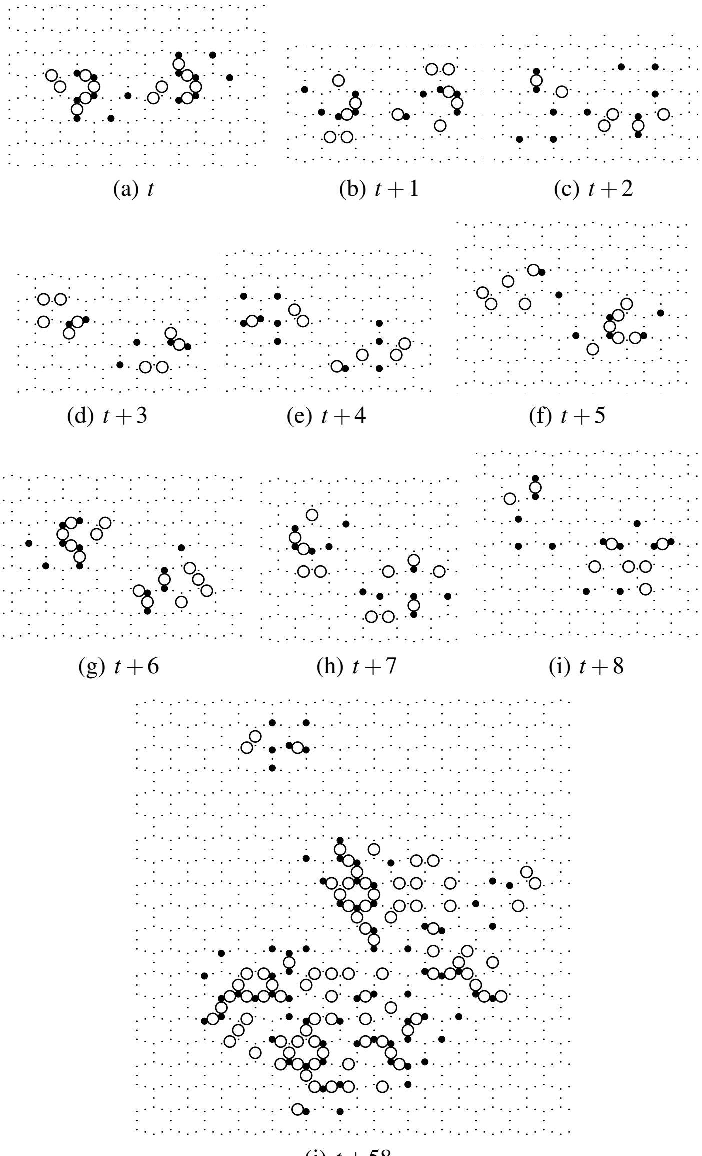
Fig. 16.12 Glider G_{68}^{12} in rule $\mathcal{R}_{68}$. Glider moves down from north straight to south. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

parallel to its velocity vector, see e.g. Figs. 16.9dhjl. Ratio of silicon to oxygen non-resting nodes changes during the glider's cycle as follows: $\frac{2}{4}$, $\frac{4}{2}$, $\frac{2}{5}$, $\frac{4}{2}$, $\frac{4}{5}$, $\frac{2}{5}$, $\frac{6}{4}$, $\frac{4}{4}$, $\frac{4}{5}$, $\frac{4}{4}$, $\frac{4}{4}$, $\frac{4}{4}$, $\frac{2}{6}$, $\frac{4}{5}$, $\frac{8}{2}$, $\frac{0}{9}$, $\frac{10}{3}$ and $\frac{2}{4}$.

Most collisions between gliders in rule $\mathcal{R}_{65}$ lead to explosions: an unlimitedly growing pattern is formed in the result of gliders interaction. Head on collision with nil vertical or horizontal shift between gliders is a rare case of collision where gliders do not explode (Fig. 16.11) but fuse into an oscillator. One glider travels east, left non-resting pattern in Fig. 16.11a, another glider travels west, right non-resting pattern in Fig. 16.11a. The gliders come into 'contact' with each other (Fig. 16.11b) and form a transient pattern (Fig. 16.11b) of 4 non-resting silicon nodes and 10 non-resting oxygen nodes. This pattern is then transformed into an oscillator of period six (Fig. 16.11d–j).

Glider in rule $\mathcal{R}_{68}$ is shown in Fig. 16.12. The glider has period 12, and maximum weight 13 nodes (Tab. 16.1). Glider reflects along its south-north axis during motion, in middle of its cycle, namely glider state at time step t, $t = 1, 2, \cdots$, is vertically symmetric to the glider state at time step $t+6$. The glider does not show a pronounced thickening or lengthening but rather breathing around its centre of mass. Ratios of non-resting silicon nodes to non-resting oxygen nodes in the glider's cycle are changed as follows: $\frac{7}{6}$, $\frac{5}{5}$, $\frac{5}{2}$, $\frac{2}{5}$, $\frac{6}{3}$, $\frac{2}{5}$, $\frac{7}{6}$, $\frac{5}{5}$, $\frac{5}{2}$, $\frac{2}{5}$, $\frac{6}{3}$, $\frac{2}{5}$, $\frac{7}{6}$.

Head on collision between two gliders in $\mathcal{R}_{68}$ without offset leads to explosion of both gliders. For certain offsets one glider can survive collision. Such scenario is illustrated in Fig. 16.13. A glider moving north (left non-resting pattern in Fig. 16.13a) clips a glider moving south (right non-resting pattern in Fig. 16.13a).


(a) t

(b) $t+1$

(c) $t+2$

(d) $t+3$

(e) $t+4$

(f) $t+5$

(g) $t+6$

(h) $t+7$

(i) $t+8$

(j) $t+58$

Fig. 16.13 Interaction of gliders G_{68}^{12} in rule $\mathcal{R}_{68}$. Glider moving north brushes with glider moving south. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

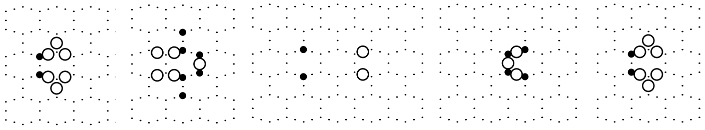


Fig. 16.14 Glider G^4_{72} in rule $\mathcal{R}_{72}$. The glider travels east. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

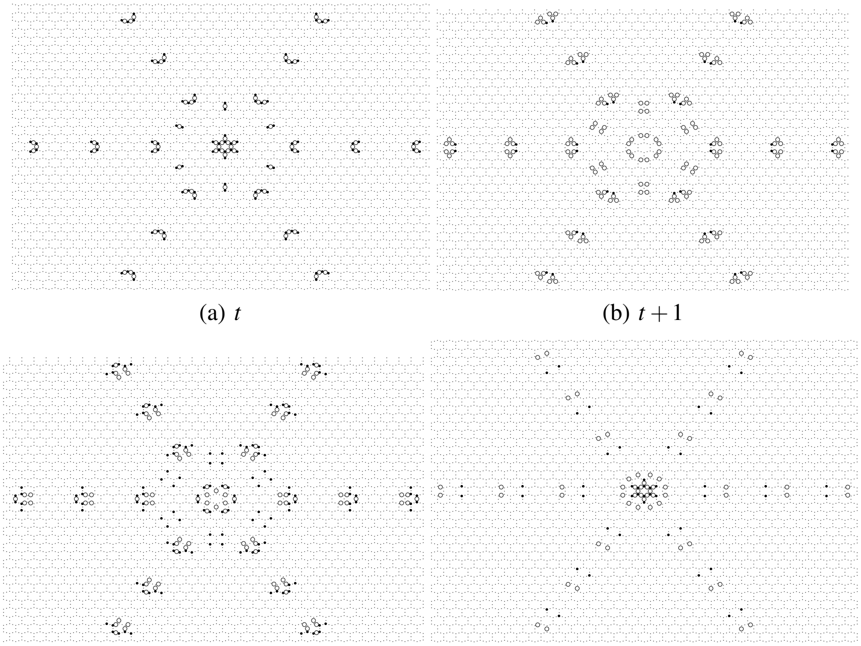


Fig. 16.15 Glider gun in rule $\mathcal{R}_{72}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

The glider moving north does not suffer any damage and continues its journey undisturbed. The glider moving south becomes tangled (Fig. 16.13b–d) and explodes (Fig. 16.13e–i and j).

Glider in rule $\mathcal{R}_{72}$ is shown in Fig. 16.14. It is smallest amongst gliders discovered and has smallest period. It has just four different states. Ratios of non-resting silicons to non-resting oxygens in the glider's cycle are $\frac{2}{6}$, $\frac{6}{5}$, $\frac{2}{2}$ and $\frac{4}{3}$.

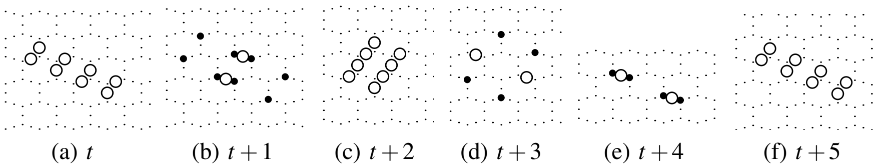


Fig. 16.16 Oscillator O^5_{65} in rule $\mathcal{R}_{65}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

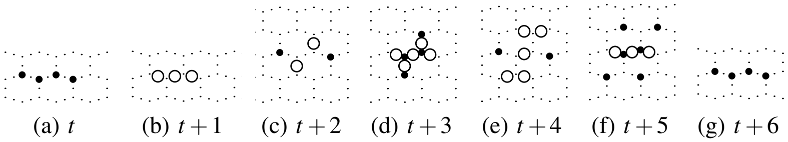


Fig. 16.17 Oscillator O^6_{65} in rule $\mathcal{R}_{65}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

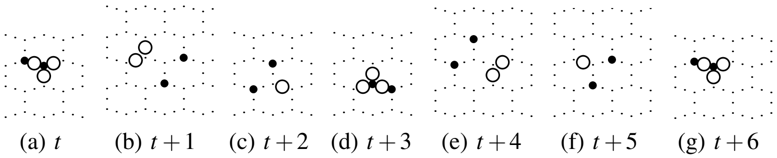


Fig. 16.18 Oscillator O^{6A}_{65} in rule $\mathcal{R}_{65}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

We did not find glider guns in rules $\mathcal{R}_{65}$ and $\mathcal{R}_{68}$, we believe they exist though and just wait to be found by someone being more patient. Glider gun in rule $\mathcal{R}_{72}$ is shown in Fig. 16.15.

16.4.2 Oscillators

Oscillator is a localised compact configuration of non-quiescent states which undergoes finite growth and modification but returns to its original state in a finite number of steps. Oscillators are generated by all three rules, discussed in the paper, and can be easily found by randomly perturbing the automaton lattice. Here we present a selection of most common oscillators. Some parameters of the oscillators are shown in Tab. 16.1. Periods of oscillators vary from 2, oscillator O^2_{72}, to 12, oscillators O^{12}_{68} and O^{12}_{72}. Minimum weights vary from 2, oscillator O^{12}_{72} to 6, oscillator O^5_{65}. Maximum weights vary from 4, oscillator O^2_{72}, to 26, oscillator O^{12}_{68}.

Three typical oscillators in rule $\mathcal{R}_{65}$ are shown in Fig. 16.16, 16.17 and 16.18. Oscillator O^5_{65} changes it configuration from two parallel chains of non-resting oxygen nodes oriented north-west to south-east (Fig. 16.16a) and north-east to south-west

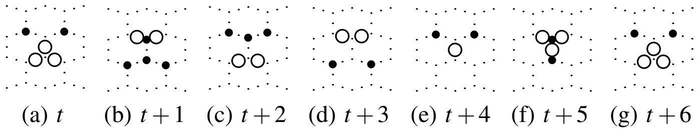


Fig. 16.19 Oscillator O^6_{68} in rule $\mathcal{R}_{68}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

(Fig. 16.16c) to sparse chains of non-resting silicons with few non-resting oxygens incorporated (Fig. 16.16bd). Oscillator O^5_{65} is also oscillator in rule $\mathcal{R}_{72}$.

Oscillator O^6_{65} switches its configuration between a single chain of non-resting silicons (Figs. 16.17ag), a single chain of non-resting oxygens (Figs. 16.17b), and regular non-linear arrangements of non-resting nodes (Figs. 16.17c–f). Oscillator O^{6A}_{65} is one of smallest oscillators (Fig. 16.18). It has period six. The oscillator configuration at time step t is symmetric — along north-east to south-west axis — to its configuration at time step $t+3$.

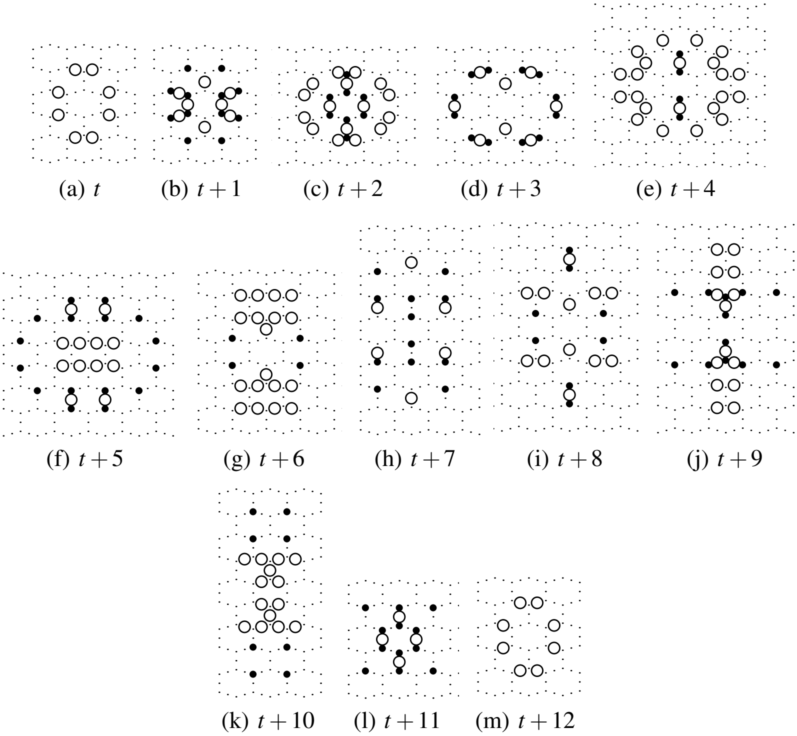
Exemplar oscillators discovered in rule $\mathcal{R}_{68}$ are shown Figs. 16.19 and 16.20. Oscillator O^6_{68} has the same configuration as oscillator O^{6A}_{65}, see Fig. 16.19f and Fig. 16.18adj. Oscillator O^{12}_{68} is the largest oscillator, amongst presented in the paper (Tab. 16.1). The oscillator has period 16, and consists of 28 non-resting nodes in its heaviest configuration (Fig. 16.20f).

Oscillator O^2_{72} is the smallest one (Tab. 16.1). It has just two configurations: one consists of four non-resting silicon nodes and another of three non-resting oxygen nodes (Fig. 16.21).

Oscillator O^{12}_{72} (Fig. 16.23) is the smallest oscillator with longest period. Its period is 12 time steps and maximum weight is 5 non-resting nodes, see e.g. Fig. 16.23d. When glider G^4_{72} collides to O^{12}_{72} there are three possible outcomes, depending on state of the oscillator at the collision and relative positions of the glider and the oscillator: oscillator and glider vanish in the result of collision, both localizations explode, and glider disappear but oscillator remains intact. In Conway's Game of Life an eater is a still life (stationary localisation that does not change its configuration) which repairs itself after collision with a glider or any other travelling localisation, while the colliding travelling localisation vanishes. Oscillator O^{12}_{72} is not a still life so it can be classified as oscillator-eater.

16.4.3 Still Life

Still lifes are localizations that do not change their pattern or a position during automaton development. So far we found still lifes only in rule $\mathcal{R}_{68}$: this is a ring of six oxygen and six silicon non-resting nodes as shown in Fig. 16.24a. This patterns becomes an oscillator in rule $\mathcal{R}_{65}$ (Fig. 16.24b) and is transformed into a glider gun in rule $\mathcal{R}_{68}$ (Fig. 16.15).


(a) t (b) $t+1$ (c) $t+2$ (d) $t+3$ (e) $t+4$

(f) $t+5$ (g) $t+6$ (h) $t+7$ (i) $t+8$ (j) $t+9$

(k) $t+10$ (l) $t+11$ (m) $t+12$

Fig. 16.20 Oscillator O_{68}^{12} in rule $\mathcal{R}_{68}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

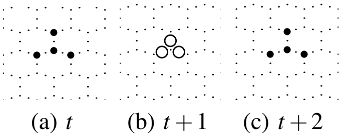

(a) t (b) $t+1$ (c) $t+2$

Fig. 16.21 Oscillator O_{72}^{2} in rule $\mathcal{R}_{72}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

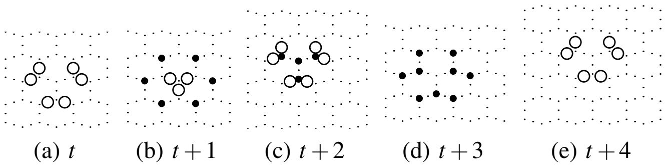

(a) t (b) $t+1$ (c) $t+2$ (d) $t+3$ (e) $t+4$

Fig. 16.22 Oscillator O_{72}^{4} in rule $\mathcal{R}_{72}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

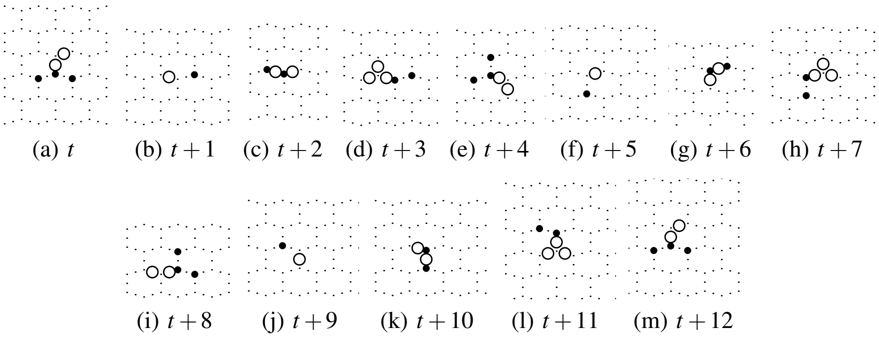


Fig. 16.23 Oscillator O_{72}^{12} in rule $\mathcal{R}_{72}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

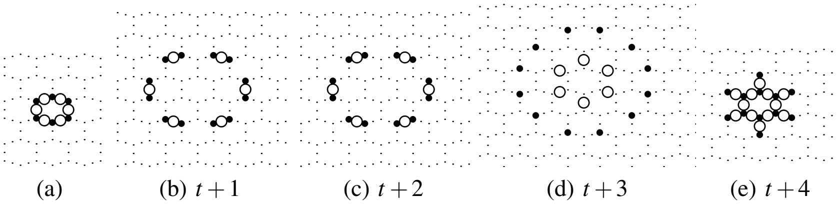


Fig. 16.24 Still localisation. (a) Still localisation in rule $\mathcal{R}_{68}$; in this rule the localisation acts as eater. The localisation (a) becomes an oscillation (b–e) in rule $\mathcal{R}_{65}$. Resting nodes are shown by dots. Non-resting silicon nodes are shown by small black discs. Non-resting oxygen nodes are shown by large circles.

16.5 Discussion

We uncovered principle morphologies of patterns emerging in automata models of phyllosilicates, two-dimensional regular networks of finite state machines imitating coordination lattice and atomic bounds of silicate sheets. We classified rules based on shape of patterns generated by the rules and internal morphologies of the patterns, and approximated distribution of functions on density of patterns and activity of cell-state transitions and maps of oxygen and silicon densities. Using shapes of patterns as a key characteristic we selected five classes: octagonally shaped patterns, almost circularly shaped, stationary patterns non-propagating beyond convex or concave hulls, patterns exhibiting dendritic growth, and patterns showing still localizations and oscillations. Internal morphology driven taxonomy is comprised of solid patterns, labyrinthine patterns, wave-like patterns, disordered patters; and still, travelling and propagating localizations. The classes proposed give rather a coarse classification of a rule space. There are rules which represent transient patterns that could be fitted in more than one shape based or internal morphology based class; such rules however do not dominate a rule space.

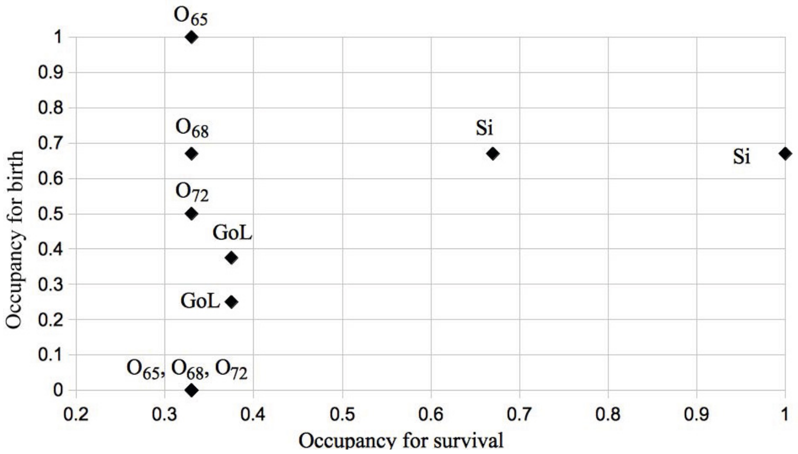


Fig. 16.25 Positions of Game of Life and phyllosilicate automata rules on 'neighbourhood occupancy necessary for birth', transition from state 0 to state 1, versus 'neighbourhood occupancy necessary for survival', transition from state 1 to state 1.

Amongst node state transition rules of phyllosilicate automata we discovered the rules exhibit behaviour similar to Conway's Game of Life automata. Classical Conway's Game of Life automata have rule B3/S23: a dead cell become alive, born if it has three live neighbours, a living cell survives if it has two or three live neighbours. In notation of Life-like automata the rules can be written as follows. All silicon nodes update their states by rule B23/S23. Rules of oxygen nodes are B2/S06 (rule $\mathcal{R}_{65}$), B2/S04 (rule $\mathcal{R}_{68}$) and B2/S03 (rule $\mathcal{R}_{72}$). Such notations could be somewhat misleading. In Conway's Game of Life a node (cell) has eight neighbours while in phyllosilicate automata a silicon node has three neighbours and oxygen node six neighbours. Thus, rules should be normalised by dividing number of non-resting neighbours necessary for birth and survival by neighbourhood size. Normalised rules are as follows. Normalised Conway's Game of Life rule is B$\frac{3}{8}$/S$\frac{2}{8}\frac{3}{8}$. Rule for silicon nodes in phyllosilicate is B$\frac{2}{3}$1/S$\frac{2}{3}$. Rules for oxygen nodes are B$\frac{2}{6}$/S01 ($\mathcal{R}_{65}$), B$\frac{2}{6}$/S0$\frac{4}{6}$ ($\mathcal{R}_{68}$), and B$\frac{2}{6}$/S0$\frac{3}{6}$ ($\mathcal{R}_{72}$). On the map 'neighbourhood occupancy necessary for birth', transition from state 0 to state 1, versus 'neighbourhood occupancy necessary for birth' (Fig. 16.25) Game of Life rules reside in the quadrant with below average occupancies. Occupancies necessary for silicon nodes to take non-resting state and to remain in the non-resting reside in the quadrant of above average occupancies. Rules for oxygen nodes are characterised by below average occupancies necessary for survival (Fig. 16.25). Distribution of rules on this map hints that conditions for survival are more critical than conditions for birth. Most rules supporting gliders obey a neigbourhood occupancy between 0.3 and 0.4 while neighbourhood occupancy for birth varies from 0 to 1.

We presented three types of node-state transition rules of a regular automaton network which mimics topology of a silicate sheet. All three rules studied support travelling and oscillating stationary localizations. Yet only one rule exhibits still localizations. Our search for phyllosilicate automata exhibiting Conway's Game of Life behaviour was extensive yet not exhaustive. Thus substantial chances remain that glider guns could be found in rules $\mathcal{R}_{65}$ and $\mathcal{R}_{68}$, and still lifes in rules $\mathcal{R}_{65}$ and $\mathcal{R}_{72}$.

Most interactions between localizations lead to explosions of the localizations, when unlimitedly growing patterns of non-resting states are formed. Just few scenario of collisions lead to annihilation of both or at least one of colliding localizations. This may somehow limit, however not totally take away, an applicability of phyllosilicate automata in design of collision-based circuits [2]. Further studies in the interaction of localizations are required to make a definite conclusion about usefulness of phyllosilicate automata in the unconventional computing field.

What are potential impacts of the results?

Phyllosilicate automata is a new species among regular automaton networks. The phyllosilicate automata are not strictly cellular automata, because they have two types of nodes which updates their states by two different functions. They are large-scale regular lattice of finite state machines and thus make a valuable addition to a family of cellular automata on non-othogonal and non-periodic lattices [14, 21, 35]. The phyllosilicate automata produce a rich spectrum of morphologies generated in the automata space-time dynamics. Shape-based and morphological classification of phyllosilicate automata well matches analogous classifications developed in two-dimensional binary cellular automata [4, 6] and automata models of excitable media [5, 13].

The phyllosilicate automata mimic structure of silicate sheets. Binary states can be seen as analogies of atoms' excitations and positions: state 0 corresponds to a baseline resting atom state and state 1 represents a disturbed atom state with possible changes of inter-atomic bonds. Patterns travelling in phyllosilicate automata imitate spatially extended defects propagating in silicate sheets. Localisations and defects are vehicles of computation in molecular arrays [11, 12]. We envisage state transitions functions — especially those support gliders, puffers, still localisations and oscillators — could be used as fast-prototyping tools for designing and future manufacturing of collision-based computing devices in silicon sheets.

How gliders, oscillators and still lifes could be represented in real phyllosilicate lattices?

We believe they will be seen as domains of localised defects, e.g. recombination-induced defects formation [42] occurred in tetrahedra, and vacancy-impurity pairs or isolated interstitial silicon defects [45]. The imperfection, vacancy, and self-interstitial, substitutional and interstitial impurities are important because they are necessary for setting up travelling localizations of e.g. a substitutional impurity atom through the lattice. The travelling impurities cause local deformations of the phyllosilicate lattice, e.g. via recombination of vacancies and carriers, these deformations produce waves. Thus, for example, state '1' of a node can represent a point interstitial defect, where the host atom moves to a non-lattice position [39]. The point defects can form travelling localizations, phenomenologically similar to gliders. Such localizations of defects may not be stable, even at low temperature, and disappear. However, duration of their life time could be sufficient to execute one or more collision-based logical gates.

A typical scenario of generating a glider on a phyllosilicate lattice would be as follows. A locally irradiation of the lattice with 1-3 MeV electrons, gamma rays, or fast neutrons [24] would move silicon atoms into metastable interstitial states. Thus a negatively charged travelling vacancy is formed. Direction of this localisation travelling could be controlled by inducing lattice vibrations and affecting electrons transitions. Obviously, to obtain an experimental confirmation would be a very difficult task.

There is at least some analogy between gliders-supporting automata rules and conditions for emergence of a travelling vacancy. In all glider-supporting phyllosilicate automata studied a silicon node excites and stays excited if it has two or three excited neighbours. An oxygen is excited if it has two excited neighbours. As hinted in [24] negatively charged travelling vacancies can migrate only if at least two interatomic bonds are broken.

When complemented by molecular-dynamics method [27] or reaction-diffusion computational approach [44] of simulating defects propagation, our approach can be useful in designing close to reality computing circuits based on travelling defects and clusters of defects. This will be a topic of future studies to find what exact structure of defect, localisation or dislocation [17, 37] corresponds to gliders and still lifes.

References

1. Adamatzky, A.: Identification of Cellular Automata. Taylor and Francis (1994)
2. Adamatzky, A. (ed.): Collision-Based Computing. Elsevier (2002)
3. Adamatzky, A., De Lacy Costello, B., Asai, T.: Reaction Diffusion Computers. Elsevier (2005)
4. Adamatzky, A., Martinez, G.J., Mora, J.C.S.T.: Phenomenology of reaction diffusion binary-state cellular automata. Int. J. Bifurcation and Chaos 16, 2985–3005 (2007)
5. Adamatzky, A., Chua, L.: Phenomenology of retained refractoriness: On semi-memristive discrete media. Int. J. Bifurcation and Chaos 22, 1230036 (2012)
6. Adamatzky, A.: Reaction-Diffusion Automata. Springer (2012)
7. Adamatzky, A.: On binary-state phyllosilicate automata. Int. J. Bifurcation and Chaos (2013)
8. Adamatzky, A.: On oscillators in phyllosilicate excitable automata. Int. J. Mod. Phys. C 24, 1350034 (2013) (10 pages)
9. Adamatzky, A.: Game of Life on Phyllosilicates: Gliders, Oscillators and Still Life. Physics Letters A 377, 1597–1605 (2013)
10. Ballantine, J.A., Purnell, J.H., Thomas, J.M.: Sheet silicates: broad spectrum catalysts for organic synthesis. J. Molecular Catalysis 27, 157–167 (1984)
11. Bandyopadhyay, A., Pati, R., Sahu, S., Peper, F., Fujita, D.: Massively parallel computing on an organic molecular layer. Nature Physics 6, 369–375 (2010)
12. Adamatzky, A.: Molecular computing: Aromatic arithmetic. Nature Nature Physics 6, 325–326 (2010)

13. Adamatzky, A., Chua, L.: Memristive excitable cellular automata. Int. J. Bifurcation and Chaos 21, 3083 (2011)
14. Bays, C.: The discovery of glider guns in a Game of Life for the triangular tessellation. J. Cellular Automata 2(4), 345–350 (2007)
15. Bergaya, F., Theng, B.G.K., Lagaly, G. (eds.): Handbook of Clay Science. Elsevier (2006)
16. Bleam, W.: Atomic theories of phyllosilicates: Quantum chemistry, statistical mechanics, electrostatic theory, and crystal chemistry. Reviews Geophysics 31(1), 51–73 (1993)
17. Bulatov, V.V., Justo, J.F., Wei Cai, S., Yip, A.S., Argon, T., Lenosky, M., de la Rubia, T.D.: Parameter-free Modeling of Dislocation Motion: The Case of Silicon. Philosophical Magazine A 81, 1257–1281 (2001)
18. Carrado, K.A., Macha, S.M., Tiede, D.M.: Effects of surface functionalization and organo-tailoring of synthetic layer silicates on the immobilization of cytochrome c. Chem. Mater. 16, 2559–2566 (2004)
19. Clarridge, A.G., Salomaa, K.: An improved cellular automata based algorithm for the 45-Convex hull problem. Journal of Cellular Automata 5, 107–112 (2010)
20. Field, R.J., Noyes, R.M.: Oscillations in chemical systems IV. Limit cycle behavior in a model of a real chemical reaction. J. Chem. Phys. 60, 1877–1884 (1974)
21. Goucher, A.P.: Gliders in cellular automata on Penrose tilings. J. Cellular Automata (2012) (in Press)
22. Greenberg, J., Hastings, S.: Spatial patterns for discrete models of diffusion in excitable media. SIAM J. Applied Math. 34, 515–523 (1978)
23. Griffen, D.T.: Silicate Crystal Chemistry. Oxford University Press (1992)
24. Janavicus, A.J., Storasta, J., Purlys, R., Mekys, A., Balakauskas, S., Norgela, Z.: Crystal lattice and carriers hall mobility relaxation processes in Si crystal irradiated by soft X-rays. Acta Physica Polonica 112, 55–67 (2007)
25. Lehmann, T., Wolff, T., Hamel, C., Veit, P., Garke, B., Seidel-Morgenstern, A.: Physico-chemical characterization of Ni/MCM-41 synthesized by a template ion exchange approach. Microporous and Mesoporous Materials 151, 113–125 (2012)
26. Liebau, F.: Structural Chemistry of Silicates: Structure, Bonding, and Classification. Springer (1985)
27. Law, M.E., Gilmer, G.H., Jaraíz, M.: Simulation of defects and diffusion phenomena in silicon. MRS Bulletin, 46–51 (June 2000)
28. Margenstern, M.: New Tools for Cellular Automata in the Hyperbolic Plane. J. UCS 6(12), 1226–1252 (2000)
29. Margenstern, M.: A universal cellular automaton on the heptagrid of the hyperbolic plane with four states. Theor. Comput. Sci. 412(1-2), 33–56 (2011)
30. Margenstern, M.: Universal Cellular Automata with Two States in the Hyperbolic Plane. J. Cellular Automata 7(3), 259–284 (2012)
31. Margenstern, M.: Universality and the Halting Problem for Cellular Automata in Hyperbolic Spaces: The Side of the Halting Problem. In: Durand-Lose, J., Jonoska, N. (eds.) UCNC 2012. LNCS, vol. 7445, pp. 12–33. Springer, Heidelberg (2012)
32. Margenstern, M.: Small Universal Cellular Automata in Hyperbolic Spaces: A Collection of Jewels. Springer (2013)
33. McDonald, A., Scott, B., Villemure, G.: Hydrothermal preparation of nanotubular particles of a 1:1 nickel phyllosilicate. Microporous and Mesoporous Materials 120, 263–266 (2009)
34. Monnier, A., Schuth, F., Huo, Q., Kumar, D., Margolese, D., Maxwell, R.S., Stucky, G.D., Krishnamurty, M., Petroff, P., Firouzi, A., Janicke, M., Chmelka, B.F.: Cooperative formation of inorganic-organic interfaces in the synthesis of silicate mesostructures. Science 261, 1299–1303 (1993)

35. Owens, N., Stepney, S.: Investigations of Game of Life cellular automata rules on Penrose tilings: Lifetime, ash, and oscillator Statistics. J. Cellular Automata 5, 207–225 (2010)
36. Pauling, L.: The structure of the micas and related minerals. Proc. Natl. Acad. Sci. U.S.A. 16, 123–129 (1930)
37. Pizzagalli, L., Godet, J., Guenole, J., Brochard, S.: Dislocation cores in silicon: new aspects from numerical simulations. Journal of Physics: Conference Series 281, 012002 (2011)
38. Richardson, I.G.: The calcium silicate hydrates. Cement and Concrete Research 38, 137–158 (2008)
39. Sokolski, M.M.: Structure and kinetics of defects in silicon. NASA TN D-4154, Washington (1967)
40. Specht, K.M., Jackson, M., Sunkel, B., Boucher, M.A.: Synthesis of a functionalized sheet silicate derived from apophyllite and further modification by hydrosilylation. Applied Clay Science 47, 212–216 (2010)
41. Suh, W.H., Suslick, K.S., Stucky, G.D., Suh, Y.-H.: Nanotechnology, nanotoxicology, and neuroscience. Progress in Neurobiology 87, 133–170 (2009)
42. Tanimura, K., Tanaka, T., Itoh, N.: Creation of quasistable lattice defects by electronic excitation in SiO_2. Phys. Rev. Lett. 51, 423–426 (1983)
43. Torbey, S., Akl, S.G.: An exact solution to the two-dimensional arbitrary-threshold density classification problem. Journal of Cellular Automata 4, 225–235 (2009)
44. Velichko, O.I., Dobrushkin, V.A., Muchynski, A.N., Tsurko, V.A., Zhuk, V.A.: Simulation of coupled diffusion of impurity atoms and point defects under nonequilibrium conditions in local domain. J. Comput Physics 178, 196–209 (2002)
45. Watkins, G.D.: Lattice vacancies and interstitials in silicon. Proc. of the US-ROC Solid State Physics Seminar. Chinese, J. Physics 15, 92–102 (1977)
46. Watkins, G.D.E.: studies of lattice defects in semiconductors. In: Henderson, B., Hughes, A.E. (eds.) Defects and Their Structure in Non-metallic Solids, p. 203. Plenum Press, New York (1976)

Chapter 17
DC Programming and DCA for Challenging Problems in Bioinformatics and Computational Biology

Le Thi Hoai An

Abstract. Nonconvex optimization is a powerful tool in bioinformatics and computational biology. As an innovative approach to nonconvex programming, Difference of Convex functions (DC) programming and DC Algorithms (DCA) have proved to be efficient for several problems in computational biology. The objective of this chapter is to show that grand challenge problems in bioinformatics and computational biology can be modeled as DC programs and solved by DCA based algorithms. We offer the community of researchers in computational biology promising approaches in a unified DC programming framework to tackle challenging biological problems such as Multiple Alignment of Sequence (MSA), Molecular conformation and Phylogenetic tree reconstruction.

17.1 Introduction

Computational biology is an important part of developing emerging technologies for the field of biology. It is used to help sequence the human genome, create accurate models of the human brain, and assist in modeling biological systems. To know a biological system, we want to get an insight in its evolution, structure, and function, in order to explain ultimately adaptation, diversity, and complexity of the system. Developing mathematical and computational approaches to analyze these and other properties of living systems is the ultimate grand challenge for bioinformatics and computational biology. Optimization plays a key role in computational biology since several issues of this domain are related to optimization problems. During the last two decades, an increasing amount of effort has been put into nonconvex optimization to deal with many difficult biological problems. The absence of

Le Thi Hoai An
Laboratory of Theoretical and Applied Computer Science (LITA EA 3097)
University of Lorraine,
Ile du Saulcy, 57045 Metz, France
e-mail: hoai-an.le-thi@univ-lorraine.fr

A. Adamatzky (ed.), *Automata, Universality, Computation,*
Emergence, Complexity and Computation 12, DOI: 10.1007/978-3-319-09039-9_17

convexity creates a source of difficulties of all kinds, in particular the distinction between the local and global minima, the non-existence of verifiable characterizations of global solutions, ... That causes all the computational complexity while passing from convex to nonconvex programming. Finding a global solution of a non-convex program, especially in the large-scale, is a graal for optimizers.

A variety of nonconvex optimization techniques have been recently developed by researchers in optimization. Generally speaking, there are two different but complementary approaches: Global approaches such as Cutting Plane (CP), Branch and Bound (B&B), Branch and Cut (B&C) can guarantee the globality of the solutions but they are very expensive, in particular for large scale setting; and Local approaches, on the contrary, are much faster while only local minima are available. Many current approaches are not generally effective for practical large-scale problems. Finding efficient algorithms that realize a compromise to overcome these drawbacks is a challenge of nonconvex programming. Such algorithms must exploit domain-specific structures of the problems being considered.

As an innovative approach to nonconvex programming, Difference of Convex functions (DC) programming and DC Algorithms (DCA) are increasingly used by researchers in various domains. In this chapter, within the four grand challenges in bioinformatics and computational biology ([61]), namely

- Protein structure prediction;
- Homology searches;
- Multiple alignment and phylogeny construction;
- Genomic sequence analysis and gene-finding;

we focus on three challenging classes of problems — Multiple Alignment of Sequence (MSA), Molecular conformation and Phylogenetic tree reconstruction. We show that these problems can be modeled as DC programs and solved by DCA based algorithms.

The chapter is organized as follows. In the next section we give a brief introduction of DC programming and DCA. The solution methods of each class of problem are discussed in Sections 17.3, 17.4 and 17.5. Finally Section17.6 concludes the chapter.

17.2 A Brief Introduction to DC Programming and DCA

DC programming, constitute the backbone of nonconvex programming and global optimization, is an extension of convex programming which is sufficiently large to cover almost all nonconvex optimization problems, but not too to still allow using the arsenal of powerful tools in convex analysis and convex optimization. DC programming and DCA were introduced by Pham Dinh Tao in their preliminary form in 1985. These theoretical and algorithmic tools are extensively developed by Le Thi Hoai An and Pham Dinh Tao since 1993 to become now classic and increasingly popular (see e.g. the list of reference in [28]).

17.2.1 DC Programming: Basic Notations and Background

We are working with the space $\mathbf{X} = \mathbb{R}^n$ which is equipped with the canonical inner product $\langle \cdot, \cdot \rangle$ and the corresponding Euclidean norm $\| \cdot \|$, thus the dual space $\mathbf{Y}$ of $\mathbf{X}$ can be identified with $\mathbf{X}$ itself. We follow [20], [59] for definitions of usual tools in modern convex analysis where functions could take the infinite values $+\infty$. A function $\theta : \mathbf{X} \to \mathbb{R} \cup \{+\infty\}$ is said to be proper if it is not identically equal to $+\infty$. The effective domain of θ, denoted by dom θ, is

$$\text{dom}\ \theta = \{x \in \mathbf{X} : \theta(x) < +\infty\}.$$

The indicator function χ_C of a nonempty closed convex C set is defined by $\chi_C(x) = 0$ if $x \in C, +\infty$ otherwise. The set of all lower semicontinuous proper convex functions on X is denoted by $\Gamma_0(\mathbf{X})$. Let $\theta \in \Gamma_0(\mathbf{X})$, then the conjugate function of θ, denoted θ^*, is defined by

$$\theta^*(y) = \sup\{\langle x, y\rangle - \theta(x) : \ x \in \mathbf{X}\}.$$

We have $\theta^* \in \Gamma_0(\mathbf{Y})$ and $\theta^{**} = \theta$.
Nonsmooth convex functions are handled using the concept of subdifferentials. For $\theta \in \Gamma_0(\mathbf{X})$ and $x_0 \in$ dom θ, $\partial\theta(x_0)$ denotes the subdifferential of θ at x_0, and is defined by

$$\partial\theta(x_0) := \{y \in \mathbf{Y} : \theta(x) \geq \theta(x_0) + \langle x - x_0, y\rangle, \ \forall x \in \mathbf{X}\}. \tag{17.1}$$

Each $y \in \partial\theta(x_0)$ is called a subgradient of θ at x_0. The subdifferential $\partial\theta(x_0)$ is a closed convex set in $\mathbf{Y}$. It generalizes the derivative in the sense that θ is differentiable at x_0 if and only if $\partial\theta(x_0) \equiv \{\nabla\theta(x_0)\}$. Recall the well-known properties related to subdifferential calculus of $\theta \in \Gamma_0(\mathbf{X})$:

$$y_0 \in \partial\theta(x_0) \Leftrightarrow x_0 \in \partial\theta^*(y_0) \Leftrightarrow \langle x_0, y_0\rangle = \theta(x_0) + \theta^*(y_0); \tag{17.2}$$

$$\partial\theta^*(y_0) = \arg\min\{\theta(x) - \langle x, y_0\rangle : x \in \mathbf{X}\}. \tag{17.3}$$

A function $\theta \in \Gamma_0(\mathbf{X})$ is said to be polyhedral convex if

$$\theta(x) = \max\{\langle a_i, x\rangle - \beta_i : i = 1, ..., m\} + \chi_C(x) \ \forall x \in \mathbf{X},$$

where C is a nonempty polyhedral convex set in $\mathbf{X}$.

A DC program is of the form

$$(P_{dc}) \ \ \alpha = \inf\{f(x) := g(x) - h(x) : \ x \in \mathbf{X}\}, \tag{17.4}$$

with $g,h \in \Gamma_0(\mathbf{X})$. Such a function f is called a DC function, and $g-h$, a DC decomposition of f, while the convex functions g and h are DC components of f. In (P_{dc}) the nonconvexity comes from the concavity of the function $-h$ (except the case h is affine since (P_{dc}) then is a convex program).
It should be noted that a convex constrained DC program of the form

$$\alpha = \inf\{f(x) := g(x) - h(x) : \ x \in C\}, \tag{17.5}$$

can be expressed in the form (17.4) by using the indicator function on C, that is

$$\inf\{f(x) := g(x) - h(x) : \ x \in C\} = \inf\{\varphi(x) - h(x) \ : \ x \in \mathbf{X}\}, \text{ with } \varphi := g + \chi_C.$$

Hence, throughout this paper, DC program of the form (17.4) is referred to as "standard DC program".

Polyhedral DC programs (P_{dc}) (i.e., when g or h are polyhedral convex) play a key role in nonconvex programming (see [36, 48, 49] and references therein), and enjoy interesting properties related to local optimality and DCA's convergence.

The DC duality is based on the conjugate functions and the fundamental characterization of a convex function $\theta \in \Gamma_0(\mathbf{X})$ as *the pointwise supremum of a collection of affine minorants:*

$$\theta(x) = \sup\{\langle x,y\rangle - \theta^*(y) : y \in \mathbf{Y}\}, \quad \forall x \in \mathbf{X}. \tag{17.6}$$

That associates the primal DC program (17.4) (P_{dc}) with its dual DC program (D_{dc}) defined by

$$(D_{dc}) \quad \alpha = \inf\{h^*(y) - g^*(y) : y \in \mathbf{Y}\}, \tag{17.7}$$

and investigates their mutual relations. We observe the perfect symmetry between primal and dual DC programs: the dual to (D_{dc}) is exactly (P_{dc}).

It is worth noting the wealth of the vector space $DC(X) = \Gamma_0(\mathbf{X}) - \Gamma_0(X)$ spanned by the "convex cone" $\Gamma_0(\mathbf{X})$ ([48,49]): it contains most realistic objective functions and is closed under operations usually considered in optimization.

The complexity of DC programs resides, of course, in the lack of verifiable globality conditions. Lets recall the general local optimality conditions in DC programming (subdifferential's inclusion): if x^* is a local solution of (P_{dc}) then

$$\emptyset \neq \partial h(x^*) \subset \partial g(x^*). \tag{17.8}$$

The condition (17.8) is also sufficient (for local optimality) in many important classes of DC programs (see [48, 49]).

A point x^* is said to be *a critical point* of $g-h$ (or generalized KKT point for (P_{dc})) if

$$\partial h(x^*) \cap \partial g(x^*) \neq \emptyset. \tag{17.9}$$

Note that, by symmetry, the dual part of (17.8) and (17.9) are trivial.

17.2.2 DCA's Philosophy

DCA is a continuous primal dual subgradient approach based on local optimality and duality in DC programming for solving standard DC programs (P_{dc}).

The key idea behind DCA is to replace in (P_{dc}), at the current point x^k, the second component h with its affine minorant defined by

$$h_k(x) := h(x^k) + \langle x - x^k, y^k \rangle, \ \ y^k \in \partial h(x^k)$$

to give birth to the primal convex program of the form

$$(P_k) \quad \inf\{g(x) - h_k(x) : \ x \in X\} \Longleftrightarrow \inf\{g(x) - \langle x, y^k \rangle \ : x \in \mathbf{X}\}$$

whose solution set is $\partial g^*(y^k)$. The next iterate x^{k+1} is taken in $\partial g^*(y^k)$.

Dually, a solution x^{k+1} of (P_k) is then used to define the dual convex program (D_{k+1}) obtained from (D_{dc}) by replacing g^* with its affine minorant

$$(g^*)_k(y) := g^*(y^k) + \langle y - y^k, x^{k+1} \rangle$$

to obtain the convex program

$$(D_{k+1}) \ \inf\{h^*(y) - [g^*(y^k) + \langle y - y^k, x^{k+1} \rangle] : y \in Y\} \Leftrightarrow \inf\{h^*(y) - \langle y, x^{k+1} \rangle \ : y \in \mathbf{Y}\}$$

whose solution set is $\partial h(x^{k+1})$. The next iterate y^{k+1} is chosen in $\partial h(x^{k+1})$.

DCA scheme

Initialization: Let $x^0 \in \mathrm{IR}^n$ be a guess, $k \leftarrow 0$.

Repeat

- Calculate $y^k \in \partial h(x^k)$
- Calculate $x^{k+1} \in \partial g^*(y^k)$, which is equivalent to

$$x^{k+1} \in arg \min\{g(x) - \langle x, y^k \rangle : x \in \mathrm{IR}^n\} \quad (P_k)$$

- $k \leftarrow k+1$

Until convergence of $\{x^k\}$.

17.2.3 DCA's Convergence Properties

Convergence properties of DCA and its theoretical basis can be found in ([36, 48, 49]). For instance it is important to mention that

i) DCA is a descent method without linesearch but with global convergence: the sequences $\{g(x^k) - h(x^k)\}$ and $\{h^*(y^k) - g^*(y^k)\}$ are decreasing.

ii) If the optimal value α of problem (P_{dc}) is finite and the infinite sequences $\{x^k\}$ and $\{y^k\}$ are bounded, then every limit point x^* (resp. y^*) of the sequence $\{x^k\}$

(resp. $\{y^k\}$) is a critical point of $g - h$ (resp. $h^* - g^*$), i.e. $\partial h(x^*) \cap \partial g(x^*) \neq \emptyset$ (resp. $\partial h^*(y^*) \cap \partial g^*(y^*) \neq \emptyset$).

iii) DCA has a *linear convergence* for DC programs.

iv) DCA has a finite convergence for polyhedral DC programs.

For a complete study of DC programming and DCA the reader is referred to [36, 48–50] and the references therein.

17.2.4 Key Properties of DCA

Without going into details, let us mention the key properties of DCA.

a. *Flexibility*: the construction of DCA is based on g and h but not on f itself, and there are as many DCA as there are DC decompositions. This is a crucial fact in DC programming. It is important to study various equivalent DC forms of a DC program, because each DC function f *has infinitely many DC decompositions which have crucial implications for the qualities* (speed of convergence, robustness, efficiency, globality of computed solutions,...) of DCA.

b. DCA is a *descent method without linesearch*, with global convergence (i.e. DCA converges from an arbitrary initial point).

c. *Return from DC programming to convex programming*: DCA consists in an iterative approximation of a DC program by a sequence of convex programs that will be solved by appropriate convex optimization algorithms. This property is called SCA (Successive Convex Approximation) in some recent works.

d. *Versatility*: With suitable DC decompositions *DCA generates most standard algorithms in convex and nonconvex optimization.* Hence DCA offers a wide framework for solving convex/nonconvex optimization problems. In particular DCA is a global approach to convex programming, i.e., it converges to optimal solutions of convex programs reformulated as DC programs. Consequently, it can be used to build efficient customized algorithms for solving convex programs generated by DCA.

To the best of our knowledge, DCA is actually one of the rare algorithms for nonsmooth nonconvex programming which allow to solve large-scale DC programs. DCA was successfully applied to a lot of different and various nonconvex optimization problems to which it quite often gave global solutions and proved to be more robust and more efficient than related standard methods (see the list of references in [28]).

We now show how to solve three challenging classes of biological problems by DC programming and DCA.

17.3 Multiple Sequence Alignment

The construction of biologically plausible alignment for given families of DNA or protein sequences is one of the major problems in computational molecular biology. Consequently, the sequence alignment plays an important role and is a very active area of research. The sequence of a DNA molecule can be modeled as a string over a 4-character alphabet. The proteins are chains of amino acids which can be represented as strings via one or three letter codes. This is very useful, for example, in designing experiments to test and modify the function of specific proteins, in predicting the function and structure of proteins, and in identifying new members of protein families (see [18]).

The multiple sequence alignment problem (MSA in short) can be described as follows. Let Σ be a finite alphabet and let $\Sigma^* = \Sigma \cup \{-\}$, where "$-$" is a symbol to represent gaps in strings. Given a set $S = \{S_1, S_2, \ldots, S_k\}$ of finite strings over the alphabet Σ, where each string S_i, $i = 1, \ldots, k$, consists of characters $s_{i,1}, \ldots, s_{i,l_i} \in \Sigma$. A set $S^* = \{S_1^*, S_2^*, \ldots, S_k^*\}$ of strings over the alphabet Σ^* is called an alignment of S if the following tree properties hold:

i) all strings in S^* have the same length l;
ii) ignoring gaps, S_i^* is identical with $S_i, \forall i = 1, \ldots, k$;

An alignment in which each sequence S_i^* has length l can be associated with an array of k rows and l columns, where row i corresponds to S_i^*. Two characters of distinct sequences in S are said to be *aligned* under S^* if they are placed into the same column of the alignment array (corresponding to S^*). Depending on the measure of the quality of alignment, the multiple sequence alignment problem aims to find the 'best' alignment in some sense. For example when two sequences are aligned, in each column depending on the characters aligned we have a score and the scores are added over columns to measure the value of the alignment. An alignment that maximizes the value is picked up. This idea is extended to multiple sequence alignment as well.

Notice that adding a column of '-' to an alignment increases its length by one. But it should not add value as the newly added column does not contain any character from the original alphabet. However if it does we could add more columns of this kind increasing the value of the alignment without a limit. Therefore, if an alignment satisfies iii) then its length is not larger than $\sum_{i=1}^{k} l_i$.

iii) no column of an alignment has "-" in all its rows. That is, there exits an i_r such that $s_{i_r,r}^* \neq$"-" for each $r, 1 \leq r \leq l$.

Computational biology provides many challenging combinatorial optimization problems, and multiple sequence alignment (MSA) problem is one of them. It has been shown in Kececioglu [26] that the MSA problem can be transformed into the NP-complete Maximum Weight Trace problem (MWT in short), as a natural formulation of merging partial alignments to form multiple alignments. The goal is to compute an alignment S^* whose trace has maximum weight. It is based on the concept of the alignment graph $G = (V, E)$ whose nodes V correspond to the characters of the k input sequences and are labeled correspondingly, the edge set E connects pairs of characters that one would like to have aligned in an alignment of the input strings.

Several methods have been developed to the MSA problem. Dynamic programming has been applied to solve this problem but the curse of dimensionality prevented its use to solve problems with large number of sequences or if the sequence lengths grew in size [45]. Recently [17] has given quadratic integer programming formulations of some problems in computational biology which includes the MSA problem. Other efficient approaches to similar problems are found in [56] and [57] .

Kececioglu [25] introduced and studied the Maximum Weight Trace (MWT) formulation of the MSA problem. Reinert et al. [58] consider the trace polytope and give an integer programming formulation of the MWT. They study the trace polytope and show some of the separation problems corresponding to some class of facet defining inequalities can be solved in polynomial time. They use polyhedral combinatorics to develop a branch-and-cut algorithm for solving the problem. Their computational experiments show that the problem size both in terms of number of sequences and sequence length could be increased beyond what was feasible with dynamic programming approaches. Polyhedral approach to sequence alignment problems like generalized maximum trace problem, RNA sequence alignment problem are studied by Kececioglu et al. [27], Lenhof [42].

Le Thi et al. [40] proposed a continuous optimization approach based on DC (Difference of Convex functions) programming and DCA (DC Algorithm) to the MSA using the 0-1 linear formulation of the MWT. The approach is inexpensive because it amounts to solve a few linear programs and it converges after a finite number of iterations to an integer solution while it works in a continuous domain. Numerical simulation experiments show the superiority of this approach using DCA with respect to other methods.

Prestwich et al. [52] give a pseudo-boolean local search algorithm and compare its performance with state-of-the art algorithms for solving MSA problem. However they report longer execution time though the quality is better in some bench mark cases. They also have a polynomial size $0-1$ formulation of the MWT problem.

In [2], Arthanari and Le Thi proposed a new integer quadratic programming formulation, which directly addresses the MSA problem. The number of constraints and variables in the problem are only of the order of kL^2, where, k is the number of sequences and L is the total length of the sequences, that is, $L = \sum_{i=1}^{k} l_i$, where l_i is the length of sequence i. Based on the new quadratic formulation of the MSA problem, an equivalent linear constrained $0-1$ quadratic programming problem is introduced. Also a linear $0-1$ formulation of the MWT problem is given. This formulation has in addition $(2L+1)|E|$ constraints, where $|E|$ is the number of edges in the alignment graph. And thus one has a compact formulation of the MWT problem.

17.3.1 Multiple Sequence Alignment Formulated as the MWT Problem

The notion of trace of two sequences was generalized to multiple sequences by Kececioglu [26]. This leads us to the discussion on the maximum weight trace (MWT) problem.

Definition 1. (Alignment Graph) Let Σ be a finite alphabet. Given a set $S = \{S_1, S_2, \ldots, S_k\}$ of finite strings over the alphabet Σ, where each string S_i, $i = 1, \ldots, k$, consists of characters $s_{i,1}, \ldots, s_{i,l_i} \in \Sigma$. Consider the graph $G = (V, E)$, where V is the vertex set and E is the edge set, each edge $s_{i,j}$ represents pairs of characters (base symbols) that one would like to have aligned in an alignment of the input sequences. We say that an edge of the alignment graph is *realized* by an alignment if the endpoints of the edge are placed in the same column of the corresponding alignment array.

In the alignment graph one may associate $w(e)$, a nonnegative weight attached to each edge, and the trace of an alignment is defined by

Definition 2. (Trace of an alignment) Given an alignment $\hat{S}$ corresponding to a set of sequences S. We call the subset of edges, $T \subset E$, realized by $\hat{S}$ as the *trace* of the alignment, and $w(T) = \sum_{e \in E} w(e)$ as the weight of the trace.

The multiple sequence alignment problem can now be stated as a maximum weight trace problem (MWT): *given a set $S = \{S_1, S_2, \ldots, S_k\}$ of finite strings over the finite alphabet Σ, and an alignment graph $G = (V, E)$, with nonnegative edge weights, find an alignment such that its trace has maximum weight.*

While MWT is known to be *NP*-complete [25], polyhedral combinatorics has been successfully used in MWT by Reinert et al. [58]. We shall give some graph preliminaries and then present the binary integer formulation of the MWT problem as developed by them.

17.3.1.1 Graph Preliminaries

We follow standard graph theoretic notations, regarding a graph. A mixed graph is a tuple $G = (V, E, A)$ where V is a set of vertices, E is a set of edges and A is a set of arcs. We use $e = \{v_i, v_j\}$ if e is an edge and we use $e = (v_i, v_j)$ to denote an arc, specifying the direction that it is from v_i to v_j. A *path* in a mixed graph is an alternating sequence $v_1, e_1, v_2, e_2, \ldots, v_k$ of vertices and arcs or edges such that in v_i, e_i, v_{i+1}, the consecutive vertices form the end points of the edges or arcs, for all $i, 1 \le i \le k$. A path is called a cycle if the first and the last vertex on the path is the same. We use the adjective 'mixed' to denote paths or cycles which contain at least one each of an arc from A and an edge from E. The number of edges in a mixed path is called its *size*. In addition we use the term *length* of a path (cycle) to denote the number of edges and arcs it has.

Definition 3. (Extended Alignment Graph) [58]

Given an alignment graph, $G = (V, E)$ we consider the mixed graph (V, E, H) by adding a set of directed arcs

$$H = \{(s_{i,j}, s_{i,j+1}) | 1 \le i \le k, 1 \le j \le l_i\},$$

where $s_{i,j}$ is the vertex in V corresponding to j^{th} base symbol in sequence S_i.

We call this mixed graph the extended alignment graph (EAG).

In addition in [58] we have the definition of a *critical* mixed cycle.

Definition 4. (Critical Mixed Cycle) Given the extended alignment graph, $G = (V,E,H)$, we call a mixed cycle R critical if for all $i, 1 \leq i \leq k$, all vertices in $R \cap S_i$ occur consecutively in R.

The important characterization of traces using critical mixed cycles is given as Theorem 1 in [58] which is stated as Theorem 1.

Theorem 1. *Let $G = (V,E,H)$ be an EAG, let $T \subset E$ and let $G' = (V,T,H)$ be the EAG induced by T. Then T is a trace $\Longleftrightarrow$ there is no critical mixed cycle in G'.*

Using this theorem we are in a position to state the integer programming formulation in [58].

17.3.1.2 Binary Linear Programming Formulation of the MWT Problem

Let x_e be the indicator variable of an $e \in E$ which is in the trace T, of an alignment, $\hat{S}$. That is, $x_e = 1$ if e in T and 0 otherwise. Denote by $|\cdot|$ the cardinality of a set.

Then the MWT problem can be stated as:

$$\begin{aligned} \textbf{(MWT)} \quad & \max \sum_{e \in E} w_e x_e \\ s.t \quad & \sum_{e \in Z \cap E} x_e \leq |Z \cap E| - 1, \\ & \forall \text{ critical mixed cycles } Z \text{ in } G \\ & x_e \in \{0,1\}, \ \forall\, e \in E. \end{aligned} \tag{17.10}$$

The formulation (17.10) has exponentially many constraints and belongs to the class of Linear constrained Binary Linear programming problems (Binary Linear programming problems in short). In [2] a compact Binary linear programming formulation of the MWT problem has been proposed which has polynomially many constraints and variables.

17.3.1.3 On Input Alignment Graph

The edges of the alignment graph and their weights are crucial for the success of any method applied to the MWT formulation of the MSA. So far, finding appropriate edges and weights has received only little attention. The edges of the alignment graph can be obtained from pairwise alignments as well as from other methods like local alignment algorithms. Using the dynamic programming algorithm of Myers and Miller ([44]) to align each pair of sequences, we compute $\binom{n}{2} = n \cdot (n-1)/2$ pairwise alignment and store them in the pairwise alignment graph. The time to compute the input graph and memory required is $O(l^3 n^2)$ with $l = \max_{i=1,\dots,n} l_i$ ([44]). The edge weight w_e can be determined in various ways:

- Sequence identity score (SIS): For every pair of aligned sequences, the number of character matches t_1 and that of character mismatches t_0 are computed (comparisons with gaps remain unconsidered), then the percent sequence identity score is equal to: $SIS = \frac{t_1}{t_1+t_0} \cdot 100$; which gives a result between 0% and 100%.
- 1-0 score (1-0S): Any edge, which connects the same characters, receives the value 1 (100%), otherwise its weight is set to 0 (0%) ([55]).
- CLUSTAL W's similarity matrices with window filtering (CSM) ([54]).
- T-Coffee ([46]): Once the sequence alignments have been calculated for all pairs of input sequences, we construct the so-called library. A library for a pairwise alignment of two sequence consists of the set of all realized edges together with a weighting of each edge ([54]).

17.3.2 Integer Quadratic Formulations of the MSA Problem

Recently Greenberg [17] has suggested some integer quadratic programming models in computational biology, and among them one of the models is for the MSA problem. This model uses a scoring function that measures the propensity for a character in a sequence to align with a character in another sequence. In addition a gap penalty function, and two sets of variables to indicate the opening and extending of gaps in sequences are used in this model. The model also has additional constraints involving theses variables. In [2] a so called True Sequence Alignment (TSA) formulation having fewer constraints and variables has been proposed.

Let any alignment be given by an array having one row for each of the k sequences and l columns. We call (i,r) a cell. Each cell has either a "-" or a $s_{i,j}$ for some j. And in any row the basic symbols in the sequence S_i, $s_{i,j}$'s appear in cells (i, r_j), such that $1 \leq r_1 < r_2, \dots, < r_{l_i}$, respectively. And we have an edge $(s_{i,j}, s_{p,q})$ of the alignment graph realized if the end points of the edge appear in the same column r, that is, $s_{i,j}$ is in cell (i,r) and $s_{p,q}$ is in cell (p,r) for some r. We are interested in maximizing the sum of the weights of the edges realized by an alignment.

17.3.2.1 True Sequence Alignment Formulation

Let a_{ijr} be defined such that $a_{ijr} = 1$ if $s_{i,j}$ occupies the cell (i,r) of the alignment, otherwise $a_{ijr} = 0$, for all i,j,r: $1 \leq i \leq k, 1 \leq j \leq l_i$ and $j \leq r \leq l$. We next give the quadratic integer formulation of the MSA. Since the formulation gives a $1-1$ correspondence between alignments and $0-1$ solutions to the formulation we call this formulation True Sequence Alignment (TSA) formulation.

(TSA Formulation)

$$maximize_{l,a_{ijr}} \quad \sum_{(s_{i,j},s_{p,q})\in E} \sum_{r=1}^{l} w((s_{i,j},s_{p,q}))a_{ijr}a_{pqr}$$

$$\max_{i=1\ldots k}(l_i) \leq l \leq \sum_{i=1}^{k} l_i. \tag{17.11}$$

$$l \text{ integer.} \tag{17.12}$$

$$\sum_{j=1}^{l_i} a_{ijr} \leq 1 \text{ for all } i,r. \tag{17.13}$$

$$\sum_{r=1}^{l} a_{ijr} = 1, \text{ for all } i,j. \tag{17.14}$$

$$a_{ijr} - \sum_{s=j-1}^{r-1} a_{i(j-1)s} \leq 0, \text{ for all } i,r, \text{ and } 2 \leq j. \tag{17.15}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{l_i} a_{ijr} \geq 1, \text{ for all } r. \tag{17.16}$$

$$a_{ijr} \in \{0,1\} \text{ for all } i,j,r. \tag{17.17}$$

The $1-1$ correspondence with alignments and the $0-1$ solutions to the TSA formulation is formally stated in the following theorem ([2]).

Theorem 2. *Let Σ be a finite alphabet and let $\hat{\Sigma} = \Sigma \cup \{-\}$, where "$-$" is a symbol to represent gaps in strings. Given a set $S = \{S_1, S_2, \ldots, S_k\}$ of finite strings over the alphabet Σ, every alignment of S, $\hat{S} = \{\hat{S_1}, \hat{S_2}, \ldots, \hat{S_k}\}$ of strings over the alphabet $\hat{\Sigma}$, corresponds to an integer solution to the TSA formulation and vice versa.*

17.3.2.2 Linear Constrained Binary Quadratic Programming Formulation

Basing on the TSA formulation Arthanari and Le Thi [2] developed an equivalent linear constrained $0-1$ quadratic programming (Binary Quadratic programming in short) formulation of the MSA problem. It has been shown in [2] that to find an optimal solution to the TSA formulation it is sufficient to solve *Problem*$[L]$ below, with $L = \sum_{i=1}^{k} l_i$.

(Problem [L])

$$\textit{maximize} \quad \sum_{(s_{i,j},s_{p,q})\in E} \sum_{r=1}^{L} w((s_{i,j},s_{p,q}))a_{ijr}a_{pqr}$$

$$\sum_{j=1}^{l_i} a_{ijr} \le 1 \text{ for all } i,r. \tag{17.18}$$

$$\sum_{r=1}^{L} a_{ijr} = 1, \text{ for all } i,j. \tag{17.19}$$

$$a_{ijr} - \sum_{s=j-1}^{r-1} a_{i(j-1)s} \le 0, \text{ for all } i,r, \text{ and } 2 \le j. \tag{17.20}$$

$$a_{ijr} \in \{0,1\} \text{ for all } i,j,r. \tag{17.21}$$

The following theorem connects traces with the $0-1$ solutions of the TSA formulation.

Theorem 3. *Let $G=(V,E)$ be the graph, where V is the set of vertices corresponding to the base symbols $s_{i,j}$, in the given sequences $S_i, i=1,\ldots,k$. And an edge $e=(s_{i,j},s_{p,q})$ is in E if and only if $w(e)>0$. $T \subset E$ is a trace corresponding to an alignment if and only if there is a corresponding* $0-1$ *solution to the TSA formulation with the same weight of the trace, $w(T)$.*

17.3.3 Solution Methods by DC Programming and DCA

In terms of optimization, Binary Linear programming is a special case of Binary Quadratic programming. In this subsection we show how to apply DC programming and DCA on the common model of three problems — the MWT problem (17.10), the TSA formulation and *Problem*$[L]$, that is the Binary Quadratic programming problem of the form:

$$\textit{minimize } \tfrac{1}{2}x^T Cx + c^T x \tag{17.22}$$

$$\textit{subjectto} \quad Ax \le b, \tag{17.23}$$

$$x \in \{0,1\}^n \tag{17.24}$$

where C is an $(n\times n)$ symmetric matrix, $c,x \in \mathbb{R}^n$, A is an $(m\times n)$ matrix and $b \in \mathbb{R}^m$.

Binary quadratic programming ($0-1$ QP in short) problems form an important class of combinatorial optimization with application in many areas of science, technology, and business. They are known to be NP-hard [65]. Different approaches are developed to solve such problems (see e.g. [5], [1], [47], [32], [51]. Pardalos and Rodgers [47] discussed computational aspects of a branch and bound algorithm.

Adams and Sherali [1] proposed a tight linearization based algorithm for solving such problems. Le Thi and Pham Dinh [32] developed a continuous approach based on DCA for large scale problems of this kind. More recently, an efficient combined DCA and B&B using DC/SDP relaxation for globally solving binary quadratic programs has been developed in Pham et al. [51]. A recent survey of the approaches for solving $0-1$ QP problems appears in [8].

17.3.3.1 From Combinatorial Optimization to DC Programming: Reformulation

Based on exact penalty techniques in concave programming [30], the $0-1$ QP problem (17.22) can be equivalently reformulated as a DC program in the following way.

Let $D := \{x \in \mathbb{R}^n : Ax \leq b\}$ be a nonempty bounded polyhedral convex set in $\mathbb{R}^n$ and let $J := \{1,\ldots,n\}$ (note that if $J \subset \{1,\ldots,n\}$ then $0-1$ QP problems becomes Mixed $0-1$ QP problems, and the results in this subsection works also in this case). Let λ be a positive number such that $C - \lambda I$ is a semidefinite negative (s.d.n) matrix (here I stands for the identity matrix of order n). Since $x \in \{0,1\}^n$, $x_i = x_i^2$ and therefore the $0-1$ QP Problem (17.22) can be rewritten as

$$\min\left\{ f(x) := \frac{1}{2}x^T(C-\lambda I)x + (c + (\lambda/2)\mathbf{e})^T x : x \in D, x_i \in \{0,1\}, \forall i \in J \right\}, \tag{17.25}$$

where $\mathbf{e}$ denotes the vector of ones in $\mathbb{R}^n$. Obviously the function f is concave.

Let $K := \{x \in D : 0 \leq x_i \leq 1, \forall i \in J\}$ and define $p(x) = \frac{1}{2}\sum_{i\in J} x_i(1-x_i)$. Clearly, p is a concave function with nonnegative values on K and

$$\{x \in D, x_i \in \{0,1\}\} = \{x \in K : p(x) = 0\} = \{x \in K : p(x) \leq 0\}.$$

Hence (17.25) can be reformulated as (according to Theorem 4 below)

$$\min\{f(x) : x \in K, p(x) \leq 0\}$$

which is equivalent to, for any $t > t_o$

$$\min\{f(x) + tp(x) : x \in K\}. \tag{17.26}$$

This is a concave quadratic programming problem which is also NP-hard (but all the variables are continuous).

Theorem 4. *[30] Let K be a nonempty bounded polyhedral convex set in $\mathbb{R}^n$ and f, p be finite concave on K. Assume the feasible set of (P) be nonempty and p be nonnegative on K. Then there exists $t_o \geq 0$ such that for every $t > t_o$ the following problems have the same solution sets:*

$$(P_t) \quad \alpha(t) = \inf\{f(x) + tp(x) : x \in K\},$$

$$(P) \quad \alpha = \inf\{f(x) : x \in K, p(x) \leq 0\}.$$

Furthermore

(i) if the vertex set of K, denoted by $V(K)$, is contained in $\{x \in K : p(x) \leq 0\}$, then $t_o = 0$.

(ii) if $V(K)$ is not contained in $\{x \in K : p(x) \leq 0\}$, then $t_o \leq \frac{f(x^o)-\alpha(0)}{S}$ for every $x^o \in K$, $p(x^o) \leq 0$, where $S := \min\{p(x) : x \in V(K), p(x) > 0\}$.

17.3.3.2 A DCA Based Algorithm for Solving Continuous Concave Quadratic Programming Problem

According to the definition of f and p, the problem (17.26) can be expressed as

$$\min\left\{F_t(x) := \frac{1}{2}x^T(C-\bar{\lambda}I)x + \left(c + (\bar{\lambda}/2)\mathbf{e}\right)^T x : x \in K\right\} \tag{17.27}$$

with $\bar{\lambda} := \lambda + t$. A natural DC formulation of this problem is

$$\min\{F_t(x) := g(x) - h(x) : x \in \mathbb{R}^n\} \text{ with } g(x) := \chi_K(x) \text{ and } h(x) := -F_t(x). \tag{17.28}$$

With this DC formulation (P_t) is a polyhedral DC program because χ_K is a polyhedral convex function, and the general DCA scheme becomes:

$$y^k \in \partial h(x^k); \quad x^{k+1} \in \arg\min\left\{-\langle x, y^k\rangle : x \in K\right\}. \tag{17.29}$$

Besides the computation of

$$y^k := \nabla h(x^k) = (C - \bar{\lambda}I)x^k + c + (\bar{\lambda}/2)\mathbf{e},$$

the algorithm requires one linear program at each iteration. The convergence properties can be stated as follows:

Theorem 5. *i) DCA generates a finite sequence $x^1, \ldots, x^{k_*}$ contained in V(K) such that $f(x^{k+1}) + tp(x^{k+1}) \leq f(x^k) + tp(x^k)$, $p(x^{k+1}) \leq p(x^k)$ for each k , and x^{k_*} is local minimizer of (17.28).*
ii) Let $t > t_1 := \max\left\{\frac{f(x)-\alpha(0)}{S} : x \in V(K) \cap \{x \in K : p(x) = 0\}\right\}$. If at an iteration q one has $p(x^q) = 0$, then $p(x^k)$=0 and $f(x^{k+1}) \leq f(x^k)$ $\forall k \geq q$.

17.3.3.3 A Constraint Generation Algorithm Based on DCA

Let $\mathcal{G}$ be the extended mixed graph (V, E, H) with $H = \{(s_{i,j}, s_{i,j+1}) | 1 \leq i \leq k, 1 \leq j \leq l_i\}$, where $s_{i,j}$ is the vertex in V corresponding to j^{th} base symbol in sequence S_i. Since the number of mixed cycles in a graph is an exponential function of the number of vertices, it is very difficult, even impossible, to solve (MWT) when the number of vertices of $\mathcal{G}$ is large. To avoid enumerating all the critical mixed cycles in $\mathcal{G}$ (i.e. all constraints of (MWT)), a constraint generation procedure based on DCA, called DCAMSA, has been proposed in [40]. A similar procedure can also be developed to the TSA formulation and to $Problem[L]$.

17.4 Molecular Conformation

In recent years there has been a very active research in the molecular optimization, especially in the protein folding framework which is one of the most important problems in biophysical chemistry. Molecular optimization problems arise also in the study of clusters (molecular cluster problems) and of large, confined ionic systems in plasma physics. The determination of a molecular conformation can be tackled by either minimizing a potential energy function (if the molecular structure corresponds to the global minimizer of this function) or solving the distance geometry problem ([11], [19]) (when the molecular conformation is determined by distances between pairs of atoms in the molecule). Both methods are concerned with global optimization problems.

17.4.1 Molecular Conformation via Distance Geometry Problem

The *exact distance geometry problem* consists in finding positions $x^1,\dots,x^n$ of n points in $\mathbb{R}^p$ such that

$$\|x^i - x^j\| = \delta_{ij}, (i,j) \in \mathcal{S}, \tag{17.30}$$

where $\mathcal{S}$ is a subset of the point pairs, δ_{ij} with $(i,j) \in \mathcal{S}$ is the given distance between points i and j, and $\|\cdot\|$ denotes the Euclidean norm. Usually, a small subset of pairwise distances is known, i.e., $\mathcal{S}$ is small, and in practice, lower and upper bounds of the distances are given instead of the exact values. In the molecular conformation, the problem is considered in three-dimensional Euclidean space - one has to find positions $x^1,\dots,x^n$ of n atoms in $\mathbb{R}^3$ verifying (17.30). In general, it can be defined in arbitrary dimensions p.

It should be noted that, by the error in the theoretical or experimental data, there may not exist a solution to this problem, then an ε-optimal solution of (17.30), namely a configuration $x^1,\dots,x^n$ satisfying

$$|\, \|x^i - x^j\| - \delta_{ij} \,| \leq \varepsilon, (i,j) \in \mathcal{S}, \tag{17.31}$$

is useful. When only the lower and upper bounds of δ_{ij} are given, we are faced with the so-called *general distance geometry problem* which consists of finding a set of positions $x^1,\dots,x^n$ in $\mathbb{R}^p$ such that

$$l_{ij} \leq \|x^i - x^j\| \leq u_{ij}, (i,j) \in \mathcal{S}, \tag{17.32}$$

where l_{ij} and u_{ij} are lower and upper bounds of the distance constraints, respectively.

In Euclidean distance geometry problems, we must take into consideration the symmetry of both the subset $\mathcal{S}$ (i.e. $(i,j) \in \mathcal{S}$ implies $(j,i) \in \mathcal{S}$). For simplifying the presentation, let $\mathcal{S}^w$ be the set defined by

$$\mathcal{S}^w := \{(i,j) \in \mathcal{S} : i < j\}.$$

In the sequel $\mathcal{M}_{n,p}(\mathrm{I\!R})$ denotes the space of real matrices of order $n \times p$ and for $X \in \mathcal{M}_{n,p}(\mathrm{I\!R})$, X_i (resp. X^i) is its i^{th} row (resp. i^{th} column). By identifying a set of positions $x^1, \ldots, x^n$ with the matrix X (i.e. $(X^T)^i = (X_i)^T = x^i$ for $i = 1, ..., n$), we can advantageously express the exact and/or general distance geometry problems in the matrix space $\mathcal{M}_{n,p}(\mathrm{I\!R})$:

$$0 = \min\left\{ \sigma(X) := \frac{1}{2}\sum_{\mathcal{S}^w} w_{ij}\theta_{ij}(X) : \quad X \in \mathcal{M}_{n,p}(\mathrm{I\!R}) \right\}, \tag{17.33}$$

where $w_{ij} > 0$ for $i \neq j$ and $w_{ii} = 0$ for all i. The pairwise potential $\theta_{ij} : \mathcal{M}_{n,p}(\mathrm{I\!R}) \longrightarrow \mathrm{I\!R}$ is defined for problem (17.30) by either

$$\theta_{ij}(X) = \left(\delta_{i,j}^2 - \|X_i^T - X_j^T\|^2\right)^2 \tag{17.34}$$

or

$$\theta_{ij}(X) = \left(\delta_{ij} - \|X_i^T - X_j^T\|\right)^2, \tag{17.35}$$

and for problem (17.32) by

$$\theta_{ij}(X) = \min{}^2\left\{ \frac{\|X_i^T - X_j^T\|^2 - l_{ij}^2}{l_{ij}^2}, 0 \right\} + \max{}^2\left\{ \frac{\|X_i^T - X_j^T\|^2 - u_{ij}^2}{u_{ij}^2}, 0 \right\}. \tag{17.36}$$

When all pairwise distances are available and a solution exists, the exact distance geometry problem (17.30) can be solved by a polynomial time algorithm (Blumenthal [6], Crippen and Havel [11]). However, in practice one knows only a subset of the distances, and it is well known (Saxe [60]) that $p-$ dimensional distance geometry problems are strongly NP-complete with $p = 1$ and strongly NP-hard for all $p > 1$. The visible sources of difficulties of these problems are

- The question of the existence of a solution,
- The non-uniqueness of solutions,
- The presence of a large number of local minimizers,
- The large scale of problems that arise in practice.

Several methods have been proposed for solving the distance geometry problems (17.30) and/or (17.32). Practically, one is often faced with very large scale problems for which global methods like branch and bound are expensive. So it is worth investigating local approaches conjointly with global techniques such as smoothing. An advantage of local approaches for these problems is that the optimal value is a priori known, therefore it will be easy to recognize whether the algorithm provides a global minimizer. For a complete list of references the reader is referred to [39]. Here we focus on DC programming and DCA based algorithms. DC programming and DCA were extensively studied for solving both exact distance geometry problem ([33]) and general distance geometry problem ([31,34,35,39]). It has been shown in these works that the DCA can be well exploited to obtain efficient algorithms for both exact and general large scale distance geometry problems. Key issues of DCA were carefully studied and several techniques were proposed to exploit the nice effect of DC decompositions and of starting points.

In [33] the exact distance geometry problem was considered in the form

$$\text{(EDP)} \qquad 0 = \min\left\{\sigma(X) := \frac{1}{2}\sum_{(i,j)\in\mathcal{S}^w} w_{ij}\left(\delta_{ij} - \|X_i^T - X_j^T\|\right)^2 : \quad X \in \mathcal{M}_{n,p}(\mathrm{I\!R})\right\}$$

for which four DC formulations were introduced.

The first is a DC formulation of (EDP) itself which takes the form

$$-\frac{1}{2}\eta_\delta^2 = \min\left\{F_1(X) := \frac{1}{2}\eta^2(X) - \xi(X) : \quad X \in \mathcal{M}_{n,p}(\mathrm{I\!R})\right\}, \tag{17.37}$$

where η and ξ are the functions defined on $\mathcal{M}_{n,p}(\mathrm{I\!R})$ by

$$\eta(X) = [\sum_{i<j} w_{ij} d_{ij}^2(X)]^{1/2} \quad \text{and} \quad \xi(X) = \sum_{i<j} w_{ij}\delta_{ij} d_{ij}(X). \tag{17.38}$$

It is not difficult to verify that η and ξ are two seminorms in $\mathcal{M}_{n,p}(\mathrm{I\!R})$ and thus (17.37) is a DC program.

The second is the following convex maximization problem which was proved to be equivalent to (EDP) via the stability of Lagrangian duality ([33]):

$$\omega := \max\{\xi(X) : \quad X \in U(\eta)\} \tag{17.39}$$

with $U(\eta) := \{X \in \mathcal{M}_{n,p}(\mathrm{I\!R}) : \eta(X) \leq 1\}$. This problem is in fact a DC program of the form

$$-\omega := \min\{\chi_{\mathcal{U}(\eta)}(X) - \xi(X) : \quad X \in \mathcal{M}_{n,p}(\mathrm{I\!R})\}. \tag{17.40}$$

The preceding DC programs (17.37) and (17.40) for (EDP) involve the l_1-norm in the definition of their objective functions. Besides, Le Thi and Pham Dinh [33] introduced the nonstandard l_∞ and $l_1 - l_\infty$ approach to reformulating the exact distance geometry problem by considering the function

$$\Phi(X) := \max\{\Phi_{ij}(X) := \frac{1}{2}w_{ij}[\|X_i^T - X_j^T\| - \delta_{ij}]^2 : (i,j) \in \mathcal{S}^w\}.$$

That leads to the two nonstandard DC programs using the $l_\infty-$norm and the combined l_1 - l_∞ norms in their formulations:

$$0 = \min\left\{ \Phi(X) := \max_{(i,j)\in\mathcal{S}^w} \left\{ \Phi_{ij}(X) := \frac{1}{2} w_{ij}[\|X_i^T - X_j^T\| - \delta_{ij}]^2 : X \in \mathcal{M}_{n,p}(\mathrm{I\!R}) \right\} \right\}, \tag{17.41}$$

$$0 = \min\left\{ \Phi(X) + \frac{\rho}{2} \sum_{(i,j)\in\mathcal{S}^w} w_{ij} \left(\delta_{ij} - \|X_i^T - X_j^T\|\right)^2 : X \in \mathcal{M}_{n,p}(\mathrm{I\!R}) \right\}. \tag{17.42}$$

Regularization techniques were also investigated to DC programs (17.37) and (17.40). Various DCA schemes were developed for solving (EDP). Besides, a two phases algorithm using shortest paths between all pairs of atoms to generate the complete dissimilarity matrix was investigated in order to compute a good starting point for the DCAs. Many numerical simulations of the molecular optimization problems with up to 12567 variables were reported which prove the practical usefulness of the nonstandard nonsmooth reformulations, the globality of found solutions, the robustness, and the efficiency of DCA based algorithms.

In [31], Le Thi and Pham Dinh proposed an *elegant* formulation to the general distance geometry problem (17.32) which is given by

$$\text{(GDP)}\ \ 0 = \min\left\{ \sum_{(i,j)\in\mathcal{S}} w_{ij}(\|x^i - x^j\| - t_{ij})^2 : x^1,\ldots,x^n \in \mathrm{I\!R}^3, l_{ij} \leqslant t_{ij} \leqslant u_{ij}, (i,j) \in \mathcal{S} \right\}. \tag{17.43}$$

This formulation corresponds to both exact and general distance geometry problems, because in the exact distance geometry problem one has $l_{ij} = u_{ij} = \delta_{ij}$, then we take $t_{ij} = \delta_{ij}$ and (17.43) becomes (EDP). Note that the standard optimization problem (17.33), (17.36) is a DC program but the objective DC function is too complex and inconvenient for DCA. The objective function of (GDP) in [31] makes it possible to express DCA in a simple form; it actually requires matrix-vector products and only one Cholesky factorization, and allows one to exploit sparsity in the large scale setting. The algorithms proposed in [31] are in fact composed of two phases: the first phase consists of finding a good starting point for Phase 2. It begins by completing the missing distance matrix (using the shortest paths between the pairs of atoms) and then solves the exact distance geometry problem in which all "distances" are known by the DCA. The second phase is the DCA applied to the original problem. The algorithms work well for the artificial test problems where a protein contains at most 4096 atoms and for the real world models derived from the PDB data bank ("http:www.rcsb.org/pdb/") with up to 4189 atoms. A disadvantage of these methods is that, although the strategy of Phase 1 is quite suitable for DCA to reach global solutions to the distance geometry problem, it is quite expensive (the running time of Phase 1 is equal to that of Phase 2).

To get around this drawback, more precisely, to find a good starting point of DCA without using Phase 1, Le Thi [35] proposed a combined DCA-smoothing technique. Instead of (17.43) a new DC formulation with a smooth (actually infinitely differentiable) objective function was considered:

$$0 = \min\left\{ \sum_{(i,j)\in\mathcal{S}} w_{ij}(\|x^i - x^j\|^2 - t_{ij}^2)^2 : x^1, \ldots, : x^n \in \mathrm{IR}^3, l_{ij} \leqslant t_{ij} \leqslant u_{ij}, (i,j) \in \mathcal{S} \right\}. \tag{17.44}$$

Like (17.43), this formulation corresponds to both exact and general distance geometry problems. The new formulation is favorable to the use of DCA as well as smoothing techniques via the Gaussian transform, because that the objective function is infinitely differentiable, and its Gaussian transform can be computed explicitly. Numerical experiments show that the proposed algorithm solves these problems more efficiently and reliably than the two phase DCA based method developed in [31].

As an illustrated example, we consider below the general distance geometry problem (GDP) and briefly present a DC formulation and the corresponding DCA for solving it.

By identifying an $n \times p$ matrix X with a $p \times n-$ vector, in what follows, we use either $\mathcal{M}_{n,p}(\mathrm{IR})$ or $\mathrm{IR}^{p\times n}$ for indicating the same notation. We can identify by rows (resp. columns) each matrix $X \in \mathcal{M}_{n,p}(\mathrm{IR})$ with a row-vector (resp. column-vector) in $(\mathrm{IR}^{\mathrm{p}})^{\mathrm{n}}$ (resp. $(\mathrm{IR}^{\mathrm{n}})^{\mathrm{p}}$) by writing respectively

$$X \longleftrightarrow \mathcal{X} = (X_1, \ldots, X_n), X_i^T \in \mathrm{IR}^{\mathrm{p}}, \mathcal{X}^{\mathrm{T}} \in (\mathrm{IR}^{\mathrm{p}})^{\mathrm{n}}, \tag{17.45}$$

and

$$X \longleftrightarrow \overline{\mathcal{X}} = (X^1, \ldots, X^p)^T, \quad X^i \in \mathrm{IR}^{\mathrm{n}}, \overline{\mathcal{X}} \in (\mathrm{IR}^{\mathrm{n}})^{\mathrm{p}}. \tag{17.46}$$

The inner product in $\mathcal{M}_{n,p}(\mathrm{IR})$ is defined as the inner product in $(\mathrm{IR}^{\mathrm{p}})^{\mathrm{n}}$ or $(\mathrm{IR}^{\mathrm{n}})^{\mathrm{p}}$. That is

$$\begin{aligned} \langle X, Y\rangle_{\mathcal{M}_{n,p}(\mathrm{IR})} &= \langle \mathcal{X}^T, \mathcal{Y}^T\rangle_{(\mathrm{IR}^{\mathrm{p}})^{\mathrm{n}}} = \sum_{i=1}^{n} \langle X_i^T, Y_i^T\rangle_{\mathrm{IR}^{\mathrm{p}}} = \sum_{i=1}^{n} X_i Y_i^T \\ &= \langle \overline{\mathcal{X}}, \overline{\mathcal{Y}}\rangle_{(\mathrm{IR}^{\mathrm{n}})^{\mathrm{p}}} = \sum_{k=1}^{p} \langle X^k, Y^k\rangle_{\mathrm{IR}^{\mathrm{n}}} = \sum_{k=1}^{p} (X^k)^T Y^k = Tr(X^T Y). \end{aligned}$$

Here $Tr(X^TY)$ denotes the trace of the matrix X^TY. In the sequel, for simplicity, we shall suppress, if no possible ambiguity, the indices for the inner product and denote by $\|\cdot\|$ the corresponding Euclidean norm on $\mathcal{M}_{n,p}(\mathrm{IR})$. Evidently, we must choose either representation in a convenient way.

We first prove that problem (GDP) is a DC program and point out DC components of the objective function of this problem. Since

$$-2t_{ij}\|x^i - x^j\| = -(\|x^i - x^j\| + t_{ij})^2 + \|x^i - x^j\|^2 + t_{ij}{}^2,$$

the objective function of (GDP) can be expressed as

$$\frac{1}{2}\sum_{(i,j)\in\mathcal{S}^w} w_{ij}\|x^i - x^j\|^2 + \frac{1}{2}\sum_{(i,j)\in\mathcal{S}^w} w_{ij}\, t_{ij}^2 - \frac{1}{4}\sum_{(i,j)\in\mathcal{S}^w} w_{ij}(\|x^i - x^j\| + t_{ij})^2. \tag{17.47}$$

Remark that by setting $w_{ij} = 0$ for $(i,j) \notin \mathcal{S}$ the constraint $(i,j) \in \mathcal{S}$ can be omitted in (GDP). Since the functions

$$\frac{1}{2}\sum_{(i,j)\in\mathcal{S}^w} w_{ij}\|x^i - x^j\|^2 + \frac{1}{2}\sum_{(i,j)\in\mathcal{S}^w} w_{ij}t_{ij}^2 \text{ and } \frac{1}{4}\left\{\sum_{(i,j)\in\mathcal{S}^w} w_{ij}(\|x^i - x^j\| + t_{ij})^2\right\}$$

are convex with respect to the variables $((x^1,\ldots,x^n),T)$ with $x^1,\ldots,x^n \in \mathrm{I\!R}^3$ and $T = (t_{ij})$ on the convex constraint set, it is clear that the function given in (17.47) is a DC function.

The matrix spaces that we shall present below are useful for various calculations of subgradients in DCA.

Let $\varphi_{ij} : \mathcal{M}_{n,n}(\mathrm{I\!R}) \longmapsto \mathrm{I\!R}$ be the pairwise function defined by $\varphi_{ij}(T) = t_{ij}$, and let η be the function defined by (17.38). Problem (GDP) can be now written in the matrix form

$$\begin{cases} 0 = \inf\,\{\mathcal{F}(X,T) := \ L(X,T) - \ K(X,T)\} \\ s.t. \qquad (X,T) \in \Omega := \mathcal{M}_{n,p}(\mathrm{I\!R}) \times \mathcal{T}, \end{cases}$$

with

$$L(X,T) := \frac{1}{2}\eta^2(X) + \mu(T),\ \mu(T) := \frac{1}{2}\sum_{i<j} w_{ij}\ \varphi_{ij}^2(T),$$

$$K(X,T) := \frac{1}{4}\sum_{i<j} w_{ij}[d_{ij}(X) + \varphi_{ij}(T)]^2,$$

$$\mathcal{T} := \{T \in \mathcal{M}_{n,n}(\mathrm{I\!R}) : l_{ij} \leqslant t_{ij} \leqslant u_{ij}, (i,j) \in \mathcal{S}^w\}. \tag{17.48}$$

Clearly, the function L is finite and convex on $\mathcal{M}_{n,p}(\mathrm{I\!R}) \times \mathcal{M}_{n,n}(\mathrm{I\!R})$. Since for every $(i,j) \in \mathcal{S}$, the function: $(X,T) \to d_{ij}(X) + \varphi_{ij}(T)$ is finite and convex on $\mathcal{M}_{n,p}(\mathrm{I\!R}) \times \mathcal{M}_{n,n}(\mathrm{I\!R})$ and nonnegative on Ω, the function K then is convex on Ω too. Let χ_Ω be the indicator function of Ω defined by $\chi_\Omega(X,T) = 0$ if $(X,T) \in \Omega, +\infty$ otherwise, then problem (GDP) can be expressed in the standard form of DC programs:

$$\begin{cases} 0 = \inf\,\{\ \mathcal{G}(X,T) - \ K(X,T)\} \\ s.t. \qquad (X,T) \in \mathcal{M}_{n,p}(\mathrm{I\!R}) \times \mathcal{M}_{n,n}(\mathrm{I\!R}), \end{cases} \tag{17.49}$$

where $\mathcal{G}(X,T) := L(X,T) + \chi_\Omega(X,T)$ is the separable function in its variables X and T. Before going further let us make precise the obvious relation between problems (17.32) and (17.49).

Proposition 1. *(i) If a set of positions $(x^1,\ldots,x^n)$ is a solution to problem (17.32), then the couple of matrices (X,T), with $X = (x^1,\ldots,x^n)^T$ and $t_{ij} = \|x^i - x^j\|$ for $(i,j) \in \mathcal{S}$, is a solution to (17.49).*

(ii) If a couple of matrices (X,T) is a solution to (17.49), then the set of positions $(x^1,\ldots,x^n) = X^T$ is a solution to (17.32) and $t_{ij} = \|X_i^T - X_j^T\|$ for $(i,j) \in \mathcal{S}$.

Performing this scheme by DCA thus is reduced to calculating subdifferentials of the functions K and $\mathcal{G}^*$.

$$(Y^{(k)},Z^{(k)}) \in \partial K(X^{(k)},T^{(k)}), (X^{(k+1)},T^{(k+1)}) \in \partial \mathcal{G}^*(Y^{(k)},Z^{(k)}). \tag{17.50}$$

Calculations of ∂K : Remark that Problem (17.49) only involves the restriction of K to Ω where this function is convex. Actually we shall compute the conditional subdifferential of K with respect to the convex set Ω ([12]) that we again denote by ∂K for simplicity, i.e., (Y^0,W^0) is a conditional subgradient of K at $(X^0,T^0) \in \Omega$ with respect to Ω if

$$K(X,T) \geq K(X^0,T^0) + \langle (X,T)-(X^0,T^0),(Y^0\ ,W^0)\rangle \text{ for } (X,T) \in \Omega.$$

By the very definition, it is easy to state the following inclusion for $(X,T) \in \Omega$

$$\partial K(X,T) \supset \frac{1}{2}\sum_{i<j} w_{ij}[d_{ij}(X)+\varphi_{ij}(T)][\partial d_{ij}(X)\times\{0\}+\{0\}\times\partial\varphi_{ij}(T)].$$

Since

$$\varphi_{ij}(T) = t_{ij} = \langle T, E_{ij}\rangle_{\mathcal{M}_{n,n}(\mathrm{I\!R})},$$

with $E_{ij} = e_i e_j^T$ ($e_i \in \mathrm{I\!R}^{\mathrm{n}}$ is the unit vector with value one in the i^{th} component and zero otherwise), φ_{ij} is differentiable on $\mathcal{M}_{n,n}(\mathrm{I\!R})$, and $\nabla\varphi_{ij}(T) = E_{ij}$. Hence $\partial K(X,T)$ contains the couples (Y,Z) defined by

$$(Y,Z) = \frac{1}{2}\sum_{i<j} w_{ij}(d_{ij}(X)+\varphi_{ij}(T))(Y(i,j),E_{ij}),$$

with $Y(i,j) \in \partial d_{ij}(X), i.e.$

$$Y = \frac{1}{2}\sum_{i<j} w_{ij}d_{ij}(X)Y(i,j) + \frac{1}{2}\sum_{i<j} w_{ij}\varphi_{ij}(T)Y(i,j),$$

$$Z = \frac{1}{2}\sum_{i<j} w_{ij}(d_{ij}(X)+\varphi_{ij}(T))E_{ij}.$$

Thus $Z = (z_{ij})$ is defined by

$$z_{ij} = \begin{cases} \frac{1}{2}w_{ij}(d_{ij}(X)+t_{ij}) & \text{if } (i,j) \in \mathcal{S}, \\ 0 & \text{if } (i,j) \notin \mathcal{S}. \end{cases}$$

For determining Y, we first compute $R := \sum_{i<j} w_{ij}d_{ij}(X)Y(i,j)$. By using the row-representation of $\mathcal{M}_{n,p}(\mathrm{I\!R})$, d_{ij} can be expressed as :

$$d_{ij} = \|.\| \circ \phi_{ij} : (\mathrm{IR}^{\mathrm{p}})^{\mathrm{n}} \longrightarrow \mathrm{IR}^{\mathrm{p}} \longrightarrow \mathrm{IR},\, X \longmapsto \phi_{ij}(X) = X_i^T - X_j^T \longmapsto \|X_i^T - X_j^T\|,$$

we have ([59])

$$\partial d_{ij}(X) = \phi_{ij}^T \partial(\|.\|)(\phi_{ij}(X)).$$

Hence

$$Y(i,j) \in \partial d_{ij}(X) \Leftrightarrow Y(i,j) = \phi_{ij}^T y, \quad y \in \partial(\|.\|)(X_i^T - X_j^T),$$

which implies

$$Y(i,j)_k^T = 0 \text{ if } k \notin \{i,j\} \text{ and } \quad Y(i,j)_i^T = -Y(i,j)_j^T \in \partial(\|.\|)(X_i^T - X_j^T). \quad (17.51)$$

Then $Y(i,j)$ can be chosen as

$$Y(i,j)_i = -Y(i,j)_j = \begin{cases} \frac{X_i - X_j}{\|X_i^T - X_j^T\|} & \text{if } X_i \neq X_j, \\ 0 & \text{otherwise.} \end{cases} \quad (17.52)$$

And so

$$R_k := \sum_{i<j} w_{ij} d_{ij}(X) Y(i,j)_k = \sum_{i<k} w_{ik} d_{ik}(X) Y(i,k)_k + \sum_{j>k} w_{kj} d_{kj}(X) Y(k,j)_k$$

$$= \sum_{i<k} w_{ik}(X_k - X_i) + \sum_{j>k} w_{jk}(X_k - X_j) = \left[\sum_{i=1}^n w_{ik}\right] X_k - \sum_{i=1}^n w_{ik} X_i.$$

It follows that

$$R = \sum_{i<j} w_{ij} d_{ij}(X) Y(i,j) = VX, \quad (17.53)$$

where $V = (v_{ij})$ is the $n \times n$ matrix given by

$$v_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j, \\ \sum_{k=1}^n w_{ik} & \text{if } i = j. \end{cases} \quad (17.54)$$

By the same way, we get

$$\sum_{i<j} w_{ij} \varphi_{ij}(T) Y(i,j) = C(X,T)X, \quad (17.55)$$

where $C(X,T) = (c_{ij}(X,T))$ is the $n \times n$ matrix valued function given by

$$c_{ij}(X,T) = \begin{cases} -w_{ij} t_{ij} s_{ij}(X) & \text{if } i \neq j, \\ -\sum_{k=1,k\neq i}^n c_{ik} & \text{if } i = j, \end{cases} \quad (17.56)$$

with

$$s_{ij}(X) = \begin{cases} 1/(\|X_i^T - X_j^T\|) & \text{if } X_i \neq X_j \text{ and } (i,j) \in \mathcal{S}, \\ 0 & \text{otherwise.} \end{cases} \quad (17.57)$$

Finally, we have

$$Y = \frac{1}{2}(V + C(X,T))X. \tag{17.58}$$

It has been shown in [31] that the range of V and that of $C(X,T)$, for a given couple (X,T), are contained in $\mathcal{A}^{\perp}$, where

$$\mathcal{A} := \{X \in \mathcal{M}_{n,p}(\mathrm{I\!R}) : X_1 = \cdots = X_n\}.$$

Hence the sequence $\{Y^{(k)}\}$ defined by (17.50) is contained in the subspace $\mathcal{A}^{\perp}$.

Computing $\partial\mathcal{G}^*$. As aforementioned, the calculation of $\partial\mathcal{G}^*(Y^{(k)}, Z^{(k)})$ consists of solving the next problem:

$$\min\{\mathcal{G}(X,T) - \langle (Y^{(k)}, Z^{(k)}), (X,T)\rangle : (X,T) \in \mathcal{M}_{n,p}(\mathrm{I\!R}) \times \mathcal{M}_{n,n}(\mathrm{I\!R})\ \}.$$

Since the functions $\mathcal{G}$ is separable in its variables, the last problem can be decomposed into the following two problems:

$$\min\left\{\frac{1}{2}\eta^2(X) - \langle Y^{(k)}\ , X\rangle :\ X \in \mathcal{M}_{n,p}(\mathrm{I\!R})\right\}, \tag{17.59}$$

$$\min\left\{\mu(T) - \langle Z^{(k)}\ , T\rangle :\ T \in \mathcal{T}\ \right\}. \tag{17.60}$$

According to the properties of V, solving (17.59) is reduced to computing the solution of the nonsingular linear system (17.61).

$$\left(V + \frac{1}{n}ee^T\right)X = Y. \tag{17.61}$$

It is easy to see that the solutions of (17.60) can be explicitly determined. Indeed, by definition of $Z^{(k)} = (z_{ij}^{(k)})$ we have

$$\begin{aligned}\mu(T) - \langle Z^{(k)}\ , T\rangle &= \frac{1}{2}\sum_{i<j,(i,j)\in\mathcal{S}} w_{ij}t_{ij}^2 - \sum_{i,j=1}^{n} z_{ij}^{(k)}t_{ij} \\ &= \frac{1}{2}\sum_{(i,j)\in\mathcal{S}^w} w_{ij}t_{ij}^2 - \frac{1}{2}\sum_{(i,j)\in\mathcal{S}^w} w_{ij}(d_{ij}(X^{(k)}) + t_{ij}{}^{(k)})t_{ij}.\end{aligned}$$

The solutions to (17.60) can be explicitly computed in the following way. $T^* = (t_{ij}^*)$ is a solution to this problem if and only if t_{ij} is arbitrary for $(i,j) \notin S$ and

$$t_{ij}^* = \begin{cases} \frac{1}{2}(d_{ij}(X^{(k)}) + t_{ij}^{(k)}) & \text{if} \quad l_{ij} \leqslant \frac{1}{2}(d_{ij}(X^{(k)}) + t_{ij}^{(k)}) \leqslant u_{ij}, (i,j) \in \mathcal{S}, \\ l_{ij} & \text{if} \quad \frac{1}{2}(d_{ij}(X^{(k)}) + t_{ij}^{(k)}) < l_{ij}, (i,j) \in \mathcal{S}, \\ u_{ij} & \text{if} \quad \frac{1}{2}(d_{ij}(X^{(k)}) + t_{ij}^{(k)}) > u_{ij}, (i,j) \in \mathcal{S}. \end{cases} \tag{17.62}$$

Only the components $t_{ij}^{(k+1)}$ with $(i,j) \in \mathcal{S}$ actually intervene in Problem (17.49), so we set $T^{(k+1)} = (t_{ij}^{(k+1)})$ as follows:

$$t_{ij}^{(k+1)} = 0 \text{ for } (i,j) \notin \mathcal{S} \text{ and } t_{ij}^{(k+1)} = t_{ij}^{*} \ for\ (i,j) \in \mathcal{S}. \tag{17.63}$$

From the above displayed calculations, we can now provide the description of the DCA applied to (17.49).

Description of the DCA for solving (17.49)

GDCA *(DCA applied to (17.49)).*

The sequence $\{(X^{(k)}, T^{(k)})\}$ *with* $X^{(k)} \in \mathcal{A}^{\perp}$ *is generated as follows:*

Let $\varepsilon > 0$, and $X^{(0)} \in \mathcal{A}^{\perp} \setminus \{0\}, T^{(0)} \in \mathcal{T}$ be given.

For $k = 0, 1, \ldots$ until

$$l_{ij} \leq \|X_i^{(k)} - X_j^{(k)}\| \leq u_{ij}, \text{ for all } (i,j) \in \mathcal{S} \tag{17.64}$$

determine $T^{(k+1)}$ following (17.62), and solve the nonsingular linear system

$$(V + \frac{1}{n} ee^T)X = \frac{1}{2}(V + C(X^{(k)}, T^{(k)}))X^{(k)} \tag{17.65}$$

to obtain $X^{(k+1)}$.

17.4.2 Molecular Conformation by Minimizing the Lennard-Jones Energy Function

The Lennard-Jones potential energy is defined by

$$f(x) = f(x^1, x^2, \ldots, x^n) := \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} v(\|x^i - x^j\|),$$

where n is the number of atoms in the cluster, $\|x^i - x^j\|$ stands for the Euclidean distance between atoms x^i and x^j, and the Lennard-Jones pair potential $v(r)$ is given by:

$$v(r) := \frac{1}{r^{12}} - \frac{2}{r^6} \text{ for } r > 0.$$

The so called Lennard-Jones problem is to *determine atomic cluster configurations with minimum Lennard-Jones potential energy*. It is one of the most studied and significant problems in molecular biophysics and biochemistry. Due to the non convexity of the Lennard-Jones energy function and the large number of distinct local minima of this function (about $O(e^{n^2})$), the Lennard-Jones problem has been considered to be a very difficult and challenging global optimization problem. It has become a benchmark for any new global optimization algorithm.

Several algorithms have been proposed during the past thirty years for solving this problem, with significant progresses, especially in recent years (see e.g. [3], [4], [13] - [15], [21], [23], [43], [53], [63], [64], [37], [41]). At the current time, the lowest energies of Lennard-Jones clusters containing up to 10000 atoms are archived. However, global optima of the Lennar-Jones energies are known only within the range $2 \leq n \leq 147$.

The Lennard-Jones potential energy can not be directly written as a DC function. In [37, 41], using some properties of composite functions and convex functions, Le Thi and Pham Dinh transformed f into a DC function and introduced a nice DC reformulation of the Lennard-Jones problem. Afterwards the authors developed a DCA scheme and a two phase DCA based algorithm for the resulting DC program. The proposed methods successfully locate the lowest known minima with a good performance on running time. Then, putative global minima of clusters having up to 6525 atoms are predicted with these methods. We describe shortly the approach developed in [37, 41]. First, the Lennard-Jone energy function is rewritten as

$$f(.) = f(x^1, x^2, \ldots, x^n) := \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w(\|x^i - x^j\|^2), \tag{17.66}$$

where the function w is defined on $]0, +\infty[$ by

$$w(r) := \frac{1}{r^6} - \frac{2}{r^3}, \; r > 0.$$

For a given $r_0 \in (0, \frac{1}{2}]$, let W_1 and W_2 be the convex functions on the whole $\mathbb{R}$ given by

$$W_1(r) := \begin{cases} \frac{1}{r^6} & \text{if } r \geq r_0, \\ a_1 r + b_1 & \text{if } r \leq r_0, \end{cases}, \; W_2(r) := \begin{cases} \frac{2}{r^3} & \text{if } r \geq r_0, \\ a_2 r + b_2 & \text{if } r \leq r_0, \end{cases}$$

where

$$a_1 := -\frac{6}{r_0^7}, \; b_1 = \frac{1}{r_0^6} - a_1 r_0 = \frac{7}{r_0^6}, \; a_2 := -\frac{6}{r_0^4}, \; b_2 = \frac{2}{r_0^3} - a_2 r_0 = \frac{8}{r_0^3}.$$

Now let G and H be the functions defined by

$$G(x^1, x^2, \ldots, x^n) := \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} g(\|x^i - x^j\|^2), : H(x^1, x^2, \ldots, x^n) := \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} h(\|x^i - x^j\|^2) \tag{17.67}$$

where

$$g(r) := W_1(r) + K_0 r, \; h(r) := W_2(r) + K_0 r, \; K_0 := \frac{6}{r_0^7}. \tag{17.68}$$

Then the following result has been proved in [37, 41] :

Theorem 6. *The Lennard-Jones problem is equivalent to the next DC program:*

$$\min\{G(x^1,x^2,\ldots,x^n)\ -H(x^1,x^2,\ldots,x^n)\ \mid x^i \in \mathbb{R}^3, i=1,...,n\} \tag{17.69}$$

in the sense that they have the same optimal value and the same set of global minima.

DCA applied on this DC program consists of computing at each iteration a subgradient of H and solving a smooth convex program ([37,41]).

17.4.3 Molecular Conformation by Minimizing the Morse Energy Function

The Morse potential is a convenient model for the potential energy of a diatomic molecule. The Morse potential energy is defined by

$$M(x) = M(x_1,x_2,\ldots,x_n;\rho) := \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(\|x_i - x_j\|;\rho),$$

where n is the number of atoms $x_i = (x_{i1},x_{i2},x_{i3}) \in \mathbb{R}^3$, ($i = 1,2,\ldots,n$) in the cluster, $\|x_i - x_j\|$ stands for the Euclidean distance between x_i and x_j, and the Morse pair potential $E(r;\rho)$ is defined as follows: ($\rho > 0$ is a parameter)

$$E(r;\rho) := e^{2\rho(1-r)} - 2e^{rho(1-r)}.$$

The so called Morse problem then is to *determine atomic cluster configurations with minimum Morse potential energy.*

Since the Morse energy is nonconvex and the number of distinct local minima in the potential energy surface of an n atoms Morse cluster is about $O(e^{n^2})$, the Morse problem is considered to be a very difficult and challenging global optimization problem. Although many algorithms have been proposed during the past thirty years, good putative global optima are known for small size problems. Like the Lennard-Jones problem, the Morse problem has become a benchmark for any new global optimization algorithms.

In [38] a nonsmooth DC formulation of the Morse problem and the corresponding DCA have been proposed. Preliminary computational results (for the magic sequence $n = 13,55,...,2057$) are very promising, they show the efficiency, globality and robustness of DCA with respect to existing methods.

17.5 Phylogenetic Tree Reconstruction

A fundamental task in evolutionary biology is the inference of phylogenetic or ancestral relationships among contemporary species. Given a set of aligned sequences (genes, proteins) from species, the goal of the phylogenetic tree reconstruction is to reconstruct the tree which best explains the evolutionary history of this gene/protein. Phylogenetic tree reconstruction is still a challenge today. Many concrete questions are still unresolved (e.g. mammalian evolutionary tree). Most realistic formulations of the problem, which take errors into account, give rise to hard computational problems. Popular phylogeny reconstruction methods can be divided into two main categories: Distance based methods (UPGMA, Neighbor Joining, Buneman trees) and Character based methods (Maximum Parsimony, Maximum Likelihood). Besides, there are some additional methods such as Quartets based, Disk-Covering. Experimental simulations in several works (see [22] for instance) indicate that maximum likelihood (ML) estimates of phylogenetic trees are consistently superior to parsimony or distance-based method. ML is one of the most widely used techniques to infer evolutionary histories.

Given a set of observed sequences and an underlying substitution model. ML aims to find the weighted tree (the tree with corresponding branch lengths) that maximizes the likelihood function $L(x,\tau)$ jointly over the vector of branch lengths x (continuous variables specifying the length of each edge in the tree) and the topology τ (combinatorial variable). The likelihood of a data is the conditional probability of producing the data, given the model parameters. Likelihood is a common optimization criteria in numerous settings, including phylogenetic. Du to the existence of multiple maxima, the problem of computing globally optimal ML estimates of phylogenetic trees is NP-hard and computationally intractable (see for example [10]).

Even if one of the two variables, x or τ, is held fixed, the likelihood maximization is difficult. For a fixed topology τ, and for most reasonable probabilistic models of nucleotide substitution, the likelihood function $L(x,\tau)$ is a highly nonlinear function of the branch lengths x. In [16] the authors proved that for a given τ the likelihood $L(x,\tau)$ is a DC function and proposed a cutting plane method for computing the branch lengths x by solving the problem

$$\max_{x\geq 0} \ln L(x). \tag{17.70}$$

This algorithms works on small datasets (5 sequences). In [29] the author developed a DCA based algorithm for (17.70) which works well on large size datasets. Hence the use of DCA is very recommended in phylogenetic trees reconstruction.

17.6 Conclusion

We have presented DC programming and DCA for modeling and solving three challenging classes of problems in computational biology. All the problems considered are also great challenge for the community of optimizers. These theoretical and algorithmic tools have been outlined in an appropriate way to make them understandable to the reader. They highlight the distinctive features (flexibility, versatility, inexpensiveness, scalability, efficiency and globality) of DC programming and DCA. It is desirable that our approaches will help researchers and practitioners tackle efficiently their nonconvex programs, especially in the large-scale setting.

References

1. Adams, W.P., Sherali, H.D.: A tight linearization and an algorithm for 0-1 quadratic programming problems. Management Science 32(10), 1274–1290 (1986)
2. Arthanari, T.S., Le Thi, H.A.: New formulations of the multiple sequence alignment problem. Optimization Letter 5(1), 27–40 (2011)
3. Barron, C., Gomez, S., Romero, D.: Lower Energy Icosahedral Atomic Cluster with Incomplete Core. Applied Mathematics Letters 10(5), 25–28 (1997)
4. Barron, C., Gomez, S., Romero, D., Saavedra, A.: A Genetic Algorithm for Lennard-Jones Atomic clusters. Applied Mathematics Letters 12, 85–90 (1999)
5. Billionnet, A., Elloumi, S.: Using a mixed integer quadratic programming solver for unconstrained quadratic 0-1 problem. Math. Programming 109(1, Ser.A), 55–68 (2007)
6. Blumenthal, L.M.: Theory and Applications of Distance Geometry. Oxford University Press (1953)
7. Cai, W., Jiang, H., Shao, X.: Global optimization of Lennard-Jones clusters by a parallel fast annealing evolutionary algorithm. Journal of Chemical Information and Computer Sciences 42(5), 1099–1103 (2002)
8. Caprara, A.: Constrained 0-1 quadratic programming: Basic approaches and extensions. European Journal of Operational Research 187, 494–1503 (2008)
9. Carr, R.D., Lancia, G.: Compact vs. Exponential-size LP relaxations. Operations Research Letters 30, 57–65 (2002)
10. Chor, B., Tuller, T.: Maximum likelihood of evolutionary trees is hard. In: Miyano, S., Mesirov, J., Kasif, S., Istrail, S., Pevzner, P.A., Waterman, M. (eds.) RECOMB 2005. LNCS (LNBI), vol. 3500, pp. 296–310. Springer, Heidelberg (2005)
11. Cripen, G.M., Havel, T.F.: Distance Geometry and Molecular Conformation. John Wiley & Sons (1988)
12. Demyanov, V.F., Vasilev, L.V.: Nondifferentiable optimization. Optimization Software, Inc. Publications Division, New York (1985)
13. Deaven, D.M., Tit, N., Morris, J.M., Ho, K.M.: Structural optimization of Lennard-Jones clusters by a genetic algorithm. Chemical Physics Letters 256(1), 195–200 (1996)
14. Deng, Y., Rivera, C.: Approximate energy minimization for large Lennard-Jones clusters. Journal of Global Optimization 16(4), 325–341 (2000)
15. Doye. J.P.K.: Thermodynamics and the global optimization of Lennard-Jones clusters. Journal of Chemical Physics 109(19), 8143–8153 (1998)
16. Ellis, S.E., Nayakkankuppam, M.V.: Phylogenetic analysis via DC programming. Department of Mathematics and Statistics (preprint)
17. Greenberg, H.J.: Integer Quadratic Programming Models in Computational biology. Operations Research Proceedings 2006, 83–95 (2007)
18. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
19. Havel, T.F.: An evaluation of computational strategies for use in the determination of protein structure from distance geometry constraints obtained by nuclear magnetic resonance. Prog. Biophys. Mol. Biol. 56, 43–78 (1991)
20. Hiriart Urruty, J.B., Lemarechal, C.: Convex Analysis and Minimization Algorithms. Springer, Heidelberg (1993)
21. Huang, H.X., Pardalos, P.M., Shen, Z.J.: Equivalent formulations and necessary optimality conditions for the Lennard-Jones problem. Journal of Global Optimization 22(1-4), 97–118 (2002)
22. Huelsenbeck, J.P.: Performance of phylogenetic methods in simulation. Systematic Biology 44, C17–C48 (1995)

23. Jiang, H., Cai, W., Shao, X.: New lowest energy sequence of marksf decahedral Lennard-Jones clusters containing up to 10,000 atoms. Journal of Physical Chemistry A 107(21), 4238–4243 (2003)
24. Kalantari, B., Rosen, J.B.: Algorithm for global minimization of linearly constrained concave quadratic functions. Mathematics of Operations Research 12, 544–561 (1987)
25. Kececioglu, J.: The maximum weight trace problem in multiple sequence alignment. In: Proceedings of the 4th Symposium on Combinatorial Pattern Matching, pp. 106–119 (1993)
26. Kececioglu, J.D.: Exact and Approximation Algorithms for DNA Sequence Reconstruction. PhD thesis, University of Arizona (1991)
27. Kececioglu, J.D., Lenhof, H.P., Mehlhorn, K., Mutzel, P., Reinert, K., Vingron, M.: A polyhedral approach to sequence alignment problems. Discrete Applied Mathematics 104, 143–186 (2000)
28. Le Thi, H.A.: DC Programming and DCA, `http://lita.sciences.univ-metz.fr/~lethi`
29. Le Thi, H.A.: Phylogenetic tree reconstruction by a DCA based algorithm. Research Report, LITA, University of Lorraine, 1–27 (2013)
30. Le Thi, H.A., Pham Dinh, T., Muu, L.D.: Numerical solution for optimization over the efficient set by d.c. optimization algorithm. Operations Research Letters 19, 117–128 (1996)
31. Le Thi, H.A., Pham Dinh, T.: D.C. programming approach for large scale molecular optimization via the general distance geometry problem. In: Optimization in Computational Chemistry and Molecular Biology: Local and Global Approaches. pp. 301–339. Kluwer Academic Publishers (2000)
32. Le Thi, H.A., Pham Dinh, T.: A continuous approach for large-scale constrained quadratic zero-one programming. Optimization (In honor of Professor ELSTER, Founder of the Journal Optimization) 50(1-2), 93–120 (2001)
33. Le Thi, H.A., Pham Dinh, T.: Large Scale Molecular Optimization from distance matrices by a d.c. optimization approach. SIAM Journal on Optimization 14(1), 77–114 (2003)
34. Le Thi, H.A., Pham Dinh, T.: A new algorithm for solving large scale molecular distance geometry problems. In: Applied Optimization: Special Issue “HighPerformance Algorithms and Software for Nonlinear Optimization”, pp. 279–296. Kluwer Academic Publishers (2003)
35. Le Thi, H.A.: Solving large scale molecular distance geometry problems by a smoothing technique via the gaussian transform and d.c. programming. Journal of Global Optimization 27(4), 375–397 (2003)
36. Le Thi, H.A., Pham Dinh, T.: The DC programming and DCA revisited with DC models of real world nonconvex optimization problems. Annals of Operations Research 133, 23–46 (2005)
37. Le Thi, H.A., Pham Dinh, T.: A two phases DCA based algorithm for solving the Lennard-Jones problem. Research Report, LITA, University of Metz, 1–36 (2011)
38. Le Thi, H.A., Pham Dinh, T.: Minimizing the Morse potential energy function by a DC programming approach. Research Report, LITA, University of Lorraine, 1–30 (2012)
39. Le Thi, H.A., Pham Dinh, T.: DC programming approaches for Distance Geometry problems. In: Mucherino, A., Lavor, C., Liberti, L., Maculan, N. (eds.) Distance Geometry: Theory, Methods and Applications. Springer (2013)
40. Le Thi, H.A., Pham Dinh, T., Belghiti, T.: DCA based algorithms for Multiple Sequence Alignment (MSA). Central European Journal of Operations Research, 1–24 (2013)

41. Le Thi, H.A., Pham Dinh, T.: DC programming and DCA for minimizing Lennard-Jones potential energy (submitted, 2014)
42. Lenhof, H.P., Retnert, K., Vingron, M.: A Polyhedral Approach to RNA Sequence Structure Alignmen. Journal of Computational Biology 5(3), 517–530 (1998)
43. Locatelli, M., Schoen, F.: Efficient algorithms for large scale global optimization: Lennard-Jones clusters. Computational Optimization and Applications 26(2), 173–190 (2003)
44. Myers, E., Miller, W.: Optimal alignments in linear space. Computer Applications in the Biosciences 4(1), 11–17 (1988)
45. Notredame, C.: Recent progresses in multiple sequence alignment: a survey. Pharmacogenomics 3(1), 131–144 (2002)
46. Notredame, C., Higgins, D.G., Heringa, J.: T-COFFEE: A novel method for fast and accurate multiple sequence alignment. J. Mol. Biol. 392, 205–217 (2000)
47. Pardalos, P.M., Rodgers, G.P.: Computational aspects of a branch and bound algorithm for quadratic zero-one programming. Computing 45, 131–144 (1990)
48. Pham Dinh, T., Le Thi, H.A.: Convex analysis approach to d.c. programming: Theory, Algorithms and Applications, Acta Mathematica Vietnamica (dedicated to Professor Hoang Tuy on the occasion of his 70th birthday) 22, 289–355 (1997)
49. Pham Dinh, T., Le Thi, H.A.: D.C. optimization algorithms for solving the trust region subproblem. SIAM J. Optimization 8, 476–505 (1998)
50. Pham Dinh, T., Le Thi, H.A.: Recent advances in DC programming and DCA. In: Nguyen, N.-T., Le-Thi, H.A. (eds.) TCCI 2013. LNCS, vol. 8342, pp. 1–37. Springer, Heidelberg (2014)
51. Pham Dinh, T., Nguyen, C.N., Le Thi, H.A.: An efficient combined DCA and B&B using DC/SDP relaxation for globally solving binary quadratic programs. J. Global Optimization 48(4), 595–632 (2010)
52. Prestwich, S., Higgins, D., O'Sullivan, O.: Pseudo-Boolean Multiple Sequence Alignment. Technical Report,TR-03-2003,, Cork Constraint Computation Centre, University College, Cork, Ireland (2003), `http://www.4c.ucc.ie/web/techreps.jsp`
53. Oskolkov, N.N., Jakob, B.: A Lennard-Jones-like perspective on first order transitions in biological helices. Central European Journal of Physics 11(3), 357–362 (2013)
54. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W:Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. Nucleic Acids Research 22(22), 4673–4680 (1994)
55. Thompson, J.D., Plewniak, F., Poch, O.: BAliBASE: A benchmark alignments database for the evaluation of multiple sequence alignment programs. Bioinformatics 15, 87–88 (1999)
56. Rajasekaran, S., Nick, H., Pardalos, P.M., Sahni, S., Shaw, G.: Efficient Algorithms for local alignment search. Journal of Combinatorial Optimization 5(1), 117–124 (2001)
57. Rajasekaran, S., Hu, Y., Luo, J., Nick, H., Pardalos, P.M., Sahni, S., Shaw, G.: Efficient Algorithms for similarity alignment search. Journal of Combinatorial Optimization 5(1), 117–124 (2001)
58. Reinert, K., Lenhof, H., Mutzel, P., Mehlhorn, K., Kececioglu, J.D.: A branch-and-cut algorithm for multiple sequence alignment. In: RECOMB, pp. 241–250 (1997)
59. Rockafellar, R.T.: Convex Analysis. Princeton University, Princeton (1970)
60. Saxe, J.B.: Embeddability of weighted Graphs in k-space is strongly NP-hard. In: Proc. 17 Allerton Conference in Communications, Control and Computing, pp. 480–489 (1979)

61. Searls, D.: Grand Challenges in Computational Biology. In: Salzberg, S., Searls, D., Kasif, S. (eds.) Computational Methods in Molecular Biology. Elsevier Science (1998)
62. Shashi, K.D., Katiyar, V.K.: Global Optimization of Lennard-Jones Potential Using Newly Developed Real Coded Genetic Algorithms. In: Proceedings of IEEE International Conference on Communication Systems and Network Technologies, pp. 614–618 (2011)
63. Xiang, Y., Cheng, L., Cai, W., Shao, X.: Structural distribution of Lennard-Jones clusters containing 562 to 1000 atoms. Journal of Physical Chemistry A 108(44), 9516–9520 (2004)
64. Xue, G.L.: Minimum Inter-Particle Distance at Global Minimizers of Lennard-Jones Clusters. Journal of Global Optimization 11, 83–90 (1997)
65. Vavasis, S.A.: Nonlinear Optimization, Complexity Issues, Oxford University Press (1991)

Subject Index