Lecture Notes in Computer Science 2509
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Cristian S. Calude Michael J. Dinneen
Ferdinand Peper (Eds.)

Unconventional Models of Computation

Third International Conference, UMC 2002
Kobe, Japan, October 15-19, 2002
Proceedings

Springer

Volume Editors

Cristian S. Calude
Michael J. Dinneen
University of Auckland, Department of Computer Science
Private Bag 92019, Auckland, New Zealand
E-mail: {cristian, mjd}@cs.auckland.ac.nz

Ferdinand Peper
Nanotechnology group, Kansai Advanced Research Center
Communications Research Laboratory
588-2 Iwaoka, Iwaoka-cho, Nishi-ku, Kobe-city, 651-2492, Japan
E-mail: peper@crl.go.jp

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Unconventional models of computation : third international conference ; proceedings / UMC 2002. Cristian S. Calude ... (ed.). - Berlin ; Heidelberg ; New York ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; 2509)
ISBN 3-540-44311-8

CR Subject Classification (1998): F.1, F.2

ISSN 0302-9743
ISBN 3-540-44311-8 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York,
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna e.K.
Printed on acid-free paper SPIN: 10870897 06/3142 5 4 3 2 1 0

Preface

The Third International Conference on **Unconventional Models of Computation, UMC 2002** was organized by the Center for Discrete Mathematics and Theoretical Computer Science and the Kansai Advanced Research Center of the Communications Research Laboratory, Kansai, Japan. The venue was held in the "unconventional" multipurpose Orbis Hall from 15 to 19 October 2002. Being part of the Kobe Fashion Museum, a disk-shaped building in the center of Kobe's Rokko Island, the Hall is conveniently located near the Hotel Plaza Kobe and the Kobe Bay Sheraton hotel.

Various natural processes motivate the construction of radically new models of computation. For example, the paper "Not Just a Pretty Face" published in the July 27, 2002 issue of *New Scientist* discusses the hypothesis that plants may have the power to compute without the benefit of a brain. This prompts the question of what sort of computation capability and complexity human bodies may be capable of, even without the help of the nervous system. Although thriving, the realization of powerful unconventional models of computing is still at an early stage in its development, and a huge and concerted effort is required to assess and exploit its real potential. This volume reports the main ideas, results, directions of research, and open questions, discussed at a highly intellectual gathering of leaders in this new field of research. The flow of discussion varies from theoretical aspects to practical implementations and philosophical reflection.

The eight invited speakers at the conference were: M.L. Campagnolo (Lisbon, Portugal), J. Copeland (Canterbury, New Zealand), A. DeHon (CalTech, USA), M. Ogihara (Rochester, USA), M. Ohya (Japan), M. Ozawa (Tohoku, Japan), P. Siwak (Poznan, Poland), and T. Toffoli (Boston, USA).

The Program Committee, consisting of L. Accardi (Roma, Italy), C.S. Calude (Chair; Auckland, NZ), M.J. Dinneen (Secretary; Auckland, NZ), R. Freivalds (Riga, Latvia), L.K Grover (Murray Hill, USA), J. Gruska (Brno, Slovak Republic), K. Morita (Hiroshima, Japan), M. Ogihara (Rochester, USA), F. Peper (Kobe, Japan), G. Rozenberg (Leiden, The Netherlands), P. Siwak (Poznan, Poland), T. Toffoli (Boston, USA), H. Umeo (Osaka, Japan), and T. Yokomori (Waseda, Japan), had the serious task of selecting 18 papers out of 36 submissions covering all major areas of unconventional computation, especially quantum computing, computing using organic molecules (DNA), membrane computing, cellular computing, and possibilities for breaking Turing's barrier.

The conference would not have been possible without the assistance of the following referees:

Luigi Accardi
Joshua Arulanandham
Cristian S. Calude
B. Jack Copeland
Michael J. Dinneen
Claudio Ferretti
Rudolf Freund
Jozef Gruska
Peter Hertling
Lila Kari
Fred Kroon
Cris Moore
Mitsunori Ogihara
Andrei Păun
George Păun
Ferdinand Peper
Pawel Siwak
M. Thakur
Hiroshi Umeo
H. Todd Wareham
Takashi Yokomori
Claudio Zandrom

We express our gratitude to Dr. Shinro Mashiko, Head of the Nanotechnology Group in the Kansai Advanced Research Center (KARC), Communications Research Laboratory (CRL), for his continuous encouragement and support.

We extend our thanks to all members of the Conference Committee, P. Davis (Kyoto), M. Hagiya (Tokyo), J.-Q. Liu (Kyoto), N. Matsui (Himeji), K. Morita (Hiroshima), H. Nishimura (Hyogo), F. Peper (Chair; Kobe), Y. Sato (Tokyo), S. Takeuchi (Sapporo), U. Guenther (Registration; Auckland), K. Umeno (Sapporo), and A. Yamaguchi (Fukuoka) for their invaluable organizational work, particularly to Prof. Nobuyuki Matsui of Himeji Institute of Technology, Prof. Kenichi Morita of Hiroshima University, and Dr. Yuzuru Sato of RIKEN.

We thank KARC (Nanotechnology Group) for the main financial support, and the Support Center for Advanced Telecommunications Technology Research (SCAT) for additional financial support.

It is a great pleasure to acknowledge the ideal cooperation with the Springer-Verlag team in Heidelberg for producing this volume in time for the conference.

August 2002

C.S. Calude
M.J. Dinneen
F. Peper

Table of Contents

Invited Papers

Contributed Papers

The Complexity of Real Recursive Functions

Manuel Lameiras Campagnolo

D.M./I.S.A., Universidade Técnica de Lisboa,
Tapada da Ajuda, 1349-017 Lisboa, Portugal
mlc@math.isa.utl.pt

Abstract. We explore recursion theory on the reals, the analog counterpart of recursive function theory. In recursion theory on the reals, the discrete operations of standard recursion theory are replaced by operations on continuous functions, such as composition and various forms of differential equations. We define classes of real recursive functions, in a manner similar to the classical approach in recursion theory, and we study their complexity. In particular, we prove both upper and lower bounds for several classes of real recursive functions, which lie inside the primitive recursive functions and, therefore, can be characterized in terms of standard computational complexity.

Keywords: Continuous-time computation, differential equations, recursion theory, computational complexity.

1 Introduction

Recursive function theory provides the standard notion of computable function [Cut80,Odi89]. Moreover, many time and space complexity classes have recursive characterizations [Clo99]. As far as we know, Moore [Moo96] was the first to extend recursion theory to real valued functions. We will explore this and show that all main concepts in recursion theory like basic functions, operators, function algebras, or functionals, are indeed extendable in a natural way to real valued functions. In this paper, we define recursive classes of real valued functions analogously to the classical approach in recursion theory and we study the complexity of those classes. In recursion theory over the reals, the operations typically include composition of functions, and solutions of several forms of differential equations. On one hand, we look at the structural properties of various algebras of real functions, i.e., we explore intrinsic properties of classes of real recursive functions such as closure under iteration, bounded sums or bounded products. We investigate links between analytical and computational properties of real recursive functions. For instance, we show that a departure from analyticity to C^∞ gives closure under iteration, a fundamental property of discrete functions. On the other hand, we use standard computational complexity theory to establish upper and lower bounds on those algebras. We establish connections between subclasses of real recursive functions, which range from the functions computable in linear space to the primitive recursive functions, and subclasses of

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 1–14, 2002.

the recursive functions closed under various forms of integration. We consider, in particular, indefinite integrals, linear differential equations, and more general Cauchy problems. Finally, we describe some directions of work that suggest that the theory of real recursive functions might be fruitful in addressing open problems in computational complexity.

2 Recursive Functions over $\mathbb{R}$

Moore [Moo96] proposed a theory of recursive functions on the reals, which is defined in analogy with classical recursion theory. A function algebra

$$[B_1, B_2, ...; O_1, O_2, ...],$$

which we also call a computational class, is the smallest set containing basic functions $\{B_1, B_2, ...\}$ and closed under certain operations $\{O_1, O_2, ...\}$, which take one or more functions in the class and create new ones.

Although function algebras have been defined in the context of recursion theory on the integers, they are equally suitable to define classes of real valued recursive functions. As a matter of fact, if the basic functions in a function algebra are real functions and the operators map real functions into real functions, then the function algebra is a set of real functions. Furthermore, if the basic functions have a certain property (e.g. continuity or differentiability) which is preserved by the operators, then every function in the class will have that same property on its domain of definition. In recursion theory on the reals we consider operations such as the following.

COMP (Composition). Given functions $f_1, \ldots, f_p$ of arity n and g of arity p, then define h such that $h(\boldsymbol{x}) = g(f_1(\boldsymbol{x}), ..., f_p(\boldsymbol{x}))$.

$\int$ (S-integration). Given functions $f_1, \ldots, f_m$ of arity n, and $g_1, \ldots, g_m$ of arity $n + 1 + m$, if there is a unique set of functions $h_1, \ldots, h_m$, such that

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}, 0) &= \boldsymbol{f}(\boldsymbol{x}), \\ \partial_y \boldsymbol{h}(\boldsymbol{x}, y) &= \boldsymbol{g}(\boldsymbol{x}, y, \boldsymbol{h}(\boldsymbol{x}, y)), \qquad \forall y \in I - S, \end{aligned} \tag{1}$$

on an interval I containing 0, where $S \subset I$ is a countable set of isolated points, and $\boldsymbol{h}$ is continuous for all $y \in I$, then $h = h_1$ is defined.

$\boldsymbol{\mu}$ (Zero-finding). Given f of arity $n + 1$, then define h such that

$$h(\boldsymbol{x}) = \mu_y f(\boldsymbol{x}, y) \stackrel{\text{def}}{=} \begin{cases} y^- = \sup\{y \in \mathbb{R}_0^- \ : \ f(\boldsymbol{x}, y) = 0\}, \text{ if } -y^- \leq y^+ \\ y^+ = \inf\{y \in \mathbb{R}_0^+ \ : \ f(\boldsymbol{x}, y) = 0\}, \text{ if } -y^- > y^+ \end{cases}$$

whenever it is well-defined.

To match the definition in [Moo96], derivatives of functions can have singularities (we denote the set of singularities by S). The definition above allows the derivative of h to be undefined on the singularities, as long as the solution is unique and continuous on the whole domain. To illustrate the definition of the operator $\int$, let's look at the following example.

Example 1. ($\sqrt{\cdot}$) Suppose that the constant 1 and the function $g(y, z) = 1/2z$ are defined. Then, the solution of

$$\partial_y h = \frac{1}{2h} \quad \text{and} \quad h(0) = 1. \tag{2}$$

is defined on $I = [-1, 1]$. Indeed, the function $h(y) = \sqrt{y+1}$ is the unique solution of Equation (2) on $[-1, 1]$. The set of singularities is $S = \{-1\}$, so $\partial_y h(y)$ doesn't have to be defined on $y = -1$.

Clearly, the operations above are intended as a continuous analog of operators in classical recursion theory, replacing primitive recursion and zero-finding on $\mathbb{N}$ with S-integration and zero-finding on $\mathbb{R}$. Composition is a suitable operation for real valued functions and it is therefore unchanged. Then, the class of real recursive functions is defined in [Moo96] as:

Definition 1. *The real recursive functions are* $[0, 1, U; \text{COMP}, \int, \boldsymbol{\mu}]$,

where 0 and 1 are simply constant functions, and U denotes the set of projections $U_i^n(x_1, \ldots, x_n) = x_i$. We also define real recursive constants as:

Definition 2. *A constant a is said to be computable if there is an unary real recursive function f such that $f(0) = a$.*

Then, if a constant a is computable, then one can also define, with composition and zero, a constant unary function g as $g(x) = f(0(x)) = a$, for all x. As we will see below, some irrational constants like e or π are real recursive, and therefore we can define a function whose value is precisely e or π. This is in contrast to the definition of real numbers computable by Turing machines, where an irrational number is said to be computable if there is a sequence of rationals that converge to it effectively.

If μ is not used at all we get M_0, the "primitive real recursive functions", i.e., $[0, 1, -1, U; \text{COMP}, \int]$. These include the differentially algebraic functions, as well as constants such as e and π. However, M_0 also includes functions with discontinuous derivatives like $|x| = \sqrt{x^2}$.

To prevent discontinuous derivatives, and to make our model more physically realistic, we may require that functions defined by integration only be defined on the largest interval containing 0 on which their derivatives are continuous. This corresponds to the physical requirement of bounded energy in an analog device. We define this in a manner similar to S-integration, but with the additional requirement of the continuity of the derivative:

$\bar{\text{I}}$ (SC^1-integration). Given functions $f_1, \ldots, f_m$ of arity n, and $g_1, \ldots, g_m$ of arity $n + 1 + m$, if there is a unique set of functions $h_1, \ldots, h_m$, such that

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}, 0) &= \boldsymbol{f}(\boldsymbol{x}), \\ \partial_y \boldsymbol{h}(\boldsymbol{x}, y) &= \boldsymbol{g}(\boldsymbol{x}, y, \boldsymbol{h}(\boldsymbol{x}, y)), \qquad \forall y \in I - S, \end{aligned} \tag{3}$$

on an interval I containing 0, where $S \subset I$ is a countable set of isolated points, and $\boldsymbol{h}$ and $\partial_y \boldsymbol{h}$ are both continuous for all $y \in I$, then $h = h_1$ is defined.

Then, restricting $\int$ to $\bar{\mathrm{I}}$, we define the class $[0, 1-1, U; \mathrm{COMP}, \bar{\mathrm{I}}]$. It is clear that all functions in $[0, 1-1, U; \mathrm{COMP}, \bar{\mathrm{I}}]$ are continuously differentiable on their domains. (A question that arises naturally is if they are of class C^∞.) Therefore, $f(y) = \sqrt{y+1}$ mentioned in Example 1 cannot be defined on the interval $[-1, 1]$ in $\overline{\mathcal{D}}$ anymore, since its derivative is not continuous on that interval.

Example 2. (θ_∞) In $\overline{\mathcal{D}}$ we can define a non-analytic function θ_∞ such that $\theta_\infty(t) = \exp(-1/t)$, when $t > 0$, and $\theta_\infty(t) = 0$, when $t \leq 0$. First consider the unique solution of the initial condition problem

$$z' = \frac{1}{(t+1)^2} z \quad \text{and} \quad z(0) = \exp(-1) \tag{4}$$

with a singularity at $t = -1$. This is $z(t) = 0$ if $t \leq -1$, and $z(t) = \exp(-\frac{1}{t+1})$ if $z > -1$. Then $\theta_\infty(t) = z(t-1)$. The function θ_∞ can be though as a C^∞ version of the Heaviside function θ, defined by $\theta(x) = 0$ when $x < 0$ and $\theta(x) = 1$ when $x \geq 0$.

We can restrict the integration operation even more, if we don't allow singularities for the derivatives in the domain of existence of the solution. Formally, we say that a functions is defined by proper integration if it is defined with the following operator:

I (Proper integration). Given functions $f_1, \ldots, f_m$ of arity n, and $g_1, \ldots, g_m$ of arity $n + 1 + m$, if there is a unique set of continuous functions $h_1, \ldots, h_m$, such that

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}, 0) &= \boldsymbol{f}(\boldsymbol{x}), \\ \partial_y \boldsymbol{h}(\boldsymbol{x}, y) &= \boldsymbol{g}(\boldsymbol{x}, y, \boldsymbol{h}(\boldsymbol{x}, y)), \qquad \forall y \in I, \end{aligned} \tag{5}$$

on an interval I containing 0, then $h = h_1$ is defined.

This proper form of integration preserves analyticity [Arn96]. Moreover, if the functions $f_1, \ldots, f_m$ and $g_1, \ldots, g_m$ are of class C^k, then h is also of class C^k on its domain of existence (cf. [Har82, 5.4.1]). Since constants and projections are analytic and composition and integration preserve analyticity, then:

Proposition 1. *All functions in* $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$ *are analytic on their domains.*

Similarly, one proves that functions of one variable in $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$ are precisely the differentially algebraic functions [Moo96,Gra02]. This means that the Gamma function, for instance, is not in the class $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$. Next, we give some examples of functions that do belong to that class.

Proposition 2. *The functions* $+$, $-$, $\times$, $\exp$, $\exp^{[m]}$, *defined as* $\exp^{[0]}(x) = 1$ *and* $\exp^{[n+1]}(x) = \exp(\exp^{[n]}(x))$ *for any integer* n, $\sin$, $\cos$, $1/x$, $\log$, *and* $\arctan$ *belong to* $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$.

To further explore the theory of real recursive functions, we restrict the integration operator to solving time-varying linear differential equations, i.e.,

LI *Linear integration.* Given $f_1, \ldots, f_m$ of arity n and $g_{11}, \ldots, g_{mm}$ of arity $n+1$, then define the function $h = h_1$ of arity $n+1$, where $\boldsymbol{h} = (h_1, \ldots, h_m)$ satisfies the equations $\boldsymbol{h}(\boldsymbol{x}, 0) = \boldsymbol{f}(\boldsymbol{x})$ and $\partial_y \boldsymbol{h}(\boldsymbol{x}, y) = \boldsymbol{g}(\boldsymbol{x}, y)\boldsymbol{h}(\boldsymbol{x}, y)$.

As in classical recursion theory, we define new classes by restricting some operations but adding to the class certain basic functions which are needed for technical reasons. A typical example is the integer function called cut-off subtraction, defined by $x \dot{-} y = x - y$ if $x \geq y$, and $x \dot{-} y = 0$ otherwise. In some real recursive classes we include, instead, a basic function we denote by θ_k and is defined by $\theta_k(x) = 0$ if $x \leq 0$, and $\theta_k(x) = x^k$ if $x > 0$. Clearly, θ_0 is an extension to the reals of the Heaviside function, and $\theta_1(x)$ is an extension to the reals of $x \dot{-} 0$. In general, θ_k is of class $\mathcal{C}^{k-1}$.

For example, we explore the class $[0, 1, -1, \pi, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$ for some fixed k. Since, unlike solving more general differential equations, linear integration can only produce total functions, then:

Proposition 3. *For any integer $k > 0$, if $f \in [0, 1, -1, \pi, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$, then f is defined everywhere and belongs to class C^{k-1}.*

We will also consider an even more restricted form of integration, which is just the indefinite integral. Formally, this is defined by:

INT *Indefinite integral.* Given $f_1, \ldots, f_m$ of arity n and $g_1, \ldots, g_m$ of arity $n+1$, then define the function $h = h_1$ of arity $n+1$, where $\boldsymbol{h} = (h_1, \ldots, h_m)$ satisfies the equations $\boldsymbol{h}(\boldsymbol{x}, 0) = \boldsymbol{f}(\boldsymbol{x})$ and $\partial_y \boldsymbol{h}(\boldsymbol{x}, y) = \boldsymbol{g}(\boldsymbol{x}, y)$.

3 Structural Complexity

In this section, we ask questions about *intrinsic* properties of classes of real recursive functions such as closure under certain operations. We will see that some intriguing connections exist among closure properties and analytical properties of the classes we consider.

Closure under iteration is a basic operation in recursion theory. If a function f is computable, so is $F(x, t) = f^{[t]}(x)$, the t'th iterate of f on x. We ask whether these analog classes are closed under iteration, in the sense that if f is in the class, then so is some $F(x, t)$ that equals $f^{[t]}(x)$ when t is restricted to the natural numbers.

Proposition 4. $[0, 1, -1, U; \mathrm{COMP}, \bar{\mathrm{I}}]$ *is closed under iteration.*

Proof. (Sketch) Let's denote $[0, 1, -1, U; \mathrm{COMP}, \bar{\mathrm{I}}]$ bu $\overline{\mathcal{D}}$. Given f, we can define in $\overline{\mathcal{D}}$ the differential equation

$$\begin{aligned} \big(\theta_\infty(\cos \pi t) + \theta_\infty(-\cos \pi t)\big)\, \partial_t y_1 &= -(y_1 - f(y_2))\, \theta_\infty(\sin 2\pi t) \\ \big(\theta_\infty(\sin \pi t) + \theta_\infty(-\sin \pi t)\big)\, \partial_t y_2 &= -(y_2 - y_1)\, \theta_\infty(-\sin 2\pi t) \end{aligned} \tag{6}$$

with initial condition $y_1(x, 0) = y_2(x, 0) = x$, where θ_∞ is the function defined in Example 2. We claim that the solution satisfies $y_1(x, t) = f^{[t]}(x)$, for all integer

$t \geq 0$. On the interval $[0, \frac{1}{2}]$, $y_2'(x,t) = 0$ because $\theta_\infty(-\sin 2\pi t) = 0$. Therefore, y_2 remains constant with value x, and $f(y_2) = f(x)$. The solution for y_1 on $[0, \frac{1}{2}]$ is then given by

$$\exp\left(\tfrac{1}{\sin(2\pi t)} - \tfrac{1}{\cos(\pi t)}\right) y_1' = -(y_1 - f(x)),$$

which we rewrite as $\epsilon\, y_1' = -(y_1 - f(x))$. Note that $\epsilon \to 0^+$ when $t \to 1/2$. Integrating the equation above we obtain

$$y_1 - f(x) = \exp\left(-\frac{1}{\epsilon}t\right),$$

where the right hand side goes to 0 when t approaches $1/2$. Therefore, $y_1(1/2) = f(x)$. A similar argument for y_2 on $[\frac{1}{2}, 1]$ shows that $y_2(x,1) = y_1(x,1) = f(x)$, and so on for y_1 and y_2 on subsequent intervals. The set of singularities of Equation (6) is $\{n/2, n \in \mathbb{N}\}$. □

However, if we replace $\mathrm{S}C^1$-integration by proper integration, which preserves analyticity, then the resulting class is no longer closed under iteration. More precisely,

Proposition 5. $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$ *is not closed under iteration.*

Proof. (Sketch) We denote $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$ by $\mathcal{D}$. Let's suppose that $\mathcal{D}$ is closed under iteration. Since $\exp \in \mathcal{D}$, then there is a function F in $\mathcal{D}$ such that $F(x,t) = \exp^{[t]}(x)$ for all $t \in \mathbb{N}$ and all $x \in \mathbb{R}$. Therefore, F has a finite description in $\mathcal{D}$ with a certain fixed number of uses of the I operation. However, it is known that functions of one variable in $\mathcal{D}$ are differentially algebraic [Moo96], that is, they satisfy a polynomial differential equation of finite order. So, for any fixed t, F is differentially algebraic in x. But, from a result of Babakhanian [Bab73], we know that $\exp^{[t]}$ satisfies no non-trivial polynomial differential equation of order less than t. This means that the number of integrations that are necessary to define $\exp^{[t]}$ has to grow with t, which creates a contradiction. □

Since $[0, 1, -1, U; \mathrm{COMP}, \bar{\mathrm{I}}]$ contains non-analytic functions while all functions in $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$ are analytic, one could ask if there is a connexion between those two structural properties of real recursive classes. We believe that closure under iteration and analyticity are related in the following sense:

Conjecture 1. Any non trivial real recursive class which is closed under iteration must contain non-analytic functions.

As a matter of fact, even if it is known that the transition function of a Turing machine can be encapsulated in an analytic function [KM99,Moo98], no analytic form of an iteration function is known.

Next we consider restricted operations as bounded sums and bounded products and we ask which real recursive classes are closed under those operations. We say that an analog class is closed under bounded sums (resp. products) if

given any f in the class, there is some g also in the class that equals $\sum_{n<t} f(x,n)$ (resp. $\prod_{n<t} f(x,n)$) when t is restricted to the natural numbers.

Let's see how to define bounded sums in a real recursive class. Not surprisingly, we find that this is related to indefinite integrals. We first define a step function F which matches f on the integers, and whose values are constant on the interval $[j, j+1/2]$ for integer j. F can be defined as $F(t) = f(s(t))$, where s is a continuous step function that matches the identity over the integers. This can be defined with the indefinite integral $s(0) = 0$ and $s'(x) = c_k\theta_k(-\sin 2\pi x)$.[1] Then $s(t) = j$, and $F(t) = f(s(t)) = f(j)$, whenever $t \in [j, j+1/2]$ for integer j. The bounded sum of f is then given by g, such that $g(0) = 0$ and $g'(t) = c_k F(t)\,\theta_k(\sin 2\pi t)$. Then $g(t) = \sum_{z<n} f(z)$ whenever $t \in [n-1, n-1/2]$. So, we can define bounded sums with the constant π, a periodic function like sin, θ_k, and the operation of indefinite integrals. More precisely,

Proposition 6. *For all $k \in \mathbb{N}$, $[0, 1, -1, \pi, \theta_k, \sin, U; \mathrm{COMP}, \mathrm{INT}]$ is closed under bounded sums. Moreover, any real recursive class which is closed under composition and indefinite integrals and contains the functions $0, 1, -1, \pi, \theta_k, \sin, U$ is closed under bounded sums.*

If a class is closed under bounded products and it contains, for instance, the identity function, then it has to contain functions that grow faster than polynomials. For instance, the class $[0, 1, -1, \pi, \theta_k, \sin, U; \mathrm{COMP}, \mathrm{INT}]$ cannot be closed under bounded products. What can we say if the analog class is closed under linear integration, instead of just indefinite integrals? We conjecture that the answer is still negative, since we believe that the simulation of bounded products would have to rely on a technique similar to Proposition 4 using synchronized clock functions, although we have no proof of this.

Let's then consider the following weaker property. We say that a class is closed under bounded products in a *weak sense* if, given any f in the class which has integer values for integer arguments (i.e., f is an extension to the reals of some $\tilde{f} : \mathbb{N} \to \mathbb{N}$), there is a g in the class such that $g(x,t) = \prod_{n<t} f(x,n)$ when t is restricted to the natural numbers. Then, in the presence of some appropriate non-analytic function like θ_k, proper integration and even linear integration are sufficient to simulate bounded products. In particular, we proved in [CMC] the following:

Proposition 7. *For all $k \in \mathbb{N}$, $[0, 1, -1, \pi, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$ is closed under bounded products in a weak sense.*

We can also show that a class is closed under the iteration of extensions to the reals of integer valued functions, as long as it is closed under proper integration, and it contains the non-analytic function θ_k or θ_∞. We call this property closure under iteration in a weak sense. For instance, it can be shown, using a technique similar to [Bra95], that

Proposition 8. *$[0, 1, -1, \theta_\infty, U; \mathrm{COMP}, \mathrm{I}]$ is closed under iteration in a weak sense.*

[1] The constant c_k is a rational or a rational multiplied by π.

4 Computational Complexity

In this section we explore connections among real recursive classes and standard recursive classes. Since we are interested in classes below the primitive recursive functions, we can characterize them in terms of standard space or time complexity, and consider the Turing machine as the underlying computational model. This approach differs from others, namely BSS-machines [BSS89] or information-based complexity [TW98], since it focus on *effective* computability and complexity. There are two main reasons to this. First, the Turing machine model allows us to represent the concept of Cauchy sequences and, therefore, supports a very natural theory of computable analysis. Second, we aim to use the theory of real recursive functions to address problems in standard computational complexity. This would be difficult to achieve with an intrinsically analog theory like the BSS-machines over $\mathbb{R}$.

To compare the computational complexity of real recursive classes and standard recursive classes we have to set some conventions. On one hand, we follow a straightforward approach to associate a class of integer functions to a real recursive class. We simply consider the *discretization* of a real recursive class, i.e., the subset of functions with integer values for integer arguments. More precisely,

Definition 3. *Given a real recursive class $\mathcal{C}$, $\mathcal{F}_{\mathbb{N}}(\mathcal{C}) = \{f : \mathbb{N}^n \to \mathbb{N}$ s.t. f has an extension to the reals in $\mathcal{C}\}$.*

If $\mathcal{F}_{\mathbb{N}}(\mathcal{C})$ contains a certain complexity class $\mathcal{C}'$, this means that $\mathcal{C}$ has at least the computational power of $\mathcal{C}'$, i.e., we can consider $\mathcal{C}'$ as a lower bound for $\mathcal{C}$.

On the other hand, we consider the computational complexity of real functions. We use the notion of [Ko91], which is equivalent to the one proposed by Grzegorczyk [Grz55], and whose underlying computational model is the function-oracle Turing machine. Intuitively, the time (resp. space) complexity of f is the number of moves (resp. the amount of tape) required by a function-oracle Turing machine to approximate the value of $f(x)$ within an error bound 2^{-n}, as a function of the input x and the precision of the approximation n.

Let's briefly recall what a function-oracle Turing machine is (we give an informal description: details can be found in [HU79,Ko91]). For any x in the domain of f, the oracle is a computable sequence ϕ such that for all $n \in \mathbb{N}$, $|\phi(n) - x| < 2^{-n}$. The machine is a Turing machine equiped with an additional query tape, and two additional states. When the machine enters in the query state, it replaces the current string s in the query tape by the string $\phi(s)$, where ϕ is the oracle, moves the head to the first cell of the query tape, and switches to the answer state. This is done in one step of the computation. We say that the time (resp. space) complexity of f on its domain is bounded by a function b if there is a function-oracle Turing machine which, for any x in the domain of f and an oracle ϕ that converges to x, computes an approximation of $f(x)$ with precision 2^{-n} in a number of steps (resp. amount of tape) bounded by $b(x, n)$. Then, for space complexity we define:

Definition 4. *Given a set of functions S, $\mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(S) = \{f : \mathbb{R}^n \to \mathbb{R}$ s.t. the space complexity of f is bounded by some function in $S\}$.*

Therefore, if a real recursive class $\mathcal{C}$ is contained in $\mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(S)$, then S can be considered a space complexity upper bound for $\mathcal{C}$.

Suppose that a function f can be successively approximated within an error 2^{-n} in a certain amount of space. Then, if f is integer, it just has to be approximated to an error less than $1/2$ to know its value exactly. Therefore, if a real recursive class $\mathcal{C}$ is computable in space bounded in $\mathcal{S}$, then the discretization of $\mathcal{C}$ is also computable in space bounded in S. Formally,

Proposition 9. *Let $\mathcal{C}$ be a real recursive class. If $\mathcal{C} \subset \mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(S)$, then $\mathcal{F}_{\mathbb{N}}(\mathcal{C}) \subset \mathcal{F}\mathrm{SPACE}(S)$.*

Given the two conventions established in Definition 3 and Definition 4, we will show upper and lower bounds on some real recursive classes. We can use the closure properties described in the last section to compare discretizations of real recursive classes with standard recursive classes. For instance, since $[0, 1, -1, U; \mathrm{COMP}, \bar{\mathrm{I}}]$ contains extensions of the zero function, successor, projections, and the cut-off function, and is closed under and composition and iteration, then we have the following upper bound for the primitive recursive functions:

Proposition 10. $\mathcal{PR} \subset \mathcal{F}_{\mathbb{N}}([0, 1, -1, U; \mathrm{COMP}, \bar{\mathrm{I}}])$.

Note that the same inductive proof works for $[0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{I}]$. Therefore, $\mathcal{PR} \subset \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{I}])$.

The elementary functions $\mathcal{E}$, which are closed under bounded sums and products, are a well-known class in recursion theory. All elementary functions are computable in elementary time or space, i.e., in time or space bounded by a tower of exponentials. As a matter of fact, the elementary functions are the smallest known class closed under time or space complexity [Odi00]. We showed in [CMC] that

Proposition 11. *For all $k \geq 0$, $\mathcal{E} \subset \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{LI}])$.*

In addition, we showed that all functions in $[0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$ are computable in elementary space (or time). Formally,

Proposition 12. *For all $k > 1$, $[0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{LI}] \subset \mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(\mathcal{E})$.*

Combining this with Proposition 9 and Proposition 11, we conclude that:

Proposition 13. *For all $k > 1$, $\mathcal{E} = \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{LI}])$*

which gives an analog characterization of the elementary functions. It is interesting that linear integration alone gives extensions to the reals of all elementary functions, since these are all the functions that can be computed by any practically conceivable digital device.

In recursion theory, several forms of bounded recursion are widely used, namely to obtain characterization of low complexity classes [Clo99]. In bounded recursion, an *a priori* bound is imposed on the function to be defined with the recursion scheme. Similarly, we can consider the following operator on real functions:

BI (Bounded integration). Given functions $f_1, \ldots, f_m$ of arity n, $g_1, \ldots, g_m$ of arity $n+1+m$, and b of arity $n+1$, if $(h_1, \ldots, h_m)$ is the unique function that satisfies the equations $\boldsymbol{h}(\boldsymbol{x}, y) = \boldsymbol{f}(\boldsymbol{x})$, $\partial_y \boldsymbol{h}(\boldsymbol{x}, y) = \boldsymbol{g}(\boldsymbol{x}, y, \boldsymbol{h}(\boldsymbol{x}, y))$, and $||\boldsymbol{h}(\boldsymbol{x}, y)|| \leq b(\boldsymbol{x}, y)$ on $\mathbb{R}^{n+1}$, then $h = h_1$ of arity $n+1$ is defined.

Let's consider the class $[0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]$.[2] All its functions are defined everywhere since this is true for the basic functions and its operators preserve that property. The *a priori* bound on the integration operation strongly restricts this class. All functions in the class $[0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]$ and its derivatives (for $k > 1$) are bounded by polynomials. Moreover, all functions computable in linear space have extensions in that class:

Proposition 14. *For all $k \geq 0$, $\mathcal{F}$LINSPACE $\subset \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}])$.*

Proof. (Sketch) Let's denote $[0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]$ by $\mathcal{B}_0$. Ritchie [Rit63] proved that the set of integer functions computable in linear space is the function algebra $[0, S, U, \times; \mathrm{COMP}, \mathrm{BREC}]$, usually denoted by $\mathcal{E}^2$, where BREC is bounded recursion. It is easy to verify that $\mathcal{B}_0$ contains extensions to the reals of zero, successor, projections, and binary product. Since $\mathcal{B}_0$ is closed under composition, we just have to verify that $\mathcal{B}_0$ is closed under bounded recursion in a weak sense. But since all functions in $\mathcal{B}_0$ have polynomials bounds, then this can be done with techniques similar to [Bra95] using bounded integration instead of integration. Details can be found in [Cam01]. □

The Ritchie hierarchy [Rit63] is one of the first attempts to classify recursive functions in terms of computational complexity. The Ritchie classes, which range from $\mathcal{F}$LINSPACE to the elementary functions, are the sets of functions computable in space bounded by a tower of exponentials of fixed height. Next we describe a hierarchy of real recursive classes where the first level is $[0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]$ (see above), and the n-th level is defined by allowing n nested applications of the linear integration operator. In each level of the hierarchy, indefinite integrals are freely used. As in the Ritchie hierarchy, composition is restricted. In [Rit63], the arguments of each recursive function are of two possible types: free and multiplicative. If f is multiplicative in the argument x, then f grows at most polynomially with x. The restricted form of composition forbids composition on two free arguments. For instance, if $2^x + y$ is free in x and multiplicative in y, then the composition $z = 2^x + y$ with $x(t) = 2^t$, which is free in t, is not allowed while the composition $z = 2^x + y$ with $y(t) = 2^t$ is. We denote this restricted composition by RCOMP and define the following hierarchy of real recursive classes (see [Cam01] for details):

Definition 5. *(The hierarchy $\mathcal{S}_n$) For all $n \geq 0$, $\mathcal{S}_n = [\mathcal{B}_0; \mathrm{RCOMP}, \mathrm{INT}, n \cdot \mathrm{LI}]$, where $\mathcal{B}_0 = [0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]$ for any fixed integer $k > 2$, and where the notation $n \cdot \mathrm{LI}$ means that the operator* LI *can be nested up to n times.*

[2] Given an appropriate bound, the binary product $h(x, y) = xy$ could be easily defined with bounded integration: $h(x, 0) = 0$, and $\partial_y h(x, y) = U_1^2(x, y) = x$. However, no other basic function grows as fast as the binary product, so this needs to be included explicitly in the class.

A few remarks are in order. First, all the arguments of a function h defined with linear integration, from any functions f, g of appropriate arities, are free. For instance, we are not allowed to compose the exponential function with itself, since its argument is free. Second, since solutions of linear differential equations $y'(t) = g(t)y(t)$ are always bounded by an exponential in g and t, and at most n nested applications of linear integration are allowed, then all functions in $\mathcal{S}_n$ have bounds of the form $\exp^{[n]}(p(x))$, where p is a polynomial. Even if the composition $\exp(\exp(x))$ is not permitted, towers of exponentials $\exp^{[n]} = \exp \circ \ldots \circ \exp$ can be defined in $\mathcal{S}_n$:

Example 3. ($\exp^{[n]} \circ p \in \mathcal{S}_n$). Let $u_i(\boldsymbol{x}, y) = \exp^{[i]}(p(\boldsymbol{x}, y))$ for $i = 1, \ldots, n$, where p is a polynomial. Then, the functions u_i are defined by the set of linear differential equations

$$\partial_y u_1 = u_1 \cdot \partial_y p \qquad \ldots \qquad \partial_y u_n = u_n \cdot u_{n-1} \cdots u_1 \cdot \partial_y p$$

with appropriate initial conditions. Thus u_n can be defined with up to n nested applications of LI and, therefore, $\exp^{[n]} \circ p \in \mathcal{S}_n$.

Next we relate the $\mathcal{S}_n$ hierarchy to the exponential space hierarchy (details of the proofs can be found in [Cam01]). Consider the following set of bounding functions:

$$2^{[n]} = \{b_k : \mathbb{N} \to \mathbb{N} \text{ s.t. } k > 0, b_k(m) = 2^{[n]}(k\, m) \text{ for all } m\}.$$

On one hand, $\mathcal{S}_n$ has the following upper bound:

Proposition 15. *For all $n \geq 0$, $\mathcal{S}_n \subset \mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(2^{[n+1]})$.*

Proof. (Sketch) All functions in $\mathcal{S}_n$, and its first and second derivatives, are bounded by $2^{[n]} \circ p$, where p is some polynomial. This follows from the fact that all basic functions in $\mathcal{S}_n$ have such property (this is why we restrict k in the Definition 5) and the operators of $\mathcal{S}_n$ preserve it. Then, using numerical techniques we show how to approximate a function defined by composition or bounded integration in $\mathcal{S}_0 = \mathcal{B}_0$. Given the bounds on the functions in $\mathcal{S}_0$ and their first derivative, composition can be computed in a straightforward manner, without increasing the space bounds. The major difficulty has to do with integration. We have to use an exponential amount of space to achieve a sufficiently good approximation. In fact, the standard techniques for numerical integration (Euler's method) require a number of steps which is exponential in the bounds on the derivatives of the functions we want to approximate [Hen62]. Since the bounds for functions in $\mathcal{S}_0$ are polynomial, the required number of steps N in the numerical integration is exponential. Thus all functions in $\mathcal{S}_0$ can be approximated in exponential space. Finally, we follow the same approach for other levels of the $\mathcal{S}_n$ hierarchy, where restricted composition replaces composition, and linear integration replaces bounded integration. □

On the other hand, all functions computable in space bounded by $2^{[n-1]}$ have extensions in $\mathcal{S}_n$. Formally,

Proposition 16. *For all* $n \geq 1$, $\mathcal{F}\mathrm{SPACE}(2^{[n-1]}) \subset \mathcal{F}_{\mathbb{N}}(\mathcal{S}_n)$.

Proof. (Sketch) As in [Rit63], we show that $\mathcal{F}\mathrm{SPACE}(2^{[n-1]})$ has a recursive definition, using restricted composition and a restricted form of bounded recursion. The following step is to define this restricted form of bounded recursion with bounded sums. Let's suppose that $f \in \mathcal{F}\mathrm{SPACE}(2^{[n-1]})$ is defined by bounded recursion. Then, we can encode the finite sequence $\{f(1), \ldots, f(n)\}$ as an integer (using for instance prime factorization), and replace bounded recursion by a bounded quantification over those encodings.[3] We use the fact that bounded quantifiers can be defined with bounded sums and cut-off subtraction. However, the bound on the encoding of the sequence $\{f(1), \ldots, f(n)\}$ is exponential on the bound on f. Therefore, we need an additional level of exponentials to replace bounded recursion by bounded sums. Finally, we know from Proposition 6 that $\mathcal{S}_n$ is closed under bounded sums, and contains cut-off subtraction as well. □

Unfortunately, we were not able to eliminate bounded integration from the definition of $\mathcal{B}_0$, neither were we able to show that $\mathcal{F}\mathrm{SPACE}(2^{[n]})$ is precisely $\mathcal{F}_{\mathbb{N}}(\mathcal{S}_n)$. We believe those issues are related with the open problem:

$$\mathcal{L}^2 \stackrel{?}{=} \mathcal{E}^2,$$

where $\mathcal{L}^2 = [0, S, U, \dot{-}; \mathrm{COMP}, \mathrm{BSUM}]$ is defined with bounded sums and is known as Skolem's lower elementary functions.[4] We consider instead the following problem:

$$\mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, +, U; \mathrm{COMP}, \mathrm{INT}]) \stackrel{?}{=} \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]).$$

At first sight, it seems that the equality above is false, since bounded integration is more general than indefinite integrals. However, the problem only concerns the discretizations of the analog classes. One could try to use results of the theory of differential equations to show directly that bounded integration is reducible, up to a certain error, to a finite sequence of indefinite integrals. It is known that solutions of general differential equations, $y'(t) = f(t, y)$ and $y(0) = y_0$, can be uniformly approximated by sequences of integrals, given some broad conditions that guarantee existence and uniqueness [Arn96,Har82]. However, that result, which is based on Picard's successive approximations, requires a sequence of integrals whose length increases with t. Since all functions in $[0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]$ and its derivatives are polynomially bounded, it might be possible to find a finite approximation for bounded integration, which would be sufficient to approximate functions which range on the integers. Notice that the standard numerical techniques (Euler's method) to approximate the solution of $y'(t) = f(t, y)$ and $y(0) = y_0$ require a number of approximation steps which are exponential in the bounds on the derivative, while Picard's method only needs a polynomially long sequence of indefinite integrals, if the bounds on the derivatives are polynomial.

[3] We follow a known technique in recursion theory (see [Ros84]).

[4] Recall that $\mathcal{E}^2 = [0, S, U, \times; \mathrm{COMP}, \mathrm{BREC}]$ and is precisely $\mathcal{F}\mathrm{LINSPACE}$. Notice that $\mathcal{L}^2 \subset \mathcal{E}^2$.

If the equality above is true, and if $\mathcal{F}_{\mathbb{N}}([0,1,-1,\theta_k,+,U;\mathrm{COMP},\mathrm{INT}])\subset\mathcal{L}^2$, then we would obtain a chain of inclusions that would show that $\mathcal{L}^2 = \mathcal{E}^2$. These remarks above establish a connection between the theory of real recursive functions and computational complexity that would be interesting to explore.

5 Final Remarks

We described some results on real recursive functions and we listed some open problems and directions for further research. We believe that recursion theory over the reals is not only an interesting area of research by itself, but it is also related to other areas such as computational complexity, numerical analysis or dynamical systems.

We mentioned possible links to computational complexity in the last section. It would be interesting to look at real recursive classes related to low time complexity classes. For instance, it is unlikely that the class in Proposition 6 is contained in $\mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(P)$, where P is the set of polynomials, since if $\mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(P)$ is closed under INT, then $\#P = \mathcal{F}\mathrm{PTIME}$ [Ko91]. Therefore, schemes of integration other than the ones we described in this paper have to be explored to find analogues to $\mathcal{F}$PTIME or other low time complexity classes.

We would like to clarify the connections between real recursive functions and dynamical systems. It is known that the unary functions in $[0,1,-1,U;\mathrm{COMP},\mathrm{I}]$ are precisely the solutions of equations $\boldsymbol{y}' = p(\boldsymbol{y},x)$, where p is a polynomial [Gra02]. We conjecture that $[0,1,-1,U;\mathrm{COMP},\mathrm{LI}]$ corresponds to the family of dynamical systems $\boldsymbol{y}' = f(\boldsymbol{y},x)$, where each f_i is linear and depends at most on $x, y_1, \ldots, y_i$. Given such canonical representations of classes of real recursive functions, one could investigate their dynamical properties.

Acknowledgements. I thank Cristian Calude, Michael Dinneen and Ferdinand Peper for inviting me to UMC'02, Félix Costa and Cris Moore for their collaboration, Amílcar Sernadas, Daniel Graça and Yuzuru Sato for useful discussions, Kathleen for helpful suggestions, and Emma and Isabel for dragging me out of bed at 6:30 every morning. This work was partially supported by the Center for Logic and Computation, Laboratório de Modelos e Arquitecturas Computacionais, and Fundação para a Ciência e Tecnologia (PRAXIS XXI/BD/18304/98).

References

[Arn96] V. I. Arnold. *Equations Différentielles Ordinaires*. Editions Mir, 5 ème edition, 1996.

[Bab73] A. Babakhanian. Exponentials in differentially algebraic extension fields. *Duke Math. J.*, 40:455–458, 1973.

[Bra95] M.S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 1995.

[BSS89] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completness, recursive functions and universal machines. *Bull. Amer. Math. Soc.*, 21:1–46, 1989.

[Cam01] M.L. Campagnolo. *Computational complexity of real recursive functions and analog circuits*. PhD thesis, Instituto Superior Técnico, 2001.

[Clo99] P. Clote. Computational models and function algebras. In E.R. Griffor, editor, *Handbook of Computability Theory*, pages 589–681. Elsevier, 1999.

[CMC] M.L. Campagnolo, C. Moore, and J.F. Costa. An analog characterization of the Grzegorczyk hierarchy. To appear in the *Journal of Complexity*.

[Cut80] N. J. Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, 1980.

[Gra02] D. Graça. The general purpose analog computer and recursive functions over the reals. Master's thesis, Instituto Superior Técnico, 2002.

[Grz55] A. Grzegorczyk. Computable functionals. *Fund. Math.*, 42:168–202, 1955.

[Har82] P. Hartman. *Ordinary Differential Equations*. Birkhauser, 2nd edition, 1982.

[Hen62] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations*. Wiley, New York, 1962.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addisson-Wesley, 1979.

[KM99] P. Koiran and C. Moore. Closed-form analytic maps in one or two dimensions can simulate Turing machines. *Theoretical Computer Science*, 210:217–223, 1999.

[Ko91] K.-I. Ko. *Complexity Theory of Real Functions*. Birkhauser, 1991.

[Moo96] C. Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162:23–44, 1996.

[Moo98] C. Moore. Finite-dimensional analog computers: flows, maps, and recurrent neural networks. In C.S. Calude, J. Casti, and M.J. Dinneen, editors, *Unconventional Models of Computation*, DMTCS. Springer-Verlag, 1998.

[Odi89] P. Odifreddi. *Classical Recursion Theory*. Elsevier, 1989.

[Odi00] P. Odifreddi. *Classical Recursion Theory II*. Elsevier, 2000.

[Rit63] R.W. Ritchie. Classes of predictably computable functions. *Transactions Amer. Math. Soc.*, 106:139–173, 1963.

[Ros84] H.E. Rose. *Subrecursion: Functions and Hierarchies*. Clarendon Press, 1984.

[TW98] J. Traub and A.G. Werschulz. *Complexity and Information*. Cambridge University Press, 1998.

Hypercomputation in the Chinese Room

B. Jack Copeland

Philosophy Department, University of Canterbury, New Zealand
bjcopeland@cantva.canterbury.ac.nz

Abstract. I rehearse a number of objections to John Searle's famous Chinese room argument. One is the 'hypercomputational objection' (Copeland 2002a). Hypercomputation is the computation of functions that cannot be computed in the sense of Turing (1936); the term originates in Copeland and Proudfoot (1999). I defend my hypercomputational objection to the Chinese room argument from a response recently developed by Bringsjord, Bello and Ferrucci (2001).

1 The Vanilla Argument

In its basic form, the Chinese room argument addresses the case of a human clerk-call him or her Clerk-who 'handworks' a GOFAI program.[1] The program is presented to Clerk in English in the form of a set of rule-books. One not uncommon misunderstanding takes Searle to be asserting that the rule-books contain a simple 'look-up' table that pairs possible inputs directly with ready-made outputs (see for example Sterelny (1990): 220ff). So misinterpreted, Searle's argument is weakened; Searle's intention is that the rule-books may contain any GOFAI program that is claimed by its creators (or others) to understand Chinese-or indeed to have any 'cognitive states' whatsoever (Searle 1980: 417). The symbols processed by the program are not limited to Chinese ideograms (which merely form a vivid example); the symbols may even be generated by a suite of sensors mounted on motor equipment that is itself under the control of the program (1980: 420).

To the programmers outside, the verbal behaviour of the Room-the system that includes the rule-books, Clerk, the erasable paper memory, Clerk's pencils and rubbers, the input and output provisions (paper cards and slots) and any clock, random number generator, or other equipment that Clerk may need in order to execute the precise program in question-is indistinguishable from that of a native Chinese speaker. But does the Room understand Chinese?

Here is the vanilla argument (Searle 1980: 418):

> [Clerk] do[es] not understand a word of the Chinese stories. [Clerk] ha[s] inputs and outputs that are indistinguishable from the native Chinese speaker, and [Clerk] can have any formal program you like, but [Clerk]

[1] Good Old Fashioned AI: John Haugeland's excellent term for traditional symbol-processing AI (Haugeland 1985).

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 15–26, 2002.

> still understand[s] nothing. Schank's computer for the same reasons understands nothing of any stories ... [W]hatever purely formal principles you put into the computer will not be sufficient for understanding, since a human will be able to follow the formal principles without understanding ... .

The flaw in the vanilla version is simple: the argument is not logically valid.[2] (An argument is logically valid if and only if its conclusion is entailed by its premiss(es); an argument is sound if and only if it is logically valid and each premiss is true.) The proposition that the formal symbol manipulation carried out by Clerk does not enable Clerk to understand the Chinese story by no means entails the quite different proposition that the formal symbol manipulation carried out by Clerk does not enable the Room to understand the Chinese story. One might as well claim that the statement 'The organisation of which Clerk is a part has no taxable assets in Switzerland' follows logically from the statement 'Clerk has no taxable assets in Switzerland'.

It is important to distinguish this, the logical reply to the vanilla argument, from what Searle calls the systems reply. The systems reply is the following claim (1980: 419):

> While it is true that the individual person who is locked in the room does not understand the story, the fact is that he is merely part of a whole system and the system does understand the story. As Searle correctly points out, the systems reply is worthless, since it 'simply begs the question by insisting without argument that the system must understand Chinese' (1980: 419). The logical reply, on the other hand, is a point about entailment. The logical reply involves no claim about the truth-or falsity-of the statement that the Room can understand Chinese.

2 The Roomless Room

Of course, any logically invalid argument can be rendered valid with the addition of further premisses (in the limiting case one simply adds the conclusion to the premisses). The trick is to produce additional premisses which not only secure validity but are sustainable.

In his discussion of the systems reply Searle says (1980: 419):

> My response to the systems theory is quite simple: Let the individual ... memoriz[e] the rules in the ledger and the data banks of Chinese symbols, and [do] all the calculations in his head. The individual then incorporates the entire system. ... We can even get rid of the room and suppose he works outdoors. All the same, he understands nothing of the Chinese, and a fortiori neither does the system, because there isn't anything in the system that isn't in him. If he doesn't understand, then there is no way the system could understand, because the system is just a part of him.

[2] (Copeland 1993a, 1993b.)

Let me represent this, the outdoor version of the argument, as follows:
(2.1) The system is part of Clerk.
(2.2) If Clerk (in general, x) does not understand the Chinese story (in general, does not), then no part of Clerk (x) understands the Chinese story (s).
(2.3) The formal symbol manipulation carried out by Clerk does not enable Clerk to understand the Chinese story.
Therefore:
(2.4) The formal symbol manipulation carried out by Clerk does not enable the system to understand the Chinese story.

The outdoor version is logically valid. Premiss (2.1) is perhaps innocent enough. Attention thus centres on (2.2), which I call the Part-Of principle.[3] Searle makes no mention at all of why he thinks the Part-Of principle is true. Yet the principle is certainly not self-evident. It is all too conceivable that a homunculus or homuncular system in Clerk's head should be able to understand Chinese without Clerk being able to do so. (Notice that Searle has no reservations concerning the application of predicates like 'understand' to sub-personal systems. He writes (against Dennett): 'I find nothing at all odd about saying that my brain understands English. ... I find [the contrary] claim as implausible as insisting "I digest pizza; my stomach and digestive tract don't".' (1980: 451).) Likewise for related values. Conceivably there is a special-purpose module in Clerk's brain that produces solutions to certain tensor equations, yet Clerk himself may sincerely deny that he can solve tensor equations-does not even know what a tensor equation is, we may suppose. Perhaps it is the functioning of this module that accounts for our ability to catch cricket balls and other moving objects (McLeod and Dienes 1993). Clerk himself, we may imagine, is unable to produce solutions to the relevant tensor equations even in the form of leg and arm movements, say because the output of the module fails to connect owing to the presence of a lesion. Or, to move into the realm of science fiction, neuropharmacologists may induce Clerk's liver to emulate a brain, the liver remaining in situ and receiving input directly from a computer workstation, to which the liver also delivers its output. Clerk's modified liver performs many acts of cognition that Clerk cannot. One example: Clerk stares uncomprehendingly at the screen of the computer as his liver proves theorems in quantified tense logic.

Of course, one might respond to these and similar examples as follows. Since a part of Clerk is proving a theorem of quantified tense logic (solving a set of tensor equations, etc.) then so is Clerk-there he is doing it, albeit to his own surprise. This response cannot be available to Searle. If Clerk's sincere denial that he is able to solve tensor equations (or what have you) counts for nothing, then likewise in the case of the Chinese room. However, it is a cornerstone of Searle's overall case that Clerk's sincere report 'I don't speak [sic] a word of Chinese' (1980: 418) suffices for the truth of premiss (2.3). One might call this Searle's Incorrigibility Thesis. It, like the Part-Of principle, is left totally unsupported by Searle. (Searle sometimes says, as if to give independent support to (2.3):

[3] (Copeland 2002a.)

'there is no way [Clerk] could come to understand Chinese in the [situation] as described, since there is no way that [Clerk] can learn the meanings of any of the symbols' (1990: 20). This rhetoric simply begs the question, since the matter at issue is whether 'just having the symbols by themselves ... [is] sufficient for semantics' and that this cannot be sufficient is, allegedly, 'the point that the Chinese room demonstrated' (1990: 21).)

If the Part-Of principle is taken to be a modal claim equivalent to NOT-POSSIBLY((some part of Clerk understands Chinese) & NOT(Clerk understands Chinese)) then, assuming the Incorrigibility Thesis, possible scenarios such as the foregoing do more than bear on the plausibility of the principle: they settle its truth-value. If, on the other hand, the Part-Of principle is said to be a purely contingent claim (i.e. a claim that happens to be true in the actual world but is not true in possible alternatives to the actual world), then Searle's difficulty is to produce reasons for thinking the principle true.

3 The Church-Turing Fallacy and the Hypercomputational Objection

Searle believes it follows from Church's thesis that the activity of the brain can be simulated by a universal Turing machine:

> Can the operations of the brain be simulated on a digital computer [read: Turing machine]? ... [G]iven Church's thesis that anything that can be given a precise enough characterization as a set of steps can be simulated on a digital computer, it follows trivially that the question has an affirmative answer. (1992: 200-201; see also 1997: 87.)

However, Searle's statement of Church's thesis is mistaken. Church's thesis, also known as 'Turing's thesis' and the 'Church-Turing thesis' (Kleene 1967), is in fact a proposition concerning what can be achieved by a human clerk who manipulates symbols unaided by any machinery (save paper and pencil) and who works in accordance with 'mechanical' methods, i.e. methods set out in the form of a finite number of exact instructions that call for no insight or ingenuity on the part of the clerk. The Church-Turing thesis (Turing 1936, Church 1936a, 1936b, 1937) states that whatever can be calculated by a clerk so working, even a clerk idealised to the extent of being free of all constraints on time, patience, concentration, and so forth, can also be calculated by a Turing machine. This thesis carries no implication concerning the extent of what can be calculated by a machine (say one that operates in accordance with a finite program of instructions), for among the machine's repertoire of primitive operations there may be those that no human being unaided by machinery can perform. Nor does the thesis imply that each process admitting of a precise characterisation 'as a set of steps' can be simulated by a Turing machine, for the steps need not be ones that a human clerk working in accordance with some mechanical method can carry out.

Searle's mistake concerning the nature of the Church-Turing thesis is, in fact, a common one, which can be frequently encountered in recent writing on the philosophy of mind (see further Copeland 2000). One way to assist the extinction of a fallacy is to introduce a name for it. In (1998a) I suggested the term Church-Turing fallacy for the fallacy of believing that the Church-Turing thesis, or some formal or semi-formal result established by Turing or Church, secures the truth of the proposition that the brain (or every machine) can be simulated by a universal Turing machine.

In his Ph.D thesis (Princeton 1938) Turing introduced the notional machines that he called 'O-machines'. Subsequently published as Turing (1939), this work is a classic of recursive function theory. Yet it is seemingly little-known among philosophers of mind. An O-machine (see further Copeland 1997, 1998a; Copeland and Proudfoot 1999) is a programmable device able to perform mathematical tasks that are too difficult for a universal Turing machine. (Turing proved in (1936) that such tasks exist.) An O-machine consists of a 'head' and a paper tape of unbounded length. Each square of the tape is either blank or contains a discrete symbol. A finite segment of the tape contains the program that is to be executed and the symbols that form the input. The O-machine manipulates symbols in a serial, step-by-step manner in accordance with the rules specified by the program. Every primitive operation of an O-machine that is not among the primitive operations of a Turing machine is, by definition, a formal operation on discrete symbols (in essence an operation that replaces a binary string with 1 or 0).

Trivially, the processing of an O-machine is always characterisable as a set of steps, namely, the set of steps specified by the machine's program. Employing the thesis espoused by Searle in the above quotation yields the absurdity that an O-machine can be simulated by a Turing machine-a contradiction.

An O-machine's program may call for primitive operations that a human clerk working by rote and unaided by machinery is incapable of carrying out (for otherwise, by the real Church-Turing thesis, whatever can be calculated by an O-machine can be calculated by a Turing machine). Searle's Chinese room argument cannot successfully be deployed against the functionalist hypothesis that the brain instantiates an O-machine (a hypothesis which Searle will presumably find as 'antibiological' (1990: 23) as other functionalisms). This is because the argument depends on Clerk being able to carry out by hand each operation that the program in question calls for. Turing originally introduced the Turing machine as a model of a human clerk engaged in mathematical calculation (1936: 231), and so, of course, each primitive operation of a Turing machine is indeed one that a human clerk can carry out. The same is true of the electronic machines fashioned after the universal Turing machine. As Turing himself put it (1950: 1):

Electronic computers are intended to carry out any definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner. O-machines, on the other hand, conspicuously fail to satisfy this ground condition of the Chinese room argument.

Searle has repeatedly emphasised that it is the fact that computers have 'no more than just formal symbols' which entails that programs 'cannot constitute the mind' and that this is what the Chinese room argument shows (1989: 33; 1992: 200).

The whole point of the original [i.e. indoor] example was to argue that ... symbol manipulation by itself couldn't be sufficient for understanding Chinese. (1980: 419.)

[F]ormal syntax ... does not by itself guarantee the presence of mental contents. I showed this a decade ago in the Chinese room argument. (1992: 200.)

But O-machines point up the fact that the notion of a programmed machine whose activity consists of the manipulation of formal symbols is more general than the notion of a universal Turing machine. If there is an implication from 'x's operation is defined purely formally or syntactically' to 'x's operation is neither constitutive of nor sufficient for mind', it is not one that Searle has established by the Chinese room argument.

Nor is this rich field of counterexamples to Searle's claim merely of logical interest. It may indeed be physically possible to construct a machine which, under the idealisation of unbounded storage and unbounded computing time, can compute functions that a universal Turing machine is unable to compute. Speculation that there may be physical processes-and so, potentially, machine-operations-whose behaviour cannot be simulated by Turing machine stretches back over at least four decades (for example Da Costa and Doria 1991; Doyle 1982; Geroch and Hartle 1986; Komar 1964; Kreisel 1967, 1974; Penrose 1989, 1994; Pour-El and Richards 1979, 1981; Scarpellini 1963; Stannett 1990; Vergis et al 1986; a partial survey is given in Copeland and Sylvan 1999). And the view that the mind is some form of hypercomputer is yet to be fully explored (Copeland 1997, 2000). As Mario Bunge has remarked, the traditional computational approach 'involves a frigh tful impoverishment of psychology, by depriving it of nonrecursive functions' (Bunge and Ardila 1987: 109).[4]

4 Accelerating Turing Machines

A recent attempt by Bringsjord, Bello and Ferrucci (2001) to defend the Chinese room argument against the hypercomputational objection employs the notion of what I term an accelerating Turing machine (Copeland 1998b, 1998c, 2002b).

Recapping from my paper in UMC'98, an accelerating Turing machine is a Turing machine that operates in accordance with what I call the Russell-Blake-Weyl temporal patterning. Weyl considered a machine (of unspecified architecture) that is capable of completing an infinite sequence of distinct acts of decision within a finite time; say, by supplying the first result after 1/2 minute, the second after another 1/4 minute, the third 1/8 minute later than the second, etc. In this way it would be possible ... to achieve a traversal of all natural numbers and thereby a sure yes-or-no decision regarding any existential question about

[4] I am grateful to Jon Opie for drawing my attention to this passage.

natural numbers. (Weyl 1927: 34; English translation from Weyl 1949: 42.) It seems that this temporal patterning was first described by Russell, in a lecture given in Boston in 1914. In a discussion of Zeno's paradox of the race-course Russell said 'If half the course takes half a minute, and the next quarter takes a quarter of a minute, and so on, the whole course will take a minute' (Russell 1915: 172-3). Later, in a discussion of a paper by Alice Ambrose (Ambrose 1935), he wrote:

> Miss Ambrose says it is logically impossible [for a man] to run through the whole expansion of 1/4. I should have said it was medically impossible. ... The opinion that the phrase 'after an infinite number of operations' is self-contradictory, seems scarcely correct. Might not a man's skill increase so fast that he performed each operation in half the time required for its predecessor? In that case, the whole infinite series would take only twice as long as the first operation. (1936: 143-4.)

Blake, too, argued for the possibility of completing an infinite series of acts in a finite time:

> A process is perfectly conceivable, for example, such that at each stage of the process the addition of the next increment in the series $1/2, 1/4, 1/8, \ldots$ should take just half as long as the addition of the previous increment. But ... then the addition of all the increments each to each shows no sign whatever of taking forever. On the contrary, it becomes evident that it will all be accomplished within a certain definitely limited duration. ... If, e.g., the first act ... takes 1/2 second, the next 1/4 second, etc., the [process] will ... be accomplished in precisely one second. (1926: 650-51.)

Accelerating Turing machines are Turing machines that perform the second operation called for by the program in half the time taken to perform the first, the third in half the time taken to perform the second, and so on. Let the time taken to perform the first operation be one 'moment'. Since $1/2 + 1/4 + 1/8 + ... + 1/2n + 1/2n + 1 + \ldots$ is less than 1, an accelerating Turing machine-or human computer-can perform infinitely many operations before two moments of operating time have elapsed. Accelerating Turing machines are able to perform tasks commonly regarded as impossible for Turing machines, for example, solve the Turing-machine halting problem, decide the predicate calculus, and determine whether or not the decimal representation of π contains n consecutive $7s$, for any n.

Stewart (1991: 664-5) gave a cameo discussion of accelerating Turing machines. Related to accelerating Turing machines are the anti de Sitter machines of Hogarth (1994, 1992) (concerning which see also Earman and Norton 1993, 1996) and the Zeus machines of Boolos and Jeffrey (1980: 14-15). (Boolos and Jeffrey envisaged Zeus being able to act so as to exhibit (what is here called) the Russell-Blake-Weyl temporal patterning (1980: 14). By an extension of terminology (which Boolos and Jeffrey did not make) a Zeus machine is any machine exhibiting the Russell-Blake-Weyl temporal patterning.) Also related are

the trial-and-error machines of Putnam (1965) and Gold (1965). Hamkins and Lewis (2000) give a mathematical treatment of the computability theory associated with accelerating Turing machines (which they term 'infinite-time Turing machines'). Sorensen (1999, sect. 6) contains a discussion of accelerating Turing machines based on Copeland (1998b).

5 The Zeus Room

Bringsjord, Bello and Ferrucci attempt to defend the Chinese room argument against my hypercomputational objection in the following way. Clerk (or Searle) can hand-work the program of an O-machine, they say, by accelerating in the Russell-Blake-Weyl fashion. Jack Copeland ... has recently argued that oracle machines ... are immune to Searle's (1980) CRA-based claim that mere symbol manipulation is insufficient for genuine mentation ... Unfortunately, oracle machines are not immune to the Chinese room ... It's easy to imagine that Searle in the Chinese Room manipulates symbols in order to parallel the operation of a Zeus machine ... The only difference is that in (what we might call) the "Zeus room," Searle works much faster. But this speed-up makes no difference with respect to true understanding.

The problem with this effort to shore up Searle's argument is that it leaves what is supposed to go on in the Zeus Room radically under-described. In order to get a feel for the difficulties, let us consider an accelerating machine H that supplies the values of the halting function.

The famous halting function H takes pairs of integers as arguments and returns the value 0 or 1. H may be defined as follows, for any pair of integers x and y : $H(x, y) = 1$ if and only if x represents the program of a Turing machine that eventually halts if set in motion with the number y inscribed on its otherwise blank tape; $H(x, y) = 0$ otherwise. (I shall refer to x as the 'program number' and y as the 'data number'.) A machine able to 'solve the halting problem' can inform us, concerning any given Turing machine, whether or not $H(x, y) = 1$. An accelerating Turing machine is able to do this. Let t be any Turing machine. A universal Turing machine that is set in motion bearing the data number formed by writing out the digits of t's program number p followed by the digits of t's data number d will simulate t. The universal machine performs every operation that t does, in the same order as t (although interspersed with sequences of operations not performed by t), and halts just in case t does. A machine H that computes the values of the halting function for all integers x and y will result if one equips an accelerating universal Turing machine (or a human clerk) with a signalling device-a hooter, say-such that the hooter blows when and only when the machine halts. If the hooter blows within two moments of the start of the simulation then $H(p, d) = 1$, and if it remains quiet then $H(p, d) = 0$.[5]

[5] Since, by definition, $H(x, y)$ is 0 whenever x is not the program number of a Turing machine, H must not halt and blow its hooter in this case. This is achieved by adding some instructions that drive Hinto an infinitely repeated loop if it determines x not to be the program number of some Turing machine.

As Thomson (1954, 1970) emphasized, if accelerating machines are to be taken seriously, consistent answers must be supplied to questions about the condition of the machine after it has completed its super-task. Such questions are highly pertinent to the description of the Zeus Room.

An example of a relevant Thomsonesque question is: Where is H's scanner positioned once the simulation of t is completed? Originally, Thomson claimed to have an argument showing that such questions could not be given consistent answers (1954). His argument was, however, established to be invalid (Benacerraf, 1962). Thomson withdrew the claim that such questions do not admit of consistent answers, saying only that any adequate treatment of the subject matter must confront such questions squarely (1970). Probably he thought that proponents of accelerating machines would prove unable to produce good answers. In fact, answers can be found (Copeland 2002b). But I cannot claim to know of any that might fully satisfy Thomson. Here is an example.

The Newtonian equations of motion have what are known as 'escape solutions' whereby a body disappears from the universe in consequence of its velocity increasing without bound (Geroch 1977: 82, Earman 1986: 34). For example, let the velocity of a body increase in accordance with the Russell-Blake-Weyl formula: takes 1 second to cover the first metre of its trajectory, a 1/2 second to cover the next metre, and so on. The velocity is always finite but is unbounded. At the end of the second second of motion disappears from the universe. Take any Euclidean coordinate system originated on any point of the trajectory: the axes specify every location in the universe and is to be found at none of them! Let H's scanner move in tandem with a tape generator which supplies a further square of tape each time the scanner reaches the end of the strip of tape already generated (perhaps the tape generator is bolted to the scanner). Each square of the tape is of side 1 unit. The user holds the free end of the tape and presses the start button. With each move-right called for by the program, the scanner travels one unit further from the user along a linear trajectory. In the case where the simulated machine t does not halt, the user is left holding an endless tape on which can be found each digit printed by t. The answer to the Thomsonesque question 'Where is H's scanner at that point?' is: Nowhere.

Bringsjord, Bello and Ferrucci owe us answers to some difficult Thomsonesque questions. How is it with Clerk after the period of acceleration? Suppose the program that Clerk is following calls for H to run twice, with different inputs, or even infinitely many times-how is Clerk supposed to manage this? How is H's scanner to be brought back to the starting position again?

It is not certainly not enough that Clerk carry out the simulation of H and sail out of the universe in the way just described, a hoot, if there is one, returning to the waiting crowd. One way or another, Clerk must be there in order to participate in the necessary facts. If the Chinese room argument is not to be subverted by the new twist to the story, it must be a fact about Clerk that, having carried out all the symbol-manipulations called for by the program, he or she does not understand Chinese. Moreover, this fact must be epistemologically accessible to us.

There is an ascending hierarchy of symbol-manipulators. At the bottom of the hierarchy lie the Turing machines, the zeroth-order machines. A level higher are the first-order machines: machines able to deliver the values of the Turing-machine halting function. One level higher still are machines able to deliver the values of the halting function for first-order machines, and so on ever upward through the orders. Thomsonian difficulties multiply if the machine that Clerk is supposed to mimic lies not at the first rung of this infinite hierarchy, as H does, but higher up. Bringsjord, Bello and Ferrucci do not mention this case at all, yet it is crucial to their attempt to defend the 'CRA-based claim that mere symbol manipulation is insufficient for genuine mentation'. They must demonstrate-to good Thomsonian standards-that Clerk is able to simulate not only H but every programmed symbol-manipulator in the ascending hierarchy.

References

1. Ambrose, A. (1935), 'Finitism in Mathematics (I and II)', Mind, 35: 186-203 and 317-40.
2. Benacerraf, P. (1962), 'Tasks, Super-Tasks, and the Modern Eleatics', Journal of Philosophy, 59: 765-84.
3. Blake, R.M. (1926), 'The Paradox of Temporal Process', Journal of Philosophy, 23: 645-54.
4. Boolos, G.S., Jeffrey, R.C. (1980), Computability and Logic, 2nd edition, Cambridge: Cambridge University Press.
5. Bringsjord, S., Bello, P., Ferrucci, D. (2001) 'Creativity, the Turing Test, and the (Better) Lovelace Test', Minds and Machines, 11, no. 1.
6. Bunge, M,, Ardila, R. (1987), Philosophy of Psychology, New York: Springer-Verlag.
7. Church, A. (1936a), 'An Unsolvable Problem of Elementary Number Theory', American Journal of Mathematics, 58: 345-363.
8. Church, A. (1936b), 'A Note on the Entscheidungsproblem', Journal of Symbolic Logic, 1: 40-41.
9. Church, A. (1937), Review of Turing (1936), Journal of Symbolic Logic, 2: 42-43.
10. Copeland, B.J. (1993a), Artificial Intelligence: a Philosophical Introduction, Oxford: Blackwell.
11. Copeland, B.J. (1993b), 'The Curious Case of the Chinese Gym', Synthese, 95: 173-86.
12. Copeland, B.J. (1997), 'The Broad Conception of Computation', American Behavioral Scientist, 40: 690-716.
13. Copeland, B.J. (1998a), 'Turing's O-machines, Penrose, Searle, and the Brain', Analysis, 58: 128-138.
14. Copeland, B.J. (1998b), 'Even Turing Machines Can Compute Uncomputable Functions'. In Calude, C., Casti, J., Dinneen, M. (eds) (1998), Unconventional Models of Computation, London: Springer-Verlag: 150-164.
15. Copeland, B.J. (1998c), 'Super Turing-Machines', Complexity, 4: 30-32.
16. Copeland, B.J. (2000), 'Narrow Versus Wide Mechanism', Journal of Philosophy, 96: 5-32.
17. Copeland, B.J. (2002a), 'The Chinese Room from a Logical Point of View'. In Preston, J., Bishop, M. (eds) Views into the Chinese Room: New Essays on Searle and Artificial Intelligence, Oxford: Oxford University Press.

18. Copeland, B.J. (2002b), 'Accelerating Turing Machines', Minds and Machines, Special Issue on Effective Procedures (forthcoming).
19. Copeland, B.J., Proudfoot, D. (1999), 'Alan Turing's Forgotten Ideas in Computer Science', Scientific American, 280 (April): 76-81.
20. Copeland, B.J., Sylvan, R. (1999), 'Beyond the Universal Turing Machine', Australasian Journal of Philosophy, 77: 46-66.
21. da Costa, N.C.A., Doria, F.A. (1991), 'Classical Physics and Penrose's Thesis', Foundations of Physics Letters, 4: 363-73.
22. Doyle, J. (1982), 'What is Church's Thesis?' Laboratory for Computer Science, MIT.
23. Earman, J. (1986), A Primer on Determinism, Dordrecht: D. Reidel.
24. Earman, J., Norton, J.D. (1993), 'Forever is a Day: Supertasks in Pitowsky and Malament-Hogarth Spacetimes', Philosophy of Science, 60: 22-42.
25. Earman, J., Norton, J.D. (1996), 'Infinite Pains: The Trouble with Supertasks'. In Morton, A., Stich, S.P. (eds) (1996), Benacerraf and his Critics, Oxford: Blackwell.
26. Geroch, R. (1977), 'Prediction in General Relativity'. In J. Earman, C. Glymour, J. Stachel (eds) (1977), Foundations of Space-Time Theories, Minnesota Studies in the Philosophy of Science, 8, Minneapolis: University of Minnesota Press.
27. Geroch, R., Hartle, J.B. (1986), 'Computability and Physical Theories', Foundations of Physics, 16: 533-550.
28. Gold, E.M. (1965), 'Limiting Recursion', Journal of Symbolic Logic, 30: 28-48.
29. Hamkins, J.D., Lewis, A. (2000), 'Infinite Time Turing Machines', Journal of Symbolic Logic, 65: 567-604.
30. Haugeland, J. (1985), Artificial Intelligence: the Very Idea, Cambridge, Mass.: MIT Press.
31. Hogarth, M.L. (1992), 'Does General Relativity Allow an Observer to View an Eternity in a Finite Time?', Foundations of Physics Letters, 5: 173-181.
32. Hogarth, M.L. (1994), 'Non-Turing Computers and Non-Turing Computability', PSA 1994, 1: 126-38.
33. Kleene, S.C. (1967), Mathematical Logic, New York: Wiley.
34. Komar, A. (1964), 'Undecidability of Macroscopically Distinguishable States in Quantum Field Theory', Physical Review, second series, 133B: 542-544.
35. Kreisel, G. (1967), 'Mathematical Logic: What has it done for the philosophy of mathematics?' In Bertrand Russell: Philosopher of the Century, ed. R. Schoenman, London: George Allen and Unwin.
36. Kreisel, G. (1974), 'A Notion of Mechanistic Theory', Synthese, 29: 11-26.
37. McLeod, P., Dienes, Z. (1993), 'Running to Catch the Ball', Nature, 362: 23.
38. Penrose, R. (1989), The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics, Oxford: Oxford University Press.
39. Penrose, R. (1994), Shadows of the Mind: A Search for the Missing Science of Consciousness, Oxford: Oxford University Press.
40. Pour-El, M.B., Richards, I. (1979), 'A Computable Ordinary Differential Equation Which Possesses No Computable Solution', Annals of Mathematical Logic, 17: 61-90.
41. Pour-El, M.B., Richards, I. (1981), 'The Wave Equation With Computable Initial Data Such That Its Unique Solution Is Not Computable', Advances in Mathematics, 39: 215-239.
42. Putnam, H. (1965), 'Trial and Error Predicates and the Solution of a Problem of Mostowski', Journal of Symbolic Logic, 30: 49-57.
43. Russell, B.A.W. (1915), Our Knowledge of the External World as a Field for Scientific Method in Philosophy, Chicago: Open Court.

44. Russell, B.A.W. (1936), 'The Limits of Empiricism', Proceedings of the Aristotelian Society, 36: 131-50.
45. Scarpellini, B. (1963), 'Zwei unentscheitbare Probleme der Analysis', Zeitschrift für mathematische Logik und Grundlagen der Mathematik, 9: 265-289.
46. Searle, J. (1980), 'Minds, Brains, and Programs', Behavioural and Brain Sciences, 3: 417-424, 450-456.
47. Searle, J. (1989), Minds, Brains and Science, London: Penguin.
48. Searle, J. (1990), 'Is the Brain's Mind a Computer Program?', Scientific American, 262 (1): 20-25.
49. Searle, J. (1992), The Rediscovery of the Mind, Cambridge, Mass.: MIT Press.
50. Searle, J. (1997), The Mystery of Consciousness, New York: New York Review.
51. Sorensen, R. (1999), 'Mirror Notation: Symbol Manipulation without Inscription Manipulation', Journal of Philosophical Logic, 28: 141-164.
52. Stannett, M. (1990,) 'X-machines and the Halting Problem: Building a Super-Turing Machine', Formal Aspects of Computing, 2: 331-341.
53. Sterelny, K. (1990), The Representational Theory of Mind, Oxford: Blackwell.
54. Stewart, I. (1991), 'Deciding the Undecidable', Nature, 352: 664-5.
55. Thomson, J.F. (1954), 'Tasks and Super-Tasks', Analysis, 15: 1-13.
56. Thomson, J.F. (1970), 'Comments on Professor Benacerraf's Paper'. In W.C. Salmon (ed.), Zeno's Paradoxes, Indianapolis: Bobbs-Merrill.
57. Turing, A.M. (1936), 'On Computable Numbers, with an Application to the Entscheidungsproblem', Proceedings of the London Mathematical Society, series 2, 42 (1936-37): 230-265.
58. Turing, A.M. (1938), 'Systems of Logic Based on Ordinals'. Dissertation presented to the faculty of Princeton University in candidacy for the degree of Doctor of Philosophy. Published in Proceedings of the London Mathematical Society, 45 (1939): 161-228.
59. Turing, A.M. (1950), 'Programmers' Handbook for Manchester Electronic Computer', University of Manchester Computing Laboratory. A digital facsimile of the original is available in The Turing Archive for the History of Computing at `http://www.AlanTuring.net/programmers_handbook`.
60. Vergis, A., Steiglitz, K., Dickinson, B. (1986), 'The Complexity of Analog Computation', Mathematics and Computers in Simulation, 28: 91-113.
61. Weyl, H. (1927), Philosophie der Mathematik und Naturwissenschaft, München: R. Oldenbourg.
62. Weyl, H. (1949), Philosophy of Mathematics and Natural Science, Princeton: Princeton University Press.

Very Large Scale Spatial Computing

André DeHon

California Institute of Technology
Department of Computer Science, 256-80, Pasadena, CA 91125, USA
andre@acm.org

Abstract. The early decades of computing were marked by limited resources. However, as we enter the twenty-first century, silicon is offering enormous computing resources on a single die and molecular-scale devices appear plausible offering a path to even greater capacities. Exploiting the capacities of these modern and future devices demands different computational models and radical shifts in the way we organize, capture, and optimize computations. A key shift is toward spatially organized computation. A natural consequence is that the dominant effects which govern our computing space change from the total number of operations and temporal locality to interconnect complexity and spatial locality. Old computational models which hide, ignore, or obfuscate communication and emphasize temporal sequences inhibit the exploitation of these modern capacities, motivating the need for new models which make communication and spatial organization more apparent.

1 Introduction

Severely limited physical capacity has been a stark reality of computer design from the 1940's. In the pre-VLSI era, we were limited by the physical bulk and cost of relays, vacuum tubes, and discrete transistors. In the early VLSI era, we were limited by the capacity of a single chip. Consequently, practical computing devices have been organized around clever ways to use small amounts of hardware to solve large problems.

The advent of compact memory technologies (core memory, IC memories, DRAMs) coupled with the observation that we could describe a computation and its state compactly, allowed us to reduce the size of a computation by time-multiplexing the active hardware across many operations in the computation. The most familiar embodiment of this is the sequential *processor*, where we use a few tens of bits to store each instruction, a large memory to store the state of the computation, and a single, or small number, of active computing devices to evaluate the large computation sequentially.

Our conventional abstractions for describing computations (the Instruction Set Architecture (ISA) at the machine code level, sequential programming languages like C, FORTRAN, and Java at the programmer level) grew out of this poverty. They allowed programmers and compilers to describe computations in a manner cognizant of the capacity limitations of these devices. They focused

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 27–37, 2002.

the developer on the costly items for these machines: the operations and the use of memory. Programs were optimized by reducing the number of operations and minimizing the amount of live state.

As we are all aware, the steady shrink of feature sizes in integrated circuit processing has produced more capacity per die and per dollar at an exponential rate. Our computing devices have been steadily getting richer. While we may see the end of exponential IC feature size scaling in the next couple of decades [12], advances in basic science are suggesting ways to cheaply engineer computing devices at the molecular scale and in three dimensions.

With exponentially increasing capacity, we inevitably reach a point where resource capacity is no longer such a scarce capacity. That point may vary from task to task, and there are some problems that may always feel the crunch of limited resources—particularly problems we cannot attack today even with all of our well-honed tricks to reduce system size. Nonetheless, for a larger and larger set of problems, we are crossing the poverty threshold to the world of abundance.

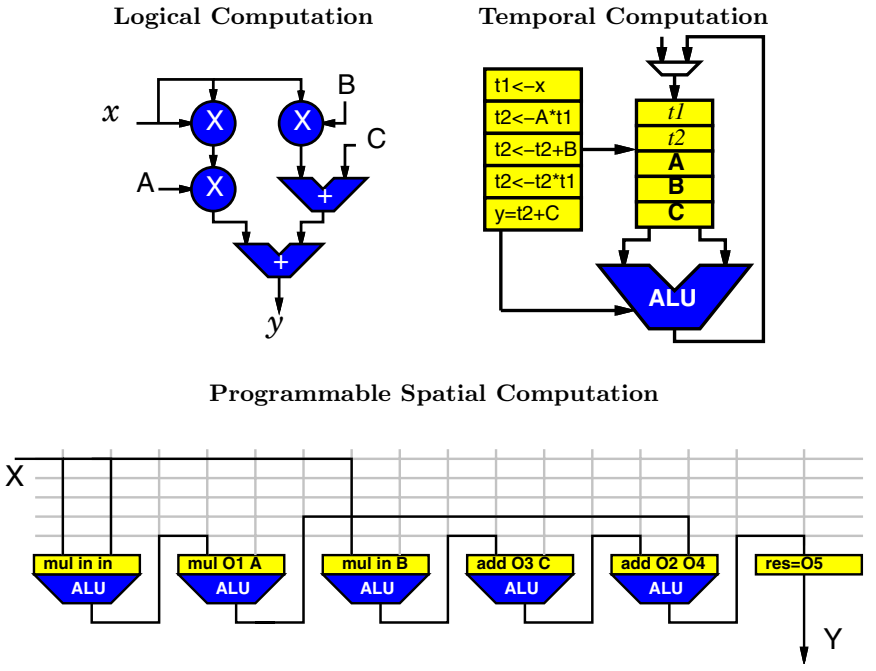
By 2016, for example, silicon technology is promising 900 million gates in $2.3 \times 10^{12}\lambda^2$ of silicon [1]. However, we were already building quite competent single-chip processors by the early 1990's with less than $2 \times 10^9\lambda^2$ of silicon (*e.g.* [6] [23]). Various estimates suggest we may be able to achieve 10^{10} gates/cm^2 using molecular scale electronics [7] in only two dimensions; that is another 30× the density of 2016 silicon.

What happens when we have an abundance of resources?

Minimal existence has the advantage that it certainly continues to work in times of plenty. But, it is not necessarily the best or more efficient way to implement a computation when more resources exist. It is this problem which computing has, for good and practical reasons, largely ignored for the past 50 years. The current and future prospects for high capacity devices and systems suggest it is now practical, beneficial, and necessary to rethink the way we formulate computations. The abstractions, metrics, and optimizations which suited capacity poor systems can be quite inappropriate when resources are less limited. Consequently, we now need to find new models, optimizations, and algorithms suitable to exploit our growing wealth of computational capacity.

2 Trend toward Spatial Computing

When capacity is abundant, we are not driven to the temporal extreme where we time-multiplex a single active computation among a large number of operators. At the opposite extreme, we give each operation its own active operator. Operations are interconnected in *space* rather than *time* (See Figure 1). Notably, this means we exploit the full parallelism available in the task, completing the task in time proportional to the longest path in the computation rather than in time proportional to the number of operations. In fact, for many designs, we may be able to pipeline the computation and produce new results every unit of operator delay.



In spatial implementations, each operator exists at a different point in space, allowing the computation to exploit parallelism to achieve high throughput and low computational latencies. In temporal implementations, a small number of compute resources are reused in time, allowing the computation to be implemented compactly.

Fig. 1. Spatial versus Temporal Computation for the expression $y = \mathrm{A}x^2 + \mathrm{B}x + \mathrm{C}$

The spatial design is larger than the minimum size temporal design, trading increased area to hold more active operators for decreased time. As area becomes less of a limiting factor in designs, we can accelerate our computation by moving to more spatial designs. Further, even programmable spatial design are more effective at exploiting high capacity than both conventional processors and multiprocessors.

Spatial designs were originally the sole domain of custom silicon application (*e.g.* custom VLSI or ASICs for dedicated signal processing tasks, video compression). They were only used for limited tasks which required computational rates infeasible with time-multiplexed, programmable designs. As capacity has grown, programmable spatial designs have become increasingly practical using Field-Programmable Gate-Arrays (FPGAs). With the capacity available in today's FPGAs, many computing tasks can be implemented entirely spatially in a single device (*e.g.* digital filters, video and cryptographic encoding and decoding). It is worthwhile to note that the size of the FPGA device is often equivalent to

the size of the sequential processor, while providing orders of magnitude greater performance [3].

Conventional, sequential processors struggle to extract more parallelism from tasks described using the capacity poor sequential ISA abstraction. The abstraction, itself, however limits and obfuscates available parallelism. Supporting the abstraction, requires substantially more hardware capacity (*e.g.* renaming, issue logic, reorder buffers) for model overhead than the capacity which is applied to the computational task.

Unlike conventional parallel processors, which are also based around heavily time-multiplexed processing nodes using the ISA abstraction, spatial computations exploit *regularity* in the computing task to build more compact, active processing nodes. A spatial operator can be built or programmed to perform the *same* operation repeatedly on a sequence of data. In the custom hardware world, this means we build a hardwired datapath that does just one thing, but is used efficiently because we need to do that one thing over and over again. In the FPGA or programmable spatial world, this means we factor out the components of a computation which are needed repeatedly and configure a spatial unit to compute them efficiently. Signal processing, encryption, compression, and image manipulation are common examples of applications which require repeated application of the same computation to a large set of data. More generally the oft-quoted 90/10 rule suggests that large portions of typical computations (90% of the dynamic operation count) require the repeated application of identical or similar computational functions (10% of the static task description). In contrast, parallel processing nodes still require a large investment in memory to describe and hold state for a large number of different operations. As a result, a spatial processing operator is less expensive than a small sequential processing node. This, in part, is how an FPGA-like device can often require as little area as a sequential processing device while offering orders of magnitude greater performance.

3 Interconnect Dominance

As we exploit more spatial parallelism by employing more active, communicating processing elements, we must interconnect a greater number of components. For these system sizes and relative delays, interconnect issues are of paramount importance as interconnect can quickly becomes the dominant resource in the design consuming area, delay, and energy.

If we simply placed operators randomly onto the available die space, with high likelihood about half of the operators on the left half of the chip will want an input from the right half of the chip, and vice-versa. That means, we have around N/2 wires crossing the middle of the chip from left to right. We can make a similar argument from top to bottom. Consequently, given a fixed number of wiring layers, it will require a chip of size $O(N^2)$ simply to handle N communicating operators. In such a random placement, the average distance between connected operators is 1/3 of the length of the chip in each dimension.

In the past we could ignore the effect of distance on delay because our components were too small for interconnect delay across a chip to be a dominant delay component. With growing chip capacity and shrinking feature sizes, the minimum delay across a chip is now large compared to desirable computational cycles [1]. This is especially troubling in light of the observation above that average communication distances for random placement could be 2/3 of the length of the die side. Since we know that delay is a function of distance with fundamental limits on the speed of propagation (*e.g.* speed of light), the fact that distance between operators implies delays should not be surprising. It was only the particular size and gate delay constants in early computing technologies that allowed us to ignore this effect for so long. By the 45nm node in the ITRS Roadmap [1], it looks like we will only be able to reach a radius of 200 custom 2-input gates [24] or 10 programmable bit operators without making interconnect a large fraction of gate-to-gate delay. This suggests we can, at most, travel a distance of a several thousand custom gates or a several hundred programmable bit operators in a single cycle on an optimally buffered wire. Since our systems will be large compared to these radii, it becomes important both to layout computations to minimize communication distances (See Section 6) and to pipeline distant interconnect as in our HSRA [26].

4 New Abstractions

To serve our increasingly spatial, increasingly communication dominated, computations, we need compute models which emphasize the parallelism and communication which occurs in these devices.

Conventional models hide communication by layering it on top of memory operations to a single, monolithic memory space; this makes it hard to rediscover the links which exist between operators and impossible to identify with certainty all the links which may exist. Using memory for communication was necessary to communicate data compactly in capacity poor, time-multiplexed systems; it also facilitated a number of optimization, such as memory location reuse, which allowed one to reduce the amount of capacity needed to evaluate a computation. However, these descriptions do not facilitate efficient spatial computations.

For spatial computation, graph-based computing models offer a number of important advantages over sequential models. Most notably, they support parallelism and make communication links between operators explicit. A number of graph-based computing models exist, dating at least back to Kahn [13]. We summarize a number of such models in [2] and introduce SCORE, our own version which attempts to pull the best ideas from many sources and provide a suitable model for today's spatial computing.

In SCORE, the computation is a graph of operator nodes connected by persistent dataflow links, or *streams*. The graph can be of arbitrary size and may evolve during the computation. Once a graph or subgraph is created it operates on the data it is given through its input streams, producing new results to its output streams. Data on the streams is tagged with presence, provid-

ing deterministic, timing-independent behavior for the graph. Operators may be composed hierarchically.

In addition to exposing parallelism and communications, the persistent operators or subgraphs help capture the regularity which exists in the computation. Heavily used subgraphs with long persistence merit spatial implementations, whereas transient or infrequently used subgraphs may still benefit from temporal implementations. The division point between the spatial and temporal domain can depend on the available capacity and desired performance.

5 Architectures

Armed with these new abstractions, we can envision a class of spatial architectures suitable for this Very Large Scale Spatial Computing domain. At the highest level, we might imagine an arbitrarily large array of computation and memory nodes supported by suitable interconnect.

To manage this space cleanly, we organize the computing operators into a series of *compute pages* and the memory into a set of *memory blocks* (See Figure 2). A compute page, for example, might hold a few hundreds or thousands of programmable bit operators or a few tens or hundreds of configurable datapath elements. The page organization has several advantages, including:

- It allows us to manage computations in modest size aggregates similar to the way we manage memory in pages in virtual memory systems. In both cases, this reduces the overhead in both time and space required to manage the mapping between virtual and physical resources.
- If we need to virtualize the physical space, pages become the unit of virtualization.
- The page size can be chosen relative to the technology so that intra-page communication can occur within an aggressive compute cycle while inter-page communications is pipelined to accommodate the greater distances.

As long as the compute pages obey the streaming dataflow communications discipline identified in the compute model, their microarchitecture becomes irrelevant to the rest of the computation. This makes it possible to build heterogeneous devices which include a wide range of spatial and temporal processing nodes (See Figure 2). The mix allows us to support the low-frequency and uncommon portions of the computation compactly in temporal form, while supporting the dominant, regular computation efficiently with a spatial implementations. The composition and mix of node types can evolve with the capacity offered by the technology while always supporting a single, unifying compute model.

6 Optimizing Spatial Computations

With abundant capacity such that it is not always necessary to time-multiplex operators to fit within available capacity, the key optimization will be to reduce the maximum *distance* along critical paths and loops. Area reduction does play a role here to the extent it makes designs more compact and reduces the distances

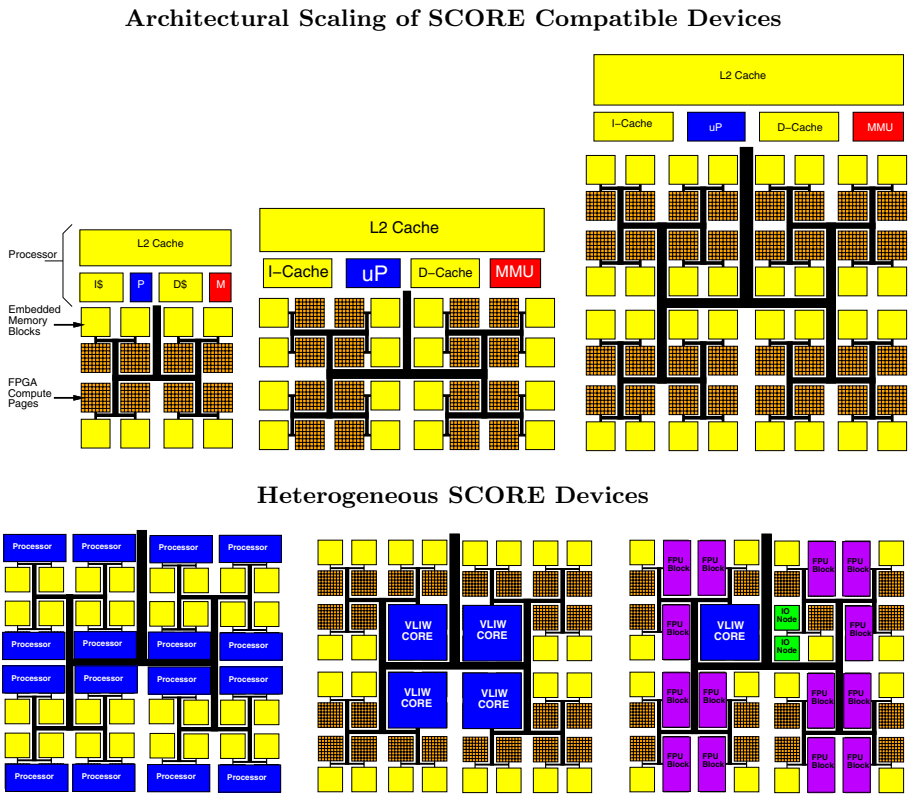


Fig. 2. Compute Model Facilitates Heterogeneous Compute Pages and Component Scaling

over which critical signals must travel. Commonly communicating blocks should be arranged to optimize spatial locality. The presence of communication links in the design representation is vital to enabling these kinds of optimizations.

Area. Our first concern is to place computations to reduce requisite wiring and switching. This is the traditional domain of placement and can reduce wiring requirements from the $O(N^2)$ area identified above to $O(N^{2p})$ [5], where p is the exponent in Rent's Rule [16] and typically has a value around 2/3. With sufficient wiring layer growth, it may be possible to even contain the two-dimensional active area to $O(N)$ [4].

We have some control in the design of our algorithms over the interconnect richness, p. It will be important to explore the extent to which we can design or optimize computing structures to reduce p. In the early days of VLSI, the systolic design style [14] focused on planar structures with a p of 1/2 such that interconnect scaled conveniently with computation. With interconnect area and delays becoming the truly dominating effects, these ideas may now attain even

greater practical importance than when they were first introduced. Logic replication, which has long been used to minimize wires crossing chip boundaries [20] [11], may play an important role here as well.

Space and Time. A key optimization will be understanding how much to fold and unfold computations to match computational throughput rates, fit within the available area, and minimize total path delay distances. Time multiplexed designs require more area between active computing operations but allow entire computations to be compact. Sometimes it will still be desirable to make a computation compact in this manner in order to reduce the distance required to communicate to it or across it. This should be quite evident for off-critical path computations, operators in loops with large cycle bounds, and operations with low relative operating frequency. An interesting question for the future will be: when it is faster to time-multiplex even a critical path computation in order to reduce the size of the circuit and hence the distance which the signals must travel?

Spatial Locality. With interconnect playing such a dominant role, we want to optimize the *spatial locality* of heavily communicating blocks. Some of this will arise out of the area and path delay minimizing optimization described above. However, we can do better in a number of important cases.

The computation may be composed of many cyclic subgraphs connected together, perhaps even nested in larger cycles of computation. The distance around each cycle will often limit the rate of computation. Consequently, it is important to place each cycle as compactly as possible. Techniques such as cycle partitioning and replication [21] may be important here. In many cases, we can cluster cycles tightly at the expense of increasing the distances between the clustered cycle and the rest of the graph; to the extent these links in and out of the cycle are not themselves on critical loops, they can be pipelined to accommodate the additional delay without adversely impacting the computation. Capturing this kind of timing freedom is an important function of the computational model.

Communication among operators will often be dynamic and data dependent. When not all communications are equally likely, we have the opportunity to preferentially cluster the blocks involved in common communications more closely than less commonly communicating blocks. In the sequential processor world, people have long exploited instruction placement to increase spatial locality and hence virtual memory [9] and cache performance [25]. Here, we need to cluster operations into spatial clusters to reduce the distance delay along the most commonly used communication paths.

7 Spatial Algorithms

The algorithms suitable for spatially organized computations may be different from the ones we have found suitable for temporal computations. The work in systolic architectures provides a number of important algorithms, such as:

matrix multiplication [15], dynamic programming [8], sorting [17], and image processing. In many cases these algorithms are specifically designed to minimize and regularize interconnect. Contemporary work in FPGA computation has offered a number of additional spatial algorithms, including: sequence matching (dynamic programming) [10], satisfiability [28] and set covering [19] search, regular expression matching [22], and image processing [27] [18]. These algorithms demonstrate the power that comes from properly reformulating algorithms for spatial computation. The development and understanding of new spatial algorithms will be an important component of understanding how to exploit the rich capacities available to us. In the capacity poor past, the kinds of techniques used in spatial algorithms were almost unfathomable compared to the resources available; today and tomorrow they may be essential.

8 Conclusions

We are rapidly entering a future where the capacity of our basic computing media is more than sufficient to contain a significant fraction of our computing problems. This opens up a much larger range of implementation options for us. However, the computing models and abstractions developed during the era of limited capacity make it difficult for us to exploit the capacity now available. Spatial computing architectures and designs offer a promising alternative to temporal organization that can better exploit the rich capacity becoming available. In these spatial designs, communication is a first order concern. Consequently, it is important to develop models that expose communications for optimization and to develop algorithms which are conscious of the costs and effects of communications in order to fully exploit the performance potential of modern and future capacity rich devices.

References

1. International Technology Roadmap for Semiconductors. <http://public.itrs.net/Files/2001ITRS/>, 2001.
2. Eylon Caspi, Michael Chu, Randy Huang, Nicholas Weaver, Joseph Yeh, John Wawrzynek, and André DeHon. Stream Computations Organized for Reconfigurable Execution (SCORE): Introduction and Tutorial. , short version appears in FPL'2000 (LNCS 1896), 2000.
3. André DeHon. The Density Advantage of Configurable Computing. *IEEE Computer*, 33(4):41–49, April 2000.
4. André DeHon. Compact, Multilayer Layout for Butterfly Fat-Tree. In *Proceedings of the Twelfth ACM Symposium on Parallel Algorithms and Architectures (SPAA'2000)*, pages 206–215. ACM, July 2000.
5. André DeHon. Rent's Rule Based Switching Requirements. In *Proceedings of the System-Level Interconnect Prediction Workshop (SLIP'2001)*, pages 197–204. ACM, March 2001.

6. Daniel Dobberpuhl, Richard Witek, Randy Allmon, Robert Anglin, Sharon Britton, Linda Chao, Robert Conrad, Daniel Dever, Bruce Gieseke, Gregory Hoeppner, John Kowaleski, Kathryn Kuchler, Maureen Ladd, Michael Leary, Liam Madden, Edward McLellan, Derrick Meyer, James Montanaro, Donald Priore, Vidya Rajagopalan, Sridhar Samudrala, and Sribalan Santhanam. A 200MHz 64b Dual-Issue CMOS Microprocessor. In *1992 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pages 106–107. IEEE, February 1992.
7. Seth Copen Goldstein and Mihai Budiu. NanoFabrics: Spatial Computing Using Molecular Electronics. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 178–189, June 2001.
8. L. J. Guibas, H. T. Kung, and C. D Thompson. Direct VLSI Implemenation of Combinatorial Algorithms. In *Caltech Conference on VLSI*, pages 509–525, January 1979.
9. D. J. Hatfield and J. Gerald. Program Restructuring for Virtual Memory. *IBM Systems Journal*, 10(3):168–192, 1971.
10. Dzung T. Hoang. Searching Genetic Databases on Splash 2. In Duncan A. Buell and Kenneth L. Pocek, editors, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 185–191, Los Alamitos, California, April 1993. IEEE Computer Society, IEEE Computer Society Press.
11. L. James Hwang and Abbas El Gamal. Min-Cut Replication in Partitioned Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):96–106, January 1995.
12. Guest Editor James Meindl. Special Issue on Limits of Semiconductor Technology. *Proceedings of the IEEE*, 89(3):223–393, March 2001.
13. Gilles Kahn. The Semantics of a Simple Language for Parallel Programming. In *Proceedings of the IFIP CONGRESS 74*, pages 471–475. North-Holland Publishing Company, 1974.
14. H. T. Kung. Why Systolic Architectures? *IEEE Computer*, 15(1):37–46, January 1982.
15. H. T. Kung and Charles E. Leiserson. Systolic Arrays (for VLSI). In *Proceedings of 1978 Sparse Matrix Conference*, pages 256–282. Society for Industrial and Applied Mathematics, 1979.
16. B. S. Landman and R. L. Russo. On Pin Versus Block Relationship for Partitions of Logic Circuits. *IEEE Transactions on Computers*, 20:1469–1479, 1971.
17. Charles E. Leiserson. Systolic Priority Queues. In *Proceedings of the Conference on Very Large Scale Integration: Architecture, Design, and Fabrication*, pages 199–214. California Institute of Technology, 1979.
18. Michael R. Piacentino, Gooitzen S. van der Wal, and Michael W. Hansen. Reconfigurable Elements for a Video Pipeline Processor. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, pages 82–91. IEEE, 1999.
19. Christian Plessl and Marco Platzner. Custom Computing Machines for the Set Covering Problem. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'2002)*. IEEE, 2002.
20. Roy L. Russo. On the Tradeoff Between Logic Performance and Circuit-to-Pin Ratio for LSI. *IEEE Transactions on Computers*, 21(2):147–153, February 1972.
21. Minshine Shih and Chung-Kuan Cheng. Data Flow Partitioning for Clock Period and Latency Minimization. In *Proceedings of the 31st Design Automation Conference (DAC'31)*, June 1994.

22. Reetinder Sidhu and Viktor K. Prasanna. Fast Regular Expression Matching using FPGAs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'2001)*. IEEE, 2001.
23. Kazumasa Suzuki, Masakazu Yamashina, Takashi Nakayama, Masanori Izumikawa, Masahiro Nomura, Hiroyuki Igura, Hideki Heiuchi, Junichi Goto, Toshiaki Inoue, Youichi Koseki, Hitoshi Abiko, Kazuhiro Okabe, Atsuki Ono, Youichi Yano, and Hachiro Yamada. A 500MHz 32b 0.4μm CMOS RISC Processor LSI. In *1994 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pages 214–215. IEEE, February 1994.
24. Dennis Sylvester and Kurt Keutzer. Rethinking Deep-Submicron Circuit Design. *IEEE Computer*, 32(11):25–33, November 1999.
25. Hiroyuki Tomiyama and Hiroto Yasuura. Code Placement Techniques for Cache Miss Rate Reduction. *ACM Transactions on Design Automation of Electronic Systems*, 2(4):410–429, October 1997.
26. William Tsu, Kip Macy, Atul Joshi, Randy Huang, Norman Walker, Tony Tung, Omid Rowhani, Varghese George, John Wawrzynek, and André DeHon. HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array. In *Proceedings of the International Symposium on Field Programmable Gate Arrays*, pages 125–134, February 1999.
27. John Villasenor, Brian Schoner, Kang-Ngee Chia, and Charles Zapata. Configurable Computer Solutions for Automatic Target Recognition. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 70–79. IEEE, April 1996.
28. Peixin Zhong, Margaret Martonosi, Pranav Ashar, and Sharad Malik. Accelerating Boolean Satisfiability with Configurable Hardware. In *Proceedings of the 1998 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, pages 186–195, April 1998.

The Minimum-Model DNA Computation on a Sequence of Probe Arrays

Mitsunori Ogihara[1] and Animesh Ray[2]

[1] Department of Computer Science, University of Rochester, Box 270226, Rochester, NY 14627-0226, USA. ogihara@cs.rochester.edu
[2] Keck Graduate Institute, 535 Watson Drive, Claremont, CA 91711, USA. Animesh_Ray@kgi.edu

Abstract. This paper investigates the computational power of a variant of the minimum DNA computation model proposed by Ogihara and Ray. In the variant, two fundamental operations (separation by length and digestion) are dispensed with. To compensate for the loss, the probe arrays are considered as the media for reaction and a wash operation is added to the set of permissible operations. Computation in this model necessarily consists of cycles of five steps: merge, anneal, wash, denature, and wash. It is shown that boolean circuits can be theoretically simulated repetition of the cycle on the same set of four probe arrays.

1 Introduction

DNA-based computation is a new field that integrates molecular biology and computer science. Its goal is to design methods for massively parallel computation by manipulating biological molecules. Although the idea of controlling small particles, such as DNA, was suggested as early as in the late 1950's by Richard Feynman [8], it was not until the mid 1990's that an active use of molecules was presented in the form of computation. The work of Adleman [1] is the first to show that DNA molecules are capable of computation. Adleman designed an algorithm for solving the Hamiltonian Path Problem—the problem of testing whether a given directed graph contains a Hamiltonian path, i.e. a directed, source-to-sink path that visits each node of the graph exactly once. Adleman's algorithm executes brute-force search for a Hamiltonian path, where each node is encoded as a unique ssDNA (single-stranded DNA). Based on the encoding, for each directed edge, a large quantity of the ssDNA that encodes the edge is synthesized in billions or trillions of copies. Then these "edges" are connected by the "linker" ssDNA, which has the ability of appending an edge leaving a node after an edge arriving at the node. Then, using length-based and pattern-based selections, the "correct" Hamiltonian paths are blindly extracted from the solution.

Advantages of DNA-based (or other similar molecules) computation lie in the fact that molecular operations dissipate much less energy than silicon-based computers and in the fact that molecules have much higher storage capacity than

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 38–49, 2002.

silicon transistors [1]. Adleman implemented his algorithm and successfully solved a seven-node instance of the problem [1]. However, the algorithm is not suited to handle large instances. The Avogadro number (6.02×10^{23}) is essentially an upper bound on the number of DNA molecules that one can manipulate in a single biochemical reaction, with the practical limit being several orders of magnitude below this number. Adleman analyzes that the maximum number of nodes that his algorithm can deal with is no more than 60, which is well within the reach of the silicon-based computers. Also, biochemical operations are both error-prone and slow, and little is known about the behavior of molecular reactions in the presence of extremely large numbers of molecules. These observations raise the question of whether DNA-based computation will be able to compute something the silicon-based one cannot do, which has been the driving-force of the field. Although the field has made a steady progress (see [22] and [20] for a survey; also, [3] and [4]), a definitive answer to the question has not been given.

A critical issue in the study of molecular computation is the selection of computation models, the environment in which the computation is carried out. Various computational models have been proposed in the past, including the test-tube ssDNA model that was pioneered by Adleman [1] and further extended by Lipton [15,5], the sticker DNA/PNA model proposed by Roweis et al. [28,4], the hairpin DNA model of Sakamoto et al. [29], the surface DNA model of Liu et al. [16], the enzymatic model of Rothemund [27] and of Benenseon et al. [3], and the two-dimensional DNA self-assembly model of Seeman and Winfree [33].

The objective of this paper is to explore further the minimum DNA computation model proposed and studied in our earlier papers [17,19]. The study of the minimum DNA computation model is motivated by the desire to clarify what computational power DNA has when the set of permissible operations is the smallest. The following operations are chosen for the minimum DNA model:

1. *Synthesize*: With this operation ssDNA that are used for computation are synthesized before the actual computation is conducted.
2. *Anneal*: This operation permits the ssDNA in a given solution to hybridize with each other based on their Watson-Crick complementarity.
3. *Separate*: This operation separates DNA (ssDNA or dsDNA) based on their base length.
4. *Merge*: This operation mixes two DNA solutions.
5. *Denature*: This operation pulls apart ssDNA that are forming dsDNA.
6. *Digest*: This operation digests ssDNA molecules into single DNA molecules.
7. *Detect*: This operation tests whether a solution contains ssDNA of a certain length.

It is quite natural to view molecular computation as parallel computation because many operations are carried out simultaneously by redundant copies of molecular logical units. This justifies the study of relationships between existing parallel computation models and molecular computation model. Reif [26] introduces the PAM model and shows that CREW PRAM's can be simulated by DNA computation with a small overhead. Gehani and Reif [9] suggest a computational model in which DNA strands are divided into small compartments

and strands are transfered from a compartment to another using microfluidics. Amos, Dunne, and Gibbons [2] propose a DNA-based method for simulating of NAND circuits. In [17] tight relationships between the minimum DNA model and the boolean circuit are shown. The main result of [17] is that the minimum DNA computation has exactly the same power as the boolean circuit class NC^1 if the base-length of the longest DNA compounds generated during the computation is $\mathcal{O}(\log n)$ and if the total number of legitimate compounds is $\mathcal{O}(n^k)$ for some fixed k.

In this paper we address the issue of whether DNA computation retains its computation power equivalent to that of NC^1 if two of its operations, digestion of single-stranded DNA and separation by base length, are dispensed with. To compensate for the loss, additional changes are made. First, hybridization is now assumed to take place on probe arrays, i.e. single-stranded DNA immobilized on a surface. Second, a *Wash* operation is added, which removes strands that do not hybridize with the probes. With these modifications, computation on each probe array should necessarily be in cycles of five steps:

1. *Merge* to introduce strands on the prove array;
2. *Anneal* to create double-stranded molecules;
3. *Wash* to extract strands that are not part of double stranded molecules involving a probe;
4. *Denature* to turn double-stranded compounds involving probes back to single-stranded DNA; and
5. *Wash* to extract non-probe strands.

Theoretically either a single level of AND gates or a single level of OR gates can be simulated by running a sequence of such five-step cycles on four arrays. It is also theoretically possible to simulate multiple levels of AND and OR gates using the same set of four probe arrays over and over again. The fundamental principle of the simulation method is the fact that there is a significant difference in the melting temperature between short DNA strands and those that are twice as long.

2 Boolean Circuits

A *boolean circuit* of n inputs is a directed acyclic graph with labeled nodes. There are exactly $2n$ nodes with indegree 0. These nodes are called input gates and are labeled $x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}$. Other nodes are labeled by one of the boolean operations, $\wedge$ and $\vee$, which respectively compute the conjunction and the disjunction of two boolean inputs. That is, it is assumed that the negation of each input signal is provided at the input level and there is no negation elsewhere. Since each boolean circuit is acyclic it necessarily possesses nodes with no outgoing edges. Such nodes are the *output* nodes of the circuit.

The models of boolean circuits are defined in terms of the gate fan-ins (the number of maximum inputs that $\wedge$-gates and $\vee$-gates can take). The model that this paper is concerned with is the *bounded-fan-in circuits*, in which both $\wedge$-gates

and $\vee$-gates may have at most two inputs. It is assumed that these gates are stratified so that each level consists of the same kind of gates, i.e., that is, there are levels of $\vee$-gates and those of $\wedge$-gates and so that each non-input gate takes input signals from gates at precisely one level below.

The complexity of a boolean circuit is usually measured by two quantities. The *size* is the total number of its gates, and the *depth* is the number of non-input levels. This paper also uses two additional measures. The *fan-out* is the maximum number of outgoing edges of any of its gates and the *width* is the maximum number of gates at any level.

Hoover, Klawe, and Pippenger [11] show that each bounded-fan-in boolean circuit can be converted to an equivalent bounded-fan-in, max-fan-out-two boolean circuit with a very small increase in size. So, in the following it is assumed that the circuits to be simulated are bounded-fan-in, max-fan-out-two boolean circuits.

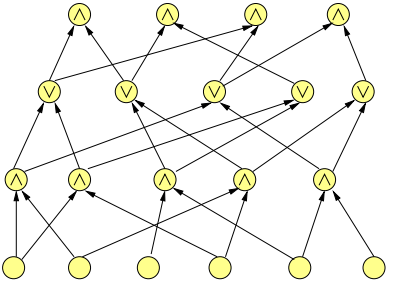


Fig. 1. A leveled, bounded-fan-in, bounded-fan-out circuit.

3 Probe Arrays

DNA is a charged copolymer of a repeating sugar-phosphate chain of indefinite length to which are attached one of four organic bases (`A`, `T`, `G`, and `C`) [32]. Each sugar-phosphate-base unit is a nucleotide. The chain is chemically and geometrically polar, with the so-called $5'$-end and the $3'$-end. A single polymer chain can have any sequence of bases from the $5'$-end to the $3'$-end. Each base has one of the four bonding surfaces, and the bonding surface of `A` is complementary to that of `T`, and of `G` to that of `C`. Because of this complementarity rule, a DNA chain can pair with another chain only when their sequences of bases are mutually complementary. Also, the chains must pair head-to-tail, i.e., antiparallel to each other. When collections of complementary single strands are mixed, heated and then slowly cooled, the complementary strands find one another and double stranded DNA is formed (the process of *annealing*) by a pseudo second order reaction kinetics. This is *DNA hybridization* [31].

Probe arrays are ssDNA molecules of defined sequences that are either synthesized *in situ* on a chemically reactive surface by photolithographic masking techniques [30] or are synthesized separately then spotted and chemically attached to the surface. Typically, the probes are at least 15 nt (i.e. bases) long, but may be as long as 70-75 nt. For 15 nt long probes, a short linker segment may be added between the anchor point on the surface and the beginning of the probe sequence to make the probe accessible to input and output strands. Several methods for chemical attachments of derivatized oligonucleotides exist. For example, amino modified oligonucleotides, with a flexible linker attached to the $5'$ phosphate to improve accessibility, can be immobilized on epoxy silane coated or isothiocyanate coated glass sides [13,10,14,25,6]. A 15 nt probe allows for at most 2^{30} ($\approx 10^9$) different probe sequences. In practice, this complexity is difficult to achieve. Ten nanogram of probe molecules, with 10^6 different sequences each 15 nt long, will contain approximately 10^6 molecules (1 attomole, or 10^{-18} mole) of each unique sequence. An attomole of DNA cannot be detected without some sequence amplification. Ideally, at least 10^3 fold amplification (by PCR or any other available technique) is needed to detect molecules by ultra sensitive techniques such as mass spectrometry. Thus, a boolean circuit of width (i.e. the largest number of gates at any level) at most 10^6 can be simulated with 10 nanogram of probe molecules on surfaces.

More commonly, only 10 picogram of oligonucleotides are spotted on an array per spot. To obtain 1 attomole of each unique sequence representation, it is necessary to have at most only 10^3 different sequences per spot. However, it is possible to make over 10^4 different spots per array. If each spot represents one level of the boolean circuit, then it is possible to compute a boolean circuit of depth 10^4 with at most 10^3 inputs. These are the limitations to the computation on probe arrays with the current technology.

4 Simulation on Probe Arrays

4.1 Basic Principles

The gates of the circuit are simulated in levels. As in the earlier work [21,18,19], at each level, a unique pattern of DNA is assigned to each gate of the boolean circuit to be simulated. Gates at different levels may share the same pattern of DNA. Similarly, at each level, a unique pattern of DNA is assigned to each edge coming out of that level.

Let g be a gate that takes inputs from two gates, a and b, and provides output to two gates, c and d. Let `x` and `y` be the ssDNA patterns associated with the wire from a to g and the wire from b to g, respectively and let `u` and `v` be the ssDNA patterns associated with the wire from g to c and the wire from g to d, respectively. Let `z` be the ssDNA pattern corresponding to g. Simulation of the gate g is divided into phases. In the first phase, using a probe and some auxiliary ssDNA, the boolean function of the gate g is simulated. In the second phase, using another probe and some other auxiliary ssDNA, the wires coming of the gate are simulated. We envision that at the beginning of the first phase the DNA

solution one the first probe contains x if and only the output of the gate a is a 1 and contains y if and only the output of the gate b. Then the simulation of g is carried out so that the strand z is present if and only if g outputs a 1. In the second phase, we attempt to produce u and v out of z so that both u and v are present at the end of second phase if z is present at the beginning of the second phase and both u and v are absent at the end of second phase if z is absent at the beginning of the second phase (see Figure 2).

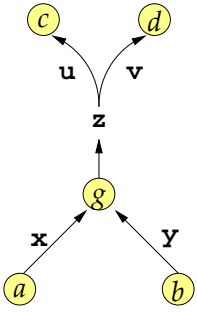


Fig. 2. Two phases of a gate simulation.

4.2 Design of and Gates and or Gates

Figures 3 and 4 depict how this principle of gate simulation is applied on AND gates and OR gates. There are four types of strands participate in the hybridization depicted in the figures. The "Probe" strands are immobilized on the surface, with which the double-stranded DNA compounds that are formed are anchored on the surface. The "Input" strands correspond to the input signals given to the gates. The "Output" strands correspond to the outputs of the gates. The "Input" strands and "Outputs" are linked side by side with the "Bridge" strands. It is assumed that there is an integer $\ell > 0$ such that the "Probe" strands have length ℓ and the others have length 2ℓ

4.3 Recovery of Output Strands

The boolean gate simulation process is followed by a recovery process for selecting only the output strands and discard the rest. This is accomplished by annealing the solution on a probe array whose probes are full antiparallel complementary strands of all the output strands that need to be picked up. The length of the double stranded DNA is 2ℓ in the case when the target strands are hybridized with the probe strands and ℓ in the case when they are hybridized with the bridge strands (see Figure 5).

To begin computation, input strands (at least 1 picomole each) are delivered to the input gate probe array along with at least 10 picomoles of each of the

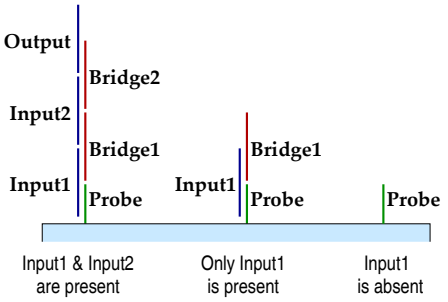


Fig. 3. The design of an AND gate. The "output" ssDNA is collected by a probe and a sequence of "hangovers." Left: both of the two "input" ssDNA are present in the input DNA solution. Middle: the first "input" ssDNA is present but the second "input" ssDNA is absent. Right: the second "input" ssDNA is absent.

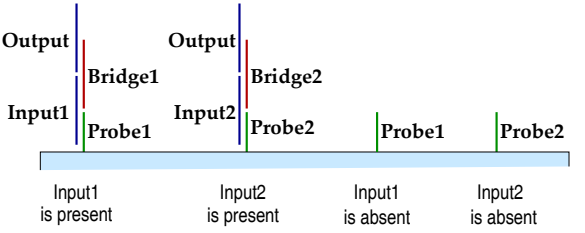


Fig. 4. The design of an OR gate. The "output" ssDNA is collected by two sequences of "hangovers." Left: collection of the "output" using the first "input" ssDNA. Middle Left: collection of the "output" using the second "input" ssDNA. Middle Right: in the case when the first "input" ssDNA is absent. Right: in the case when the second "input" ssDNA is absent.

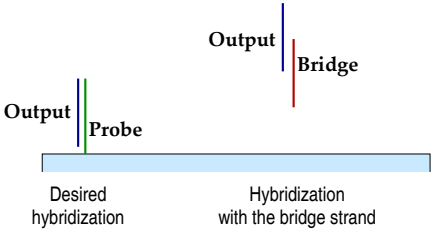


Fig. 5. Extraction of output strands. The probes are full-length antiparallel complementary strands to the output strands. The bridge strands that hybridize with the output strands may be present in the solution.

bridge 1, bridge 2, and output sequences. Hybridization can be done in the presence of cationic molecules such as dodecyl- or cetyltrimethylammonium bromide (DTAB or CTAB, respectively), which enhance DNA-DNA hybridization more than 10^4 fold, even in the presence of over 10^6 fold excess of noncomplementary DNA molecules [23]. The rate of hybridization of short oligonucleotides in the presence of DTAB or CTAB can be as high as 10^{10} $M^{-1}s^{-1}$, which is over the theoretical diffusion limit of second order hybridization rates based on naïve models. Annealing and washing can be driven by electric field on MEMS-based microelectronic circuits [7,24,12]. The excess fluid is washed away with buffer. The DNA strands are then melted by pulsed heating or electrically [7], and are pumped into the next chamber with the second set of gate array and output strands, and annealed as before in the presence of DTAB or CTAB. It is possible to repeat each cycle to amplify the signal as below. In order to reduce noise, it is essential that all unhybridized strands are removed before each succeeding cycle of hybridization. To achieve this, the efflux wash fluid is repeatedly passed over a gate/output trap to remove the unhybridized strands, and the resulting fluid is ultimately pumped on to the next probe array.

To amplify the signal, suppose that the value of ℓ is such that fully double-stranded DNA having base length 2ℓ has a much higher melting temperature than fully double-stranded DNA having base length ℓ, so that the temperature for the anneal operation can be set where hybrids between molecules having base length 2ℓ rarely break up while hybrids having base length ℓ melt with probability α, where $\frac{1}{2} < \alpha < 1$. Suppose that the five-step extraction process is repeated n times. After n rounds the proportion of the output strands that still need to be recovered is $(1-\alpha)^n$. The number of output strands is at most 4^ℓ since only their first ℓ molecules matter for the reaction preceding the extract phase. Then, if $4^\ell(1-\alpha)^n < 1$, the expect number of the output strands that are yet to be recovered is less than one after n rounds. The inequality holds when $n > 2\ell/\log\frac{1}{1-\alpha}$. Thus, the smallest n for which the desired property holds is $\left\lfloor \frac{2\ell}{\log\frac{1}{1-\alpha}} + 1 \right\rfloor$.

On the other hand, suppose that the value of ℓ is large that there is not much difference in the melting temperature between the two types of double-stranded DNA. Suppose that the temperature is set well below the melting temperatures of these double-stranded DNA. Then each output strand has a 50% chance of being recovered in a single round. If the five-cycle process is repeated n times then the probability is $1/2^n$ that the output strand is not recovered by the end of the n cycles. So, if $n \geq 4\ell$, then the expected number of output strands that are not recovered after n rounds is at most 1. Thus, in both cases, the computation is slowed down by a factor of $c\ell$ for some constant c.

4.4 Duplication

In order to resolve contention for an input signal, which is limited to two since the fan-out of the circuit is bounded by two, from one output strand, two output strands need to be created. The copies are necessarily different from the original

output strand. The mechanism in which such copies are created is essentially the same as that in which AND gates and OR gates are simulated, see Figure 6.

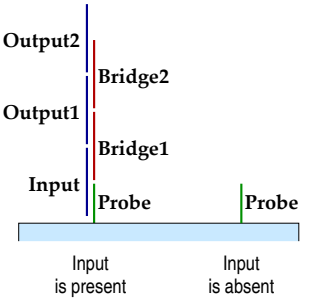


Fig. 6. Duplication of an ssDNA. The "input" ssDNA is turned into two "output" ssDNA. Left: when the "input" ssDNA is present. Right: when the "input" ssDNA is absent.

The output strands of the boolean gates and the duplication gates are recovered. A crucial point here is that some output strands are lost during the recovering process so an amplification process is needed. Using a method similar to the one shown in [19] one is able to amplify the signals: First, on one probe array create from an output two new versions. Second, on another array, create from each of the two new versions a third new version, which will be the new output signal. Also, each of the two reactions has to be followed by the recovery process.

This amplification process is the same as applying duplication followed by recovery and then OR followed by recovery. Thus, the entire computation is essentially consists of AND, OR, duplication, and recovery. To simulate a level of AND gates the sequence is

AND, recovery, duplication, recovery, duplication, recovery, OR, recovery,

and to simulate a level of OR gates the sequence is

OR, recovery, duplication, recovery, duplication, recovery, OR, recovery.

Suppose that the loss of output signals during any one of {AND, OR, duplication} combined with the subsequent recovery process is no greater than 15%. The combined loss over one of the two eight-step process is no more than $(0.85)^4$ and this is no more than 1/2. So, theoretically, the amplification process guarantees that the amplitude will never below a half of the maximum

4.5 Recycling Probe Arrays

Notice that the sequences used in these reactions need to satisfy the following condition:

– The output strands in the gate simulation phase do not hybridize with the probes.

This means that the probe sets of two consecutive reactions (i.e. AND, OR, and duplication) need to be mutually disjoint. By selecting mutually disjoint sets of probes, it is possible to simulate the entire circuit using the same set of four probe arrays. A schematic view of such design is shown in Figure 7.

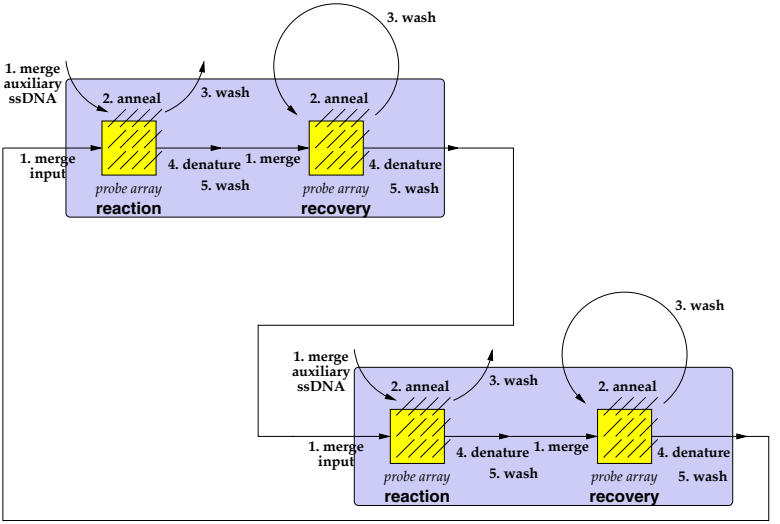


Fig. 7. A schematic view of reaction and extraction.

Acknowledgments. This work is supported in part by NSF grants EIA-0080124, DUE-998094, and EIA-0205061 and by NIH grants RO1-AG18231 and P30-AG18254.

References

1. L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
2. M. Amos, P. E. Dunne, and A. Gibbons. DNA simulation of boolean circuits. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. D. Deb, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, editors, *Proceedings of 3rd Annual Genetic Programming Conference*, pages 679–683, San Francisco, CA, 1998. Morgan Kaufmann.
3. Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430–434, 2001.

4. R. S. Braich, N. Chelapov, C. Johnson, P. W. K. Rothemund, and L. Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296:499–502, 2002.
5. A. Cukras, D. Faulhammer, R. Lipton, and L. Landweber. Chess game: a model for RNA-based computation. In *Preliminary Proceedings of 4th DIMACS Workshop on DNA Based Computers*, pages 27–37, 1998.
6. D. R. Dorris, R. Ramakrishnan, D. Trakas, F. Dudzik, R. Belval, C. Zhao, A. Nguyen, M. Domanus, and A. Mazumder. A highly reproducible, linear, and automated sample preparation method for DNA microarrays. *Genome Research*, 12(6):976–984, 2002.
7. C. F. Edman, D. E. Raymond, D. J. Wu, E. Tu, R. G. Sosnowski, W. F. Butler, M. Nerenberg, and M. J. Heller. Electric field directed nucleic acid hybridization on microchips. *Nucleic Acids Research*, 25(24):4907–4914, 1997.
8. R. Feynman. There's plenty of room at the bottom. In D. Gilbert, editor, *Miniaturization*, pages 282–296. Reingold, New York, 1961.
9. A. Gehani and J. Reif. Microflow bio-molecular computation. *Biosystems*, 52(1–3):197–216, 1999.
10. Z. Guo, R. Guigoyle, A. Thiel, R. Wang, and L. Smith. Direct fluorescence analysis of genetic polymorphisms by hybridization with oligonucleotides on glass supports. *Nucleic Acids Research*, 22:5456–5465, 1994.
11. H. J. Hoover, M. M. Klawe, and N. J. Pippenger. Bounding fan-out in logical networks. *Journal of the Association for Computing Machinery*, 31(1):13–18, 1984.
12. Y. Huang, K. L. Ewalt, M. Tirado, R. Haigis, A. Forster, D. Ackley, M. J. Heller, J. P. O'Connell, and M. Krihak. Electric manipulation of bioparticles and macromolecules on microfabricated electrodes. *Analytical Chemistry*, 73(7):1549–59, 2001.
13. J. B. Lamture, K. L. Beattie, B. E. Burke, M. D. Eggers, D. J. Ehrlich, R. Fowler, M. A. Hollis, B. B. Kosicki R. K. Reich, and S. R. Smith. Direct detection of nucleic acid hybridization on the surface of a charge coupled device. *Nucleic Acids Research*, 22(11):2121–2125, 1994.
14. K. Lindroos, U. Liljedahl, M. Raitio, and A.-C. Syvanen. Minisequencing on oligonucleotide microarrays: comparison of immobilization chemistries. *Nucleic Acids Research*, 29:e69, 2001.
15. R. Lipton. DNA solutions of hard computational problems. *Science*, 268:542–545, 1995.
16. Q. Liu, L. Wang, A. G. Frutos, R. M. Corn, and L. M. Smith. DNA computing on surfaces. *Nature*, 403:175–178, 2000. January, 13.
17. M. Ogihara. Relating the minimum model for DNA computation and Boolean circuits. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Genetic and Evolutionary Computation Conference*, pages 1817–1822. Morgan Kaufmann Publishers, San Francisco, CA, 1999.
18. M. Ogihara and A. Ray. A DNA-based self-propagating algorithm for solving bounded-fan-in Boolean circuits. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. D. Deb, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 725–730, San Francisco, CA, July 1998. Morgan Kaufman.
19. M. Ogihara and A. Ray. The minimum DNA model and its computational power. In *Unconventional Models of Computation*, pages 309–322. Springer, Singapore, 1998.
20. M. Ogihara and A. Ray. Biomolecular computing—recent theoretical and experimental advances. *SIGACT News*, 30(2):22–30, 1999.

21. M. Ogihara and A. Ray. Simulating boolean circuits on DNA computers. *Algorithmica*, 25:239–250, 1999.
22. M. Ogihara, A. Ray, and K. Smith. Biomolecular computing—a shape of computation to come. *SIGACT News*, 28(3):2–11, 1997.
23. B. W. Pontius and P. Berg. Rapid renaturation of complementary DNA strands mediated by cationic detergents: A role for high-probability binding domains in enhancing the kinetics of molecular assembly processes. *Proceedings of the National Academy of Science*, 88:8237–8241, 1991.
24. R. Radtkey, L. Feng, M. Muralhidar, M. Duhon, D. Canter, D. DiPierro, S. Fallon, E. Tu, K. McElfresh, M. Nerenberg, and R. G. Sosnowski. Rapid, high fidelity analysis of simple sequence repeats on an electronically active DNA microchip. *Nucleic Acids Research*, 28(7):E17, 2000.
25. R. Ramakrishnan, D. Dorris, A. Lublinsky, A. Nguyen, M. Domanus, A. Prokhorova, L. Gieser, E. Touma, R. Lockner, and M. Tata. Development and use of analytical tools in the dissection and optimization of microarray performance. *Nucleic Acids Research*, 30:E30, 2002.
26. J. H. Reif. Parallel biomolecular computation: methods and simulations. *Algorithmica*, 25:142–175, 1999.
27. P. Rothemund. A DNA and restriction enzyme implementation of Turing machines. In R. Lipton and E. Baum, editors, *DNA Based Computers*, pages 75–119. The American Mathematical Society DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume **27**, 1996.
28. S. Roweis, E. Winfree, R. Burgoyne, N. Chelapov, M. Goodman, P. Rothemund, and L. Adleman. A sticker based model for DNA computation. In L. Landweber and E. Baum, editors, *DNA Based Computers II*, pages 1–30. The American Mathematical Society DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume **44**, 1999.
29. J. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, and M. Hagiya. Molecular computation by DNA hairpin formation. *Science*, 288:1223–1226, 2000.
30. M. Schena, D. Shalon, R. w. Davis, and P. O. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270:467–470, 1995.
31. J. Watson, M. Gilman, J. Witkowski, and M. Zoller. *Recombinant DNA*. Scientific American Books, New York, NY, 2nd edition, 1992.
32. J. Watson, N. Hopkins, J. Roberts, J. Steiz, and A. Weiner. *Molecular Biology of the Gene*. Benjamin-Cummings, Menlo Part, CA, 4 edition, 1987.
33. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.

An Information Theoretic Approach to the Study of Genome Sequences: An Application to the Evolution of HIV

Masanori Ohya

Department of Information Sciences, Tokyo University of Science
Noda City, Chiba 278-8510, Japan ohya@is.noda.sut.ac.jp

Abstract. When a V3 sequence obtained on the n-th year after infection with human immunodeficiency virus type 1 (HIV-1) was supposed to change into a V3 sequence on the n+1-th year, the variation between the above two sequences was analyzed by means of entropic chaos degree. The entropic chaos degree measures chaotic aspects of the dynamics causing the variation of sequence. If it is large, then the dynamics produces the large complexity, in other words, the variation of sequences becomes large.
As a results, the chaos degree for the dynamics changing the V3 region showed the specific variation patterns throughout from the early stages of infection to death. That is, the variation patterns indicated that the entropic chaos degree is useful to measure the stage of disease progression after HIV-1 infection.

1 Introduction

There exist several different quantities to measure chaotic aspects of dynamical systems. A new measure, entropic chaos degree, was successfully applied to several dynamics [10]. Therefore, we analyzed the variation of human immunodeficiency virus by using this new measure.

The third variable region (V3) contained in gp 120 is the principal neutralization determinant [6] and a epitope to recognize the cytotoxic T cell [2]. Moreover it is known that the V3 region has a high substitution rate.

We calculated the entropic chaos degree of the dynamics leading from the variation of the V3 region which were observed from 19 patients infected with HIV-1 at several points in time after infection or seroconversion. Since the entropic chaos degree describes the stage of the variation, it can be considered that the state of the disease progression is characterized by this degree.

2 Materials and Methods

2.1 Patient Characteristics

The data of patients selected for this study were approximately taken once each year after HIV-infection or seroconversion (maximum; follow-up of 12 years,

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 50–57, 2002.

minimum; follow-up of 2 years). We used the sequences of the V3 region for virus clones in plasma, serum and peripheral blood mononuclear cells stored in the international DNA databases (DDBJ/EMBL/Genbank).

From nineteen HIV-infected patients, six patients (A, B, C, D, E and F) had progressed to AIDS and died of AIDS-related complications during the period of follow-up. Three patients (G, H and I) were diagnosed as having AIDS and were still alive during the period of follow-up. The remaining patients were asymptomatic throughout the period of follow-up (Table 1).

Table 1. Patient Characteristic

Patient	Duration of asymptomatic	Tissue	Duration of AIDS
Group 1			
A (tk-2)	about 6 years	PBMCs	about 3 years
B (tk-29)	about 7 years	PBMCs	about 6 years
C (ACH0208)	54 months	PBMCs	39 months
D (ACH6052)		PBMCs	40 months
E (p74)	about 4 years	PBMCs	44 months
F (p87)		PBMCs	$>$ 3 years
Group 2			
G (p495)	55months	serum	
H (p39)	37months	PBMCs	
I (p82)	7 years	plasma	
Group 3			
J (p1)	59 months	serum	
K (s5)	3 years	PBMCs	
L (s6)	4 years	PBMCs	
M (s7)	2 years	PBMCs	
N (s8)	3 years	PBMCs	
O (s9)	4 years	PBMCs	
P (s14)	4 years	PBMCs	
Q (q23)	about 2 years	cervical secretion	
R (tk-3)	about 12 years	PBMCs	
S (tk-22)	about 12 years	PBMCs	

( ): designation in original paper PBMCs: peripheral blood mononuclear cells

Group 1: Patients died of AIDS-related complications, Group 2: Patients progressed to AIDS, Group 3: Patients maintained $CD4^+$T cell numbers above 200 for follow-up period

Duration of asymptomatic in group 3 shows period untill loss of follow up

We will explain the information about the patients in detail. Patient A and B were infected with HIV-1 around 1983 [4]. Patient A had progressed to AIDS by about 6 years after infection and died in 1992. Patient B had progressed to AIDS by about 7 years after infection and died in 1996. Patient C who seroconverted in 12/1985 had progressed to AIDS in 6/1990 (4.5 years after seroconversion) and died in 9/1993 [13]. Patient D, whose seroconverion day is not known, died 40 months after AIDS diagnosis [13]. Patient E who seroconverted in 1984 had

progressed to AIDS in 4/1988 and died in 12/1991 [8,1]. Patient F seroconverted in 1984, whose the time of AIDS diagnosis is not known, died in 9/1990 [8].

Patient G was diagnosed as having AIDS in 1989; 55 months after primary infection [15]. Patient H who seroconverted in 10/1987 progressed to AIDS in 11/1990 [14]. The $CD4^+$T cell number of patient I decreased below 200 at 7 th year [3].

Patient J remained healthy for 59 months after infection [15]. Six patients' (K, L, M, N, O and P) $CD4^+$T cell numbers have not been a level of fewer than 200 for 2-4 years from seroconversion until loss of follow-up [9]. Patient Q was not showing any symptoms of disease referable to HIV-1 for 2 years from seroconversion until loss of follow-up [11]. Patient R and S were infected around 1983, who maintained stable $CD4^+$T cell number until loss of follow-up (1995) [4].

2.2 Entropic Chaos Degree

The entropic chaos degree for the sequence $X^{(n)}$ obtained on the n-th year after HIV-1 infection and that $X^{(n+1)}$ obtained on the $n+1$-th year is given as follows when the information of $X^{(n)}$ was transmitted to that of $X^{(n+1)}$. First, we consider two aligned amino acid sequences $X^{(n)}$ and $X^{(n+1)}$, which are composed of 20 kinds of amino acids and the gap. The complete event system of $X^{(n)}$ is determined by the occurrence probability p_i of each amino acid a_i and p_0 of the gap [12];

$$\left(X^{(n)}, p^{(n)}\right) = \begin{pmatrix} * & a_1 & \dots & a_{20} \\ p_0^{(n)} & p_1^{(n)} & \dots & p_{20}^{(n)} \end{pmatrix}$$

In the same way, the complete event system of the $n+1$-the year is;

$$\left(X^{(n+1)}, p^{(n+1)}\right) = \begin{pmatrix} * & a_1 & \dots & a_{20} \\ p_0^{(n+1)} & p_1^{(n+1)} & \dots & p_{20}^{(n+1)} \end{pmatrix}$$

The compound event system of $X^{(n)}$ and $X^{(n+1)}$ is denoted by

$$\left(X^{(n,n+1)}, p^{(n,n+1)}\right) = \begin{pmatrix} * & a_1 & \dots & a_{20} \\ r_{00}^{(n,n+1)} & r_{01}^{(n,n+1)} & \dots & r_{2020}^{(n,n+1)} \end{pmatrix}$$

$$\left(r^{(n,n+1)} = \left\{r_{ij}^{(n,n+1)}\right\}_{i=0,j=0}^{20,20}\right),$$

where r_{ij} represents the joint probability of the event i of the n-th year sequence and the event j of the $n+1$-th year sequence.

We suppose the dynamics describing the change of sequence $X^{(n)}$ to $X^{(n+1)}$ is given by a certain mapping Λ called a channel for $p^{(n)}$ to $p^{(n+1)}$. It's very difficult

to know the exact form of this mapping in the course of the variation of HIV-1. However, our entropic chaos degree can be used to measure the complexity without knowing the exact form [10,5].

The entropic chaos degree for the sequence obtained on the $n-$th year after HIV-1 infection and that on the $n+1-$th year is given the following formula;

$$C_{\Lambda}^{(n)}(X) = \sum r_{ij}^{(n,n+1)} \log \frac{p_i^{(n)}}{r_{ij}^{(n,n+1)}}$$

Practically, we used the V3 sequences for some virus clones at each point in time. Therefore, we take the average of the entropic chaos degree, each of which is given as above.

3 Results

Fig. 1 shows the change of the entropic chaos degree of V3 sequences obtained at several points in time during the clinical course of each patient. The value of the entropic chaos degree was low at the early stages of infection. However, from the time of primary infection until the time of AIDS diagnosis, the entropic chaos degree increased gradually and reached a maximum just after the time of AIDS diagnosis. Then, the chaos degree continued to decrease up to death for the duration of AIDS. Our study patients were classified into three groups 1) patients who died of AIDS-related complications, 2) patients who progressed to AIDS, 3) asymptomatic patients who did not progress to AIDS during the period of follow-up. The values of the entropic chaos degree were higher in AIDS patients than in asymptomatic patients, as was shown in Fig. 2. For all patients of the group 1, the value of the entropic chaos degree had temporarily larger than 0.2. On the other hand, almost all patients of the group 3 showed the value below 0.15 over the period of study (5 of 10 patients; chaos degree < 0.1 , 4 of 10; < 0.15).

4 Discussion

We analyzed the variation of the V3 sequence by using the entropic chaos degree. This degree measures chaotic aspects of the dynamics causing the variation of sequence.

Whether a patient progressed to AIDS or not is so far decided on depletion of $CD4^+$ T cell. Although the majority of patients infected with HIV-1 is considered to progress to AIDS, a small number of patients remain healthy and maintain $CD4^+$ T cell numbers or above 200 for more than a decade after infection. In addition to $CD4^+$ T cell, it is known that HIV-1 RNA level in plasma is closely related to the stage of the disease. However, it is hard to predict not only the time of the progress to AIDS and death relative to AIDS, but also rapid progressors or slow progressors.

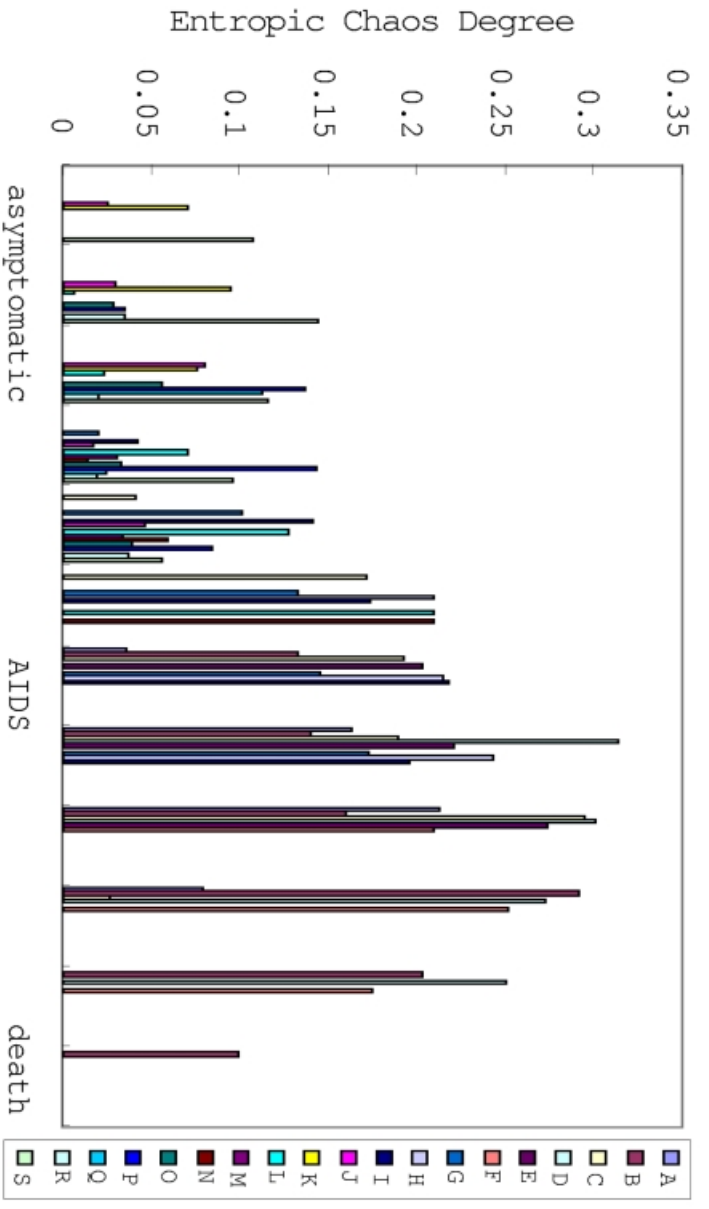


Fig. 1. The entropic chaos degree for V3 sequences obtained at several points in time of each patient are plotted altogether

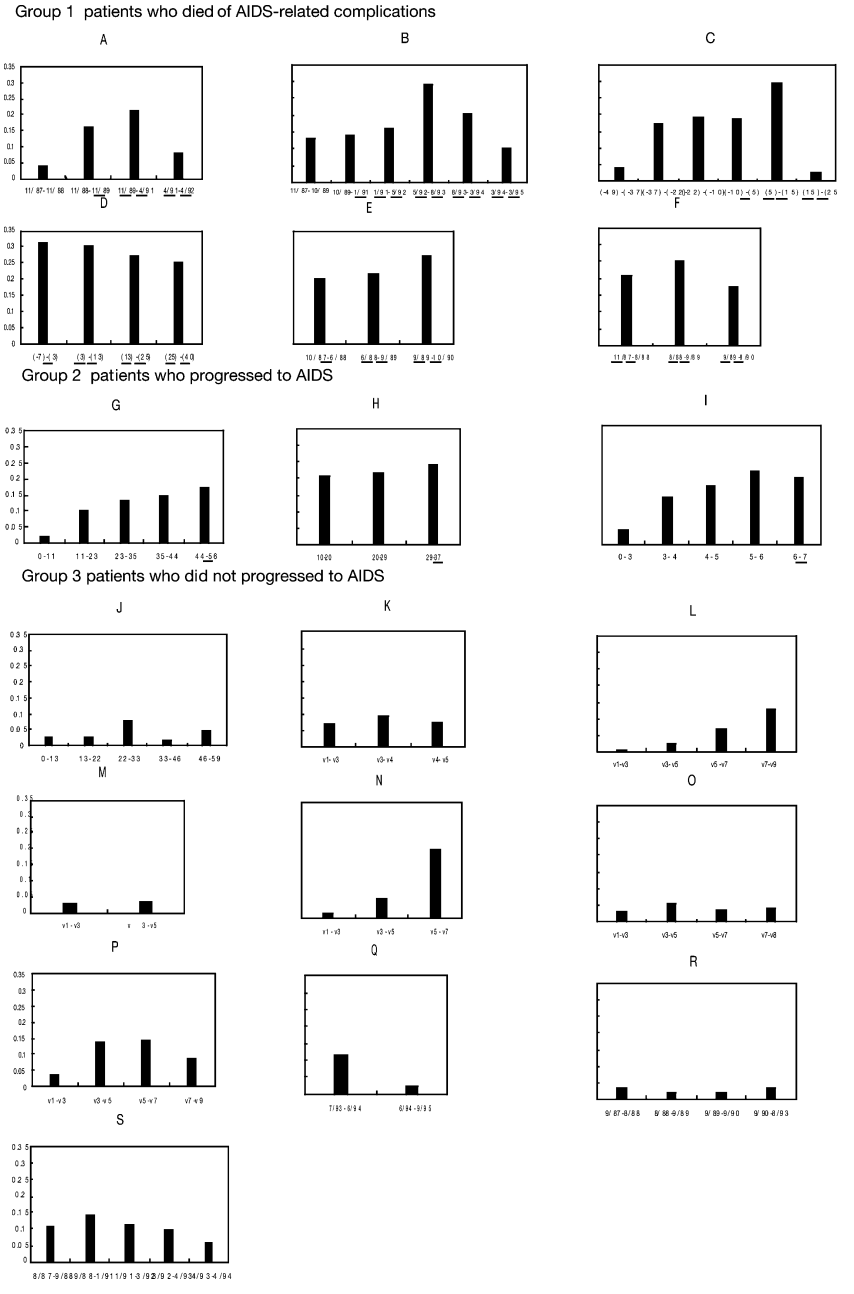


Fig. 2. Entropic chaos degree increased in group 1 and group 2 as AIDS developed. However, it remained low in group 3, except patient N. (The time to be diagnosed as AIDS is indicated by an underline.)

From our results, patients who developed AIDS took the entropic chaos degree high value compare with patients who remain asymptomatic. All patients who died of AIDS-related complications surely took the value 0.2 or more during the period of study and died within 3-4 years thereafter. In contrast, patients who continued taking the value less than 0.1 or immediately going back to the value less than 0.1, even if the value transiently becomes larger than 0.1, were asymptomatic without AIDS. Our results indicate that the chaos degree increases with leading to AIDS after infection and attains the value larger than 0.2. Therefore, when the chaos degree has decreased after this increase without attaing high value like patient P and S or it has maintained stable and low value (< 0.1) like patient J, K, M, O and R, those patients can be classified as long term non progressors. Patient N who appears to have been slow progressor was exceedingly close to AIDS development, because the value of the entropic chaos degree between 2 and 3 years (visit 5 and visit 7) after seroconversion indicated larger value than 0.2 and drew a characteristic increase pattern such as AIDS patients.

Many researcher are exploring every markers to recognize AIDS development and disease progression, and there have been some diagnosis markers used to estimate prognosis in patients with HIV-1 infection [7]. We examined whether the entropic chaos degree can be a candidate of these markers. Although the data used here are not sufficiently many so far, we in this paper propose that the entropic chaos degree can be one of the markers to estimate disease progression in HIV-1.

References

1. Cleland A, Watson HG, Robertson P, Ludlam CA, Leigh Brown AJ (1996) Evolution of Zidovudine Resistance-Associated Genotypes in Human Immunodeficiency Virus Type 1- infected Patients, Journal of Acqired Immune Deficiency Syndromes and Human Retrovirology, 12(1): 6-18
2. Takahashi H, Nakagawa Y, Pendleton C.D., Houghten R.A, Yokomuro K, Germain R.N Berzofsky J.A (1992) Induction of Brosadly Cross-Reactive Cytotoxic T-Cells Recognizing an HIV-1 Envelope Determinant. Science, Vol.255: 333-336
3. Holmes E C, Zhang LQ, Simmonds P, Ludlam CA, Leigh-Brown AJ (1992) Convergent and divergent sequence evolution in the surface envelope glycoprotein of Human immunodeficiency virus type 1 within a single infected patient. Proc. Natl. Acad. Sci. U.S.A. 89: 4835-4839
4. Ida S, Gatanaga H, Shioda T, Nagai Y, Kobayashi N, Shimada K, Kimura S, Iwamoto A, Oka S (1997) HIV type 1 V3 variation dynamics in vivo: long-term persistence of non-syncytium-inducing genotypes and transient presence of syncytium-inducing genotypes during the course of progressive AIDS. AIDS Res. Hum. Retroviruses 13 (18): 1597-1609
5. Inoue K, Ohya M and Sato K: Application of chaos degree to some dynamical systems, to appear in Chaos, Solitons & Fractals
6. Javaherian K, Langlois A.J, McDanal C, Ross K.L, Eckler L.I., Jellis C.L, Proty A.T, Rusche J.R, Bolognesi D.P, Putwey S.D., Matthews T.J. (1989) Principal Newtralizing Domain of the Human Immunodeficiency Virus Type 1 Envelope Protein, Proc. Natl. Acad. Sci. USA, Vol.86: 6768-6772

7. John W. Mellors, Lawrence A. Kingsley, Charles r. Rinaldo, John A. Todd, Brad S. Hoo, Robert P. Kokka, Phalguni Gupta (1995) Quantitation of HIV-1 RNA in Plasma Predict Outcome after Seroconversion, Ann. Intern. Med.,122: 573-579
8. Leigh Brown AJ, Cleland A (1996) Independent evolution of the env and pol genes of HIV-1 during zidovudine therapy. AIDS 10(10): 1067-1073
9. Markham,R.B., Wang,W.C., Weisstein, A.E., Wang, Z., Munoz, A., Templeton, A., Margolick, J., Vlahov, D., Quinn, T., Farzadegan, H., Yu, X.F. (1998) Patterns of HIV-1 evolution in individuals with differing rates of CD4 T cell decline. Proc. Natl. Acad. Sci. U.S.A. 95(21): 12568-12573
10. Ohya M (1998) Complexities and their applications to characterization of chaos, International Jornal of Theoretical Physics, 37, No.1: 495-505
11. Poss M, Rodrigo A.G., Gosink J.J., Learn G.H., de Vange Panteleeff D., Martin H.L. Jr., Bwayo J., Kreiss J.K., Overbaugh J (1998) Evolution of envelope sequences from the genital tract and peripheral blood of women infected with clade A human immunodeficiency virus type 1. J. Virol. 72(10): 8240-8251
12. Sato K., Miyazaki S., Ohya M. (1998) Analysis of HIV by entropy evolution rate. Amino Acids 14: 343-352
13. van't Wout A.B, Ran L.J., Kuiken C.L., Kootstra N.A., Pals S.T., Schuitemaker, H. (1998) Analysis of the temporal relationship between human immunodeficiency virus type 1 quasispecies in sequential blood samples and various organs obtained at autopsy. J. Virol. 72(1): 488-496
14. van't Wout, A.B., Blaak, H., Ran, L.J., Brouwer, M., Kuiken, C.L., Schuitemkaer, H. (1998) Evolution of syncytium-inducing and non-syncytium-inducing biological virus clones in relation to replication kinetics during the course of HIV-1 infection. J. Virol. 72(6): 5099-5107
15. Wolfs T.W., Zwart G., Bakker M., Valk M., Kuiken C., Goudsmit J. (1991) Naturally occurring mutations within HIV-1 V3 genomic RNA lead to antigenic variation dependent on a single amino scid substitution. Virology 185: 195-205

Halting of Quantum Turing Machines

Masanao Ozawa

Graduate School of Information Sciences,
Tôhoku University, Aoba-ku, Sendai, 980-8579, Japan
ozawa@math.is.tohoku.ac.jp

Abstract. The notion of a quantum Turing machine (QTM) is well established as a mathematical model. However, the difficulty on halting procedures has prevented us from formulating the notion of computing on an arbitrary quantum Turing machine. Here, an argument is outlined to show that any QTM can be efficiently simulated by a QTM with well-behaved halting flag. Thus, we can affirmatively solve the halting problem for QTMs.

1 Introduction

In the late 1980s, Deutsch introduced quantum Turing machines [1] and quantum circuits [2] to model quantum computers exploiting "quantum parallelism". In 1994, Deutsch's idea of quantum parallelism was realized strikingly by Shor [3], who found efficient quantum algorithms for the factoring problem and the discrete logarithm problem, for which no efficient algorithms have been found for classical computing machines.

Unlike the classical computer, the quantum Turing machine (QTM) cannot be monitored throughout the computation because of the inevitable disturbance caused by measurement. Thus, the machine needs a specific halt scheme to signal when the computation has been completed. For this purpose, Deutsch [1] proposed the use of the halt qubit. However, Myers [4] argued in 1997 that the halting of a QTM spoils the computation when the state is the superposition of halt qubit and non-halt qubit. In 1998, the present author [5,6] reformulated Deutsch's halting condition precisely and proved that the measurement of the halt qubit does not spoil the computation as long as that condition is satisfied. Linden and Popescu [7] pointed out that under a certain additional condition, the halting condition contradicts unitarity and claimed that the halting of the universal QTM is impossible.

This paper proposes a positive solution for the problem on the halting procedure of quantum Turing machines. For this purpose, a certain difficulty in defining the output probability for arbitrary QTMs is pointed out and a new formulation of the output probability based on the amplitude over the computational paths is introduced. It is shown that for any QTM there is a QTM with a well-behaved halting flag which efficiently simulates the output probability.

Quantum complexity theory was instituted by Bernstein and Vazirani [8], but in order to avoid the halting problem, they restricted the class of QTMs to

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 58–65, 2002.

the very special class of QTMs that synchronize the completion of computation. The above result enables us to generalize quantum complexity theory to arbitrary QTMs without affecting the basic complexity classes.

2 Computing Processes of Quantum Turing Machines

For the basic formulation and properties of QTMs, we refer the reader to Ref. [9]. Let M be a QTM with configuration space $\mathcal{C}(Q, \Sigma) = Q \times \Sigma^{\#} \times \mathbf{Z}$. We assume that we have a device to prepare the QTM M in the state $|q, T, \xi\rangle$ for any configuration $C = (q, T, \xi)$ and that we have a measuring device to measure sufficiently many but finite numbers of $\hat{T}(m)$s simultaneously.

Let Γ be a finite set of symbols and Γ^* the set of finite strings from Γ. We assume $B \notin \Gamma$. In this paper, we shall consider computations which are probabilistic transformations on Γ^*, or precisely functions from Γ^* to the set of probability distributions on Γ^*. The set Γ is called the *alphabet* of the computation. A finite string from the set Γ is called a *Γ-string*. The length of a Γ-string x is denoted by $|x|$. When $|x| = 0$, x is called the empty string. We shall identify any Γ-string $x = (x_0, \ldots, x_{|x|-1})$ with a function x from $\{0, \ldots, |x| - 1\}$ to Γ such that $x(m) = x_m$ for all m with $0 \leq m \leq |x| - 1$.

The computation by a QTM consists of *encoding*, *preparation*, *time evolution*, *measurement*, and *decoding*. The encoding transforms the *input Γ-string* to the *input tape string*. The preparation prepares the *initial state* of the QTM with the input tape string, and the time evolution transforms the initial state to the *final state*. The measurement of the tape string in the final state gives a probability distribution of the *output tape string*. The decoding transforms the output tape string to the output Γ-string and hence transforms the probability distribution of the output tape string to the probability distribution of the output Γ-string. Therefore, the initial Γ-string is transformed to the *output probability distribution* of the Γ-string.

The encoding e of the QTM M is a polynomial time computable function from Γ^* to $\Sigma^{\#}$. Thus, the encoding e transforms any Γ-string x to a tape configuration denoted by $e(x)$; if $T = e(x)$ we shall write $T \sim x$ and T is said to *represent* the Γ-string x. Inversely, the *decoding* d of M is a polynomial time computable function from $\Sigma^{\#}$ to Γ^* satisfying $d(e(x)) = x$ for all $x \in \Gamma^*$.

The encoding e is called the *standard encoding* if $\Sigma = \Gamma \cup \{B\}$ and if e is defined by

$$e(x)(m) = \begin{cases} x(n) & \text{if } m = m_n \in S \\ & \text{and } 0 \leq n < |x|, \\ B & \text{otherwise,} \end{cases} \tag{1}$$

for any $x \in \Gamma^*$. The *standard decoding* is then naturally defined by

$$|d(T)| = \min\{m_n \in S |\ T(m_n) = B\}, \tag{2}$$

$$d(T)(n) = T(m_n) \tag{3}$$

for $0 \leq n < |d(T)|$, where $T \in \Sigma^{\#}$.

The computation begins at $t = 0$. At this time M is prepared in an *initial state* $|C_0\rangle$ such that

$$|C_0\rangle = |q_0, e(x), 0\rangle, \tag{4}$$

where $x \in \Gamma^*$. Generally, any configuration C_0 of the form $C_0 = (q_0, e(x), 0)$ is called an *initial configuration* with *input* $x \in \Gamma^*$, where e is the given encoding. In this case, $|x|$ is called the *input length*.

Since the number of all the possible tape strings is countable, we assume them to be indexed as $\{T_1, T_2, \ldots\}$. Thus, the observable $\hat{T}$ on $\mathcal{H}(Q, \Sigma)$ describing the tape string can be represented by

$$\hat{T} = \sum_{j=1}^{\infty} \lambda_j \, I_Q \otimes |T_j\rangle\langle T_j| \otimes I_{\mathbf{Z}}$$

where $\{\lambda_1, \lambda_2, \ldots\}$ is a countable set of positive numbers in one-to-one correspondence with $\{T_1, T_2, \ldots\}$ by a polynomial time function and where I_Q is the identity on $\mathcal{H}(Q)$ and $I_{\mathbf{Z}}$ is the identity on $\mathcal{H}(\mathbf{Z})$. In order to save notations, we will write $E(\hat{T} = T_j)$ instead of $E(\hat{T} = \lambda_j)$ for the eigenprojection of $\hat{T}$ corresponding to the eigenvalue λ.

3 Halting Protocol

The result of a computation is obtained by measuring the tape string after the computation has been completed. Unlike the classical case, the machine configuration cannot be monitored throughout the computation because of the inevitable disturbance caused by measurement. Thus, the machine needs a specific halt scheme to signal actively when the computation has been completed.

Deutsch [1] introduced an additional single qubit, called the halt qubit, together with an observable $\hat{n}_0$, called the halt flag, with the eigenstates $|0\rangle$ and $|1\rangle$, so that the processor configuration q is represented by the state vector $|q\rangle|1\rangle$ if q is the final state in the classical picture or by $|q\rangle|0\rangle$ otherwise. The halt qubit is initialized to $|0\rangle$ before starting the computation, and every *valid* quantum algorithm sets the halt qubit to $|1\rangle$ when the computation has been completed but does not interact with the halt qubit otherwise. Deutsch claimed that *the observable $\hat{n}_0$ can then be periodically observed from the outside without affecting the operation of the machine.*

Myers [4] argued that the state entangles the non-halt qubits with the halt qubits so that the measurement of the halt flag changes the state and concluded that the halt scheme spoils the computation.

In [5], Deutsch's halt scheme is reformulated precisely and it is shown that, even though the measurement of the halt flag changes the state of the quantum Turing machine, it does not change the probability distribution of the outcome of the computation so that every valid quantum algorithm does not spoil the computation. It is also shown in [5] that the halt scheme is equivalent to the quantum nondemolition monitoring of the halt flag during the unitary evolution of computation.

In what follows, we shall give a new formulation of the halt scheme which is more general than the previous one and in which the additional halt qubit is not augmented.

The *halt flag* $\hat{n}_0$ is defined to be the observable corresponding to the projection on the final configuration of the processor, i.e.

$$\hat{n}_0 = |q_f\rangle\langle q_f| \otimes I_{\Sigma^\#} \otimes I_{\mathbf{Z}}, \tag{5}$$

where $I_{\Sigma^\#}$ is the identity on $\mathcal{H}(\Sigma^\#)$. We assume that we have a measuring apparatus to measure $\hat{n}_0$ precisely after each step instantaneously in the manner satisfying the projection postulate. Thus the $\hat{n}_0$-measurement gives surely the outcome 1 if and only if the processor is in $|q_f\rangle$. We shall denote by $E(\hat{A} = a)$ the spectral projection onto the eigenspace of an observable $\hat{A}$ corresponding to the eigenvalue a. The product $E(\hat{A} = a)E(\hat{B} = b)$, if commutable, will be denoted by $E(\hat{A} = a, \hat{B} = b)$.

The precise formulation of the halting protocol for a QTM $M(Q, \Sigma, \delta)$ with encoding e and decoding d is given as follows.

(I) The halt flag $\hat{n}_0$ is measured instantaneously after every step. This measurement is a precise measurement of the observable $\hat{n}_0$ satisfying the projection postulate. (Note that the above measurement is different from the procedure that one measures $\hat{q}$ and checks if the outcome is q_f because this does not satisfy the projection postulate.)

(II) Once the halt flag is set to $\hat{n}_0 = 1$, the QTM no more changes the halt flag nor the result of computation. Thus, we require the following *halting condition*

$$\begin{aligned} &UE(\hat{n}_0 = 1, \hat{T} = T_j)U^t|C\rangle \\ &= E(\hat{n}_0 = 1, \hat{T}(S) = T_j)UE(\hat{n}_0 = 1, \hat{T}(S) = T_j)U^t|C\rangle \end{aligned} \tag{6}$$

for any initial configuration $C = |q_0, e(x), 0\rangle$ with input $x \in \Gamma^*$, time $t \geq 0$, and tape string T_j.

(III) When the measurement of the halt flag $\hat{n}_0$ gives the outcome 1, the observable $\hat{T}$ of the tape string is measured. The outcome T_j of this measurement is defined to be the *output tape string*. The output $y \in \Gamma^*$ of computation is then defined by $y = d(T_j)$.

Now we shall show that the halting protocol does not affect the result of the computation. For that purpose, it suffices to prove that the probability distribution of the output is not affected by monitoring of the halt flag.

Let $P_N(y|x, n)$ be the probability distribution of the output $y \in \Gamma^*$ at the time n given the input $x \in \Gamma^*$ under the condition that there are no measurement before the time n. On the other hand, let $P_M(y|x, n)$ be the corresponding probability distribution under the condition that the halt flag is measured after every step instantaneously by an apparatus satisfying the projection postulate. We call $P_N(y|x, n)$ the *non-monitored output probability distribution* and $P_M(y|x, n)$ the *monitored output probability distribution*.

Then, we can extend the result of [5] to the following theorem.

Theorem 1 *If the time evolution U of a QTM $M(Q, \Sigma, \delta)$ satisfies the halting condition (6), then the non-monitored output probability coincides with the monitored output probability.*

From the above theorem, it is concluded that the the probability of finding the output y up to N steps by the computation obeying the halt protocol is equal to the probability of finding the output y by the single measurement of $\hat{T}(S)$ after N steps. It follows that in any computation obeying the halting protocol the measurement of the halt flag changes the quantum state of the QTM according to the projection postulate but does not affect the result of the computation.

Recently, Linden and Popescu [7] claimed that the halt scheme given in [5] is not consistent with unitarity of the evolution operator. However, their argument applies only to the special case in which the whole tape is required not to change after the halt. As suggested in a footnote 11 of [5], the conclusion in [5] can be obtained from the weaker condition for the general case where the tape is allowed to change except for the date slot. Linden and Popescu [7] disregarded this case and hence their conclusion is not generally true. In this paper, the halting protocol with such a general formulation is treated explicitly and it is shown that even in this case the computation is not affected by the measurement of the halt flag.

4 Simulation of Arbitrary QTMs by QTMs with Well-Behaved Halt Flags

Deutsch [1] called the quantum algorithm valid if it satisfies his halting condition. In the preceding section, we have reformulated his halting condition in a precise language and shown that for the "valid" QTM we can know when the computation is completed without affecting the computation. Thus, the criticism due to Myers is cleared in principle that the measurement of the halt flag in the entangled state of the halt qubit and nonhalt qubit spoils the computation. However, Linden and Popescu [7] pointed out the difficulty in implementing the halt protocol to a general quantum Turing machine. They showed that in a very restrictive case the halting condition is inconsistency of the unitarity of the evolution operator. However, the difficulty arises from a certain inessential aspect of the reversibility of quantum computing. From the point of view of the designer of a quantum algorithm, the time evolution of the QTM after the completion of the computation is of no need. However, if we define the "valid" output probability to be the non-monitoring output probability at the time enough later than the completion of computation in all the computational path, this probability is easily destroyed by the coherence between the completed computational paths and the incompleted computational paths in the time evolution. Thus, the true problem of halting a QTM is to recover not the non-monitoring output probability but a certain intrinsic output probability by the monitoring output probability of a valid QTM. In this section, we shall introduce an output probability based on the computational path amplitude as an "intrinsic" output probability and

show that this output probability of any QTM can be reproduced as monitoring output probability of a well-halting QTM.

A sequence $\langle C_0, \ldots, C_L \rangle$ of configurations is called a *computational path* if C_0 is an initial configuration, C_L is an final configuration, and none of $C_1, \ldots, C_{L-1}$ is initial nor final, i.e., if $C_n = (\tau_n, T_{j(n)}, \xi_n)$ for $n = 0, \ldots, L$ then $\tau_0 = q_0$, $T_{j(n)} = e(x)$ for some $x \in \Gamma^*$, $\xi_0 = 0$, $\tau_L = q_f$, $\tau_n \neq q_0$, and $\tau_n \neq q_f$. Given a computational path $\langle C_0, \ldots, C_L \rangle$, its *amplitude* $\delta(C_0, \ldots, C_L)$ is defined by

$$\begin{aligned}&\delta(C_0, C_1, \ldots, C_L)\\ &= \langle C_L|U|C_{L-1}\rangle\langle C_{L-1}|U|C_{L-2}\rangle \cdots \times \langle C_1|U|C_0\rangle.\end{aligned}$$

Let $\mathcal{P}(T_j)$ be the set of computational paths $\langle C_0, \ldots, C_L \rangle$ such that

$$E(\hat{T} = T_j)|C_L\rangle = |C_L\rangle.$$

The *path probability* of obtaining the output tape string T_j at the time $t = N$ given the initial configuration C_0 is defined by

$$\begin{aligned}&\Pr\{\text{path output}(t = N) = T_j|C_0\}\\ &= \sum_{\langle C_0,\ldots,C_N\rangle \in \mathcal{P}(T_j)} \left| \sum_{C_1,\ldots,C_{N-1} \in \mathcal{C}(Q,\Sigma)} \delta(C_0, C_1, \ldots, C_N) \right|^2.\end{aligned}$$

The computational path probability of obtaining the output tape string T_j up to the time $t = N$ given the initial configuration C_0 is naturally given by

$$\begin{aligned}&\Pr\{\text{path output}(t \leq N) = T_j|C_0\}\\ &= \sum_{K=1}^{N} \Pr\{\text{path output}(t = K) = T_j\}.\end{aligned}$$

Then, the *path probability distribution* $P_P(y|x, n)$ of the output $y \in \Gamma^*$ up to the time N given the input $x \in \Gamma^*$ is naturally defined by

$$P_P(y|x, n) = \sum_{T_j : d(T_j) = y} \Pr\{\text{path output}(t \leq N) = T_j|C_0\}$$

with $C_0 = (q_O, e(x), 0)$. Let $\mathcal{C}(y|x, n)$ be the set of all computational path $\langle C_0, \ldots, C_L \rangle$ satisfying the following conditions.

(i) $L \leq N$.

(ii) $C_0 = (q_0, e(x), 0)$.

(iii) $C_N = (q_f, T_j, \xi)$ with $d(T_j) = y$ and $\xi \in \mathbf{Z}$.

Then, we have

$$P_P(y|x, n) = \sum_{\langle C_N,\ldots,C_0\rangle \in \mathcal{C}(y|x,n)} |\delta(C_N, \ldots, C_0)|^2.$$

By definition, the computational path probability of a given QTM is not an observable probability. However, from the algorithmic point of view, this is

the most reasonable probability to be designed, since this probability is not disturbed by any measurement or any superfluous coherence due to the time evolution after completing the computation. The following theorem shows that for any QTM we can equip the halting procedure that makes the computational path probability observable by the measurement of tape cells.

Theorem 2 *For any 2-way QTM $M = \langle Q, \Sigma, \delta \rangle$ with standard encoding and decoding, there is a well-monitored QTM $M' = \langle Q', \Sigma', \delta' \rangle$ such that M' reproduces the computational path probability of M as the monitored probability of M'.*

Bernstein and Vazirani [8] required the synchronization of computational paths so that every computational path reaches a final configuration simultaneously. Since the time evolution may spoil the output of computation, the output cannot be observed unless the exact halting time is previously known. Thus, the halt scheme should be required for the observability of the output even when the paths are synchronized.

The formal definition of QTMs satisfying the above synchronization condition is given as follows. A QTM $M = (Q, \Sigma, \delta)$ is said to be *stationary* if given an initial configuration C, there exists some $t \in \mathbf{N}$ satisfying

$$\|E(\hat{\xi} = 0, \hat{q} = q_f)U^t|C\rangle\|^2 = 1$$

and for all $s < t$ we have

$$\|E(\hat{q} = q_f)U^s|C\rangle\|^2 = 0.$$

From Theorem 2, we can show that any stationary QTM can be simulated by a QTM obeying the halting protocol. Thus, quantum complexity theory developed by Bernstein and Vazirani [8] can be applied not only to stationary QTMs but also to any QTMs with their path probability distributions.

Acknowledgments. This work was supported by the programme "R&D on Quantum Communication Technology" of the MPHPT of Japan, by the CREST project of the JST, and by the Grant-in-Aid for Scientific Research of the JSPS.

References

1. D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond.*, A **400**:97–117, (1985).
2. D. Deutsch. Quantum computational networks. *Proc. R. Soc. Lond.*, A **425**:73–90, (1989).
3. P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In G. Goldwasser, editor, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Los Alamitos, CA, 1994. IEEE Computer Society Press.

4. J. M. Myers. Can a universal quantum computer be fully quantum? *Phys. Rev. Lett.*, **78**:1823–1824, (1997).
5. M. Ozawa. Quantum nondemolition monitoring of universal quantum computers. *Phys. Rev. Lett.*, **80**:631–634, (1998).
6. M. Ozawa. Quantum Turing machines: Local transition, preparation, measurement, and halting. In P. Kumar, G. M. D'Ariano, and O. Hirota, editors, *Quantum Communication, Computing, and Measurement 2*, pages 233–241, New York, (2000). Kluwer/Plenum.
7. N. Linden and S. Popescu. The halting problem for quantum computers. Eprint: quant-ph/9806054, 16 June 1998.
8. E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, **26**:1411–1473, (1997).
9. H. Nishimura and M. Ozawa. Computational complexity of uniform quantum circuit families and quantum Turing machines. *Theoret. Comput. Sci.*, **276**:147–181, (2002).

Filtrons of Automata

Paweł Siwak

Institute of Control and Information Engineering
Poznan University of Technology
Pl. Skłodowskiej-Curie 5, 60-965 Poznań, Poland
siwak@sk-kari.put.poznan.pl

Periodic moving coherent structures (particles) are well known in parallel string processing (SP) performed by cellular automata (CAs). In 1986 the discrete soliton-like objects were shown in a filter CA model that performs serial SP. Then, some other systems that support discrete solitons were proposed in nonlinear physics. Now there are iterated arrays, filter CAs, soliton CAs, higher order CAs, sequentially updated CAs, integrable CAs, IIR digital filters, filter transducers, ultradiscrete soliton equations (KdV, KP, L-V), and fast rules. Also, box-ball systems, crystal systems and affine Lie algebras were introduced recently. We show a unified approach to all these processing mechanisms. They are based on iterated automata maps (IAMs). Automaton equivalents to various systems differ on their organization of memory. We show the automata that use various finite and/or infinite memories: shift registers, counters, stacks, FIFO, lists, and pipelines of counters. In our approach the IAMs mimic transmitting media, while filtrons describe propagating coherent disturbances. We mention also various phenomena of interacting filtrons.

1 Introduction

Iterated maps have been analyzed mainly as nonlinear dynamical systems. In CS they are also considered within a framework of 1D cellular automata (CAs). This model performs iterated *parallel* string processing (SP), and in many cases supports moving coherent structures, called particles or signals [2, 8]. Iterated automata maps (IAMs) with *serial* SP are known as OLIA model (see O. Ibarra in [8]). They perform recursive digital filtering (IIR filtering), and can be treated as 1D (automaton) transmitting medium. But the emergent propagating objects of IAMs, which we call filtrons, and their connections with nonlinear physics are almost unknown.

The significance of IAMs and their filtrons, closely associated with questions about the future of computational means, is related to the following facts:

- Colliding streams of discrete solitons (or filtrons) can be applied in performing computations in solitonic processors [2, 22, 23, 42, 43, 44]. There is the rapid progress in technology, where the possibilities of shaping the optical media and guiding light by light are being dramatically extended with band gap materials. Also in the theory [25], simple computational structures are being realized; e.g. the medium of automata M, shown in Figure 1 (a), can perform any computation when $M = M_{110}$; this very simple 3-state automaton is equivalent (by de Bruijn graph

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 66-85, 2002.

- conversion) to rule 110 1D binary CA which is conjectured to be computationally universal (no proof for this has yet appeared in scientific journal).
- New models, like ball moving algorithms [16, 21, 45, 46, 48, 51, 52], that support discrete coherent structures show that filtrons indeed represent the solutions of discrete versions of classical solitonic wave equations (KdV, KP, L-V). This suggests deep and relevant connections between the computational processes occurring within the nets of automata (represented by IAMs) and the equations of motion of nonlinear dynamical systems.
- All known models that support discrete coherent structures can be described by IAMs [33-41]. This indicates that the iterating of automata maps is the fundamental mechanism that creates localized soliton-like periodic structures.

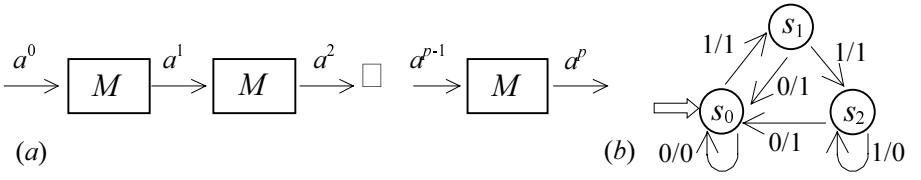


Fig. 1. (a) IAM as an automaton medium; $M(a^t) = a^{t+1}$. (b) Automaton $M_{110} \equiv$ rule-110-CA.

Investigating of IAMs was started by H. Takahashi in 1976 [50]. Analyzing the limit sets of CAs, he proposed a procedure of determining the iterated words (of fixed length) supported by given finite state automaton. Then, similar topic was undertaken by G. Paun in [31, 32], but mainly languages aspects were analyzed.

In 1986, a new model—filter CAs—was introduced [30]. This work founded a bridge between automata and nonlinear physics, presenting the discovery of discrete binary soliton-like entities emerging under particular IAMs. A number of similar models and some new ones were introduced since that time. These are:

- iterated arrays [3, 7],
- classical CAs [4, 8] and higher order CAs [2, 8, 36, 47],
- filter CAs, introduced in [30] and discussed in [1,10, 14, 15, 17, 24, 27, 28, 42, 53],
- soliton CAs considered in [11, 12, 13, 47, 49, 53],
- sequentially updated CAs [5, 9]; their dynamics depends on an updating schedule,
- fast rules [1, 17, 24, 27, 29],
- digital recursive (IIR) filters [30, 35] and filter automata [33 – 41],
- integrable CAs [6, 19, 21],
- discrete versions of classical soliton equations (KdV, KP, L-V) [20, 48, 52],
- box-ball systems introduced in [45, 46], and considered in [16, 20, 21, 51, 53],
- crystal systems [19, 20], and combinatorial processing [18, 19, 20, 53],
- affine Lie algebras [18, 19, 20].

Some of these models are widely used in computer sciences, but others have been constructed to describe the nonlinear wave phenomena, especially solitons behavior. Here, we will show that all these models have their, relatively simple, automata equivalents, but sometimes with very particular organization of memory.

2 Automata, Filter Automata Family, and Filtrons

We consider serial SP using Mealy transducers. A Mealy type automaton M with outputs and an initial state s_0 is a system $M = (S, \Sigma, \Omega, \delta, \beta, s_0)$, where S, Σ and Ω are nonempty finite sets of —respectively— states, inputs and outputs, $\delta: S \times \Sigma \to S$ is called the next state (or transition) function of M, and $\beta: S \times \Sigma \to \Omega$ is called the output function of M. The automaton converts sequences of symbols, preserving their length. Any input string is read sequentially from left to right, one symbol at each instant τ of time, in such a way that $\delta(s(\tau), \sigma(\tau)) = s(\tau+1)$ and $\beta(s(\tau), \sigma(\tau)) = \omega(\tau)$; $\tau = 1, 2, \ldots$. To allow the iterations we apply a unified set $A = \Sigma = \Omega = \{0, 1, ..., m\}$.

A finite string given at some time t is denoted by $a^t\_ = a_1^t\ a_2^t \ldots a_{L_t}^t$. Usually we consider longer strings of zeroes, infinite or periodic, which contain a given finite string as its substring; $a^t = ...0\ a^t_0... $. Any such string is called configuration a^t. The conversion $a^t \to a^{t+1}$ of the entire configuration performed by the automaton M is treated as a single step of automaton map; step number $(t+1)$. We write $M(a^t) = a^{t+1}$.

The operation of M may be described by a sequence of functions $f_s: A \to A$ (called state-implied operations) such that $f_s(a_i) = \beta(s, a_i)$ for all $s \in S$ and $a_i \in A$. Then, the succession of automaton's outputs is determined by $next\ [f_s(a_i)] = f_{\delta(s,\ a_i)}(a_{i+1})$. Input strings imply the labeled paths on state diagram of the automaton, and these paths can be identified by the appropriate sequences of operations f_s.

Iterating the automata maps over a^0, we list successive strings a^0, a^1, … one under another; this forms space-time (ST) diagram of processing. Let us mention, that such ST diagram is a special case of the phase space of dynamical systems. Sometimes we apply a shift q to the ST diagram, which means that each output string is shifted by q positions to the left with respect to the input string.

We assume that two different models of SP are equivalent (≡) if for any input string a^t they both return the same result a^{t+1} (the applied shifts q are disregarded); we write $a^{t+1} = M_1(a^t) = M_2(a^t)$.

The functions implied by states allow us to define special class of cyclic sequential SP mechanisms—the family of filter automata (FAs) [33, 34]. Filter automaton $F = (A, \iota, \mu, \kappa)$ is the sequential transducer (converts strings in serial) where:

- $A = \{0, 1, ..., m\}$ is a finite set, called the alphabet of F, with $m > 0$,
- $\iota: A \to \{0, 1\}$ is a mapping, called the activating function of F,
- $\mu: A^{\le r+1} \to I$ is a partial mapping $(r > 0)$ into a set of indexes; $I = \{0, 1, ..., m'\}$,
- $\kappa = (h_0, h_1, ..., h_{m'}, f_1, ..., f_r)$ is a sequence of mappings $A \to A$, called the program of automaton F.

The resulting string is generated in time instants τ according to the following cycles:

$$next\ [h_0(a_\tau)] = \begin{cases} h_0(a_{\square\square 1}) \text{ if } \iota(a_\square) \neq 0 \\ f_1(a_{\square\square 1}) \text{ if } \iota(a_\square) \neq 0 \end{cases}, \quad \text{and } c := a_\tau, \tag{1}$$

$$next\ [h_i(a_\tau)] = \begin{cases} h_j(a_{\square\square 1}) \text{ if } \mu(a_\square) \text{ is defined} \\ f_1(a_{\square\square 1}) \text{ otherwise} \end{cases}, \text{ where } j = \mu(a_\tau) \text{ and } 0 < i \le m', \tag{2}$$

$$next\,[f_i(a_\tau)] = \begin{cases} h_j(a_{\square\square 1}) \text{ if } \square(a_\square) \text{ is defined} \\ \square \\ f_{i\square 1}(a_{\square\square 1}) \text{ otherwise} \end{cases}, \quad \text{where } j = \mu(a_{i+\tau}) \text{ and } 0 < i < r, \tag{3}$$

$$next\,[f_r(a_\tau)] = \begin{cases} h_j(a_{\square\square 1}) \text{ if } \square(a_\square) \text{ is defined} \\ \square \\ h_c(a_{\square\square 1}) \text{ otherwise} \end{cases}, \quad \text{where } j = \mu(a_{r+\tau}). \tag{4}$$

Here, $a_{i+\tau} = (a_\tau, a_{\tau+1}, ..., a_r)$ denotes an $(i+1)$-segment (substring of a string) which coincides with a current cycle of the automaton's activity, and $A^{\le r+1}$ denotes the set $A \cup A^2 \cup ... \cup A^{r+1}$. Variable c is a memory of current activating symbol of F [14].

Filter automata perform SP using the cycles $(h_0, f_1, f_2, \ldots, f_r)(h_j, f_1, f_2, \ldots, f_r)\ldots$ of operations. The processing runs until a reset condition is met; this is any $(L+1)$-tuple that coincides with a cycle and such that $a\_ \in A^{L+1}$, $0 \le L \le r$, and $\mu(a\_) = 0$.

We consider the automata that have two modes of operation. They are either active or inactive. The automaton is said to be activated when it leaves the state s_0 and starts processing (f_1 follows h_0, in the case of F). Also, it is said to be extinguished when it is forced to return to the initial s_0 state (in the case of F its activity cycle is closed and h_0 operation follows).

We use the activity mode concept of the automata to distinguish some special segments of strings. Let an automaton M converts a configuration $a^t = ...0\ a^t\_0...$. Any string $a^t\_ = a_1^t\ a_2^t \ldots a_{L_t}^t$ ($a_1^t \ne 0$) such that a_1^t activates M and $a_{L_t}^t$ extinguishes M is said to be M-segment. When a configuration a^t is transformed by the automaton M, it may happen that a number of extinctions of M still occurs before the last element of the string segment $a^t\_$ is read by M. In such a case we say that $a^t\_$ is a multi-M-segment string. Such strings lead to the complex filtrons and their classification [33].

Here, the state s_0 has been chosen as starting and also as final state to recognize an M-segment by automaton. However, in general, another selection is possible and even the subsets of automaton states can be chosen as starting states and/or as final ones.

To show filtrons on the ST diagrams we assume periodic boundary conditions and use the following convention. Symbol $0 \in A$ typically represents a quiescent signal or a background, but sometimes it appears within an M-segment. Thus, we use three different characters to present it on the ST diagrams. A dot "." denotes zeros read by the automaton M, which is inactive. A dash "-" represents tail zeros of an M-segment that is all consecutive zeros preceding immediately the extinction of M. Remaining zeros are shown as the digit 0. Additionally, all those symbols that activate the automaton are printed in bold. Our convention allows one to recognize easily whether any two filtrons are distant, adjacent or interact and form a complex object.

Example 1. Consider the filter automaton $F_1 = (A, \iota, \mu, \kappa)$, $A = \{0, 1, 2, 3\}$, $m' = 1$ and $r = 2$. Also $\iota^{-1}(1) = \{1, 2, 3\}$, $\mu : A^3 \to A$ is such that $\mu(w) = 0$ if $w \in \{*00\}$ and $\mu(w) = 1$ otherwise. The operations of $\kappa = (h_0, h_1, f_1, f_2)$ are: $h_0 = \begin{matrix} \square 0123 \square \\ \square 0203 \square \end{matrix}$, $h_1 = \begin{matrix} \square 0123 \square \\ \square 1203 \square \end{matrix}$, and $f_1 = f_2 = \begin{matrix} \square 0123 \square \\ \square 0312 \square \end{matrix}$. The cycles $(h_0, f_1, f_2)(h_1, f_1, f_2)...(h_1, f_1, f_2)$ are performed over each

configuration, starting from the first leftmost nonzero element until a segment from {∗00} (* denotes any symbol) coincides with some cycle. The IAM of this automaton is illustrated on the ST diagram below for some initial configuration a^0. One can check the filtron 2301--. When F_1 reads the configuration ...02301000... the associated sequence of operations is: ...$h_0\ h_0\ f_1\ f_2\ h_1\ f_1\ f_2\ h_0$... . This sequence implies the output configuration of the form: ... $h_0(0)\ h_0(2)\ f_1(3)\ f_2(0)\ h_1(1)\ f_1(0)\ f_2(0)\ h_0(0)$... or ...00202000... . This is seen on the ST diagram. Two cycles of activity of automaton F_1 are involved, (h_0, f_1, f_2) and (h_1, f_1, f_2). The second one is the extinction cycle because its associated segment, 100, belongs to the set {*00} of reset conditions.

```
 0   ------------11------3-----111---132-----23031-----231
 1   1-----------2301----3-----2331--2211-----203301----23
 2   31----------202----3------222---132-------23031----2
 3   231-----------110003-------111002211-------203301---
 4   0231----------2301021------2331010331--------2303100
 5   013301----------202012-------222031222---------203301
 6   1323031----------11311-------111230111---------2303
 7   320203301---------2322301-----2330202331---------203
 8   31---23031---------210202------2211--222-----------2
 9   231---203301--------3---11------132---111-----------
10   ·231----23031-------3---2301----2211--2331----------
11   ··231----203301-----3----202-----132---222----------
```

ST diagrams of IAM strings processing exhibit all kinds of behavior typical of dynamical systems; one can see then the fractals or periodic and chaotic dynamics. But sometimes, particular moving coherent structures capable of non-demolition collisions emerge also (as it is seen on ST diagram above where three pairs of objects collide). We say, in brief, that automata can support coherent structures. These objects seem to be of fundamental significance; they are very specific and inherently associated with the IAMs. Since the IAM is a kind of recursive filtering, a new term—filtrons has been introduced [34] for these coherent objects.

The filtron is defined as follows [34, 38, 41]. By a p-periodic filtron $a^t\_$ of an automaton M we understand a finite string $a_1^t\ a_2^t \ldots a_{L_t}^t$ ($a_1^t \neq 0$) of symbols from A such that during the iterated processing of configuration a^t = ...0 a^t_0... by automaton M the following conditions are satisfied for all t = 0, 1, ..., p–1:

1. The string $a^t\_$ occurs in p different forms (orbit [30] of the filtron), with $0 < L_t < \infty$.
2. The string $a^t\_$ is an M-segment.

These conditions can be easily verified on the ST diagram of string processing; usually the periodic structure of coherent entities is clearly visible.

What is interesting, there are also some objects which form M-segments and rest coherent for a very long time, but which finally appear to be nonperiodic; they transform or decay. These are quasi-filtrons [34].

3 Automata Based on the FCA Window

The first automata supporting coherent binary entities were parity rule filter CAs [30]. This model consists of a specific ST window, which we call here the FCA (filter CA) window, and the parity Boolean (updating) function f_{PST}. The model performs serial SP, thus is a kind of sequential transducer. In the domain of signal processing, most sequential transducers are classified as IIR filters, hence we will use also the term PST filter. The PST filter is defined as follows [30]. Assume a configuration at time t,

$a^t = ... \; a_i^t \; ... = ...0\, a_1^t \; ... \, a_{L_t}^t \; 0...$, of elements from $A = \{0, 1\}$ such that $0 \le t < \infty$, $-\infty < i < \infty$, $1 \le L_t < \infty$ and $a_1^t \neq 0$. For a fixed $r \ge 1$, the model (f_{PST}, r) computes next configuration a^{t+1} at all positions i ($-\infty < i < \infty$) in such a way that:

$$a_i^{t+1} = f_{\text{PST}}(a_i^t, ..., a_{i+r}^t, a_{i+r}^{t+1}, ..., a_{i+1}^{t+1}) = \begin{cases} 1 \text{ if } S_{i,t} \text{ is even but not zero} \\ 0 \text{ otherwise} \end{cases} \quad (5)$$

where $S_{i,t} = \sum_{j=1}^{r} a_{i+j}^{t+1} + \sum_{j=0}^{r} a_{i+j}^{t}$.

Zero boundary conditions are assumed; it means that the segment $a^t\_ = a_1^t \, ... \, a_{L_t}^t$ is always preceded at the left side by enough zeros and the computation may start from $0 = f_{\text{PST}}(0, ..., 0)$. Space-time positions of the arguments of function f_{PST}, which form the FCA window, are shown in Figure 2 (a). It is seen that the memory structure of the PST model is based on two shift registers.

The PST model performs a cyclic processing and belongs to filter automata family. We have the following basic generalization.

Proposition 1 [33]. PST model (f_{PST}, r) belongs to FA family. The equivalent automaton $F_r = (A, \iota, \mu, \kappa)$ is such that $A = \{0, 1\}$, $m = m' = 1$, $a = 1$ is the only activating element, $\mu^{-1}(0) = \{*0^r\}$, and $\kappa = (h_0, h_1, f_1, ..., f_r) = (0, \text{N}, \text{A}, \ldots, \text{A})$ where $0 = \begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}$, $\text{N} = \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}$, and $\text{A} = id$.

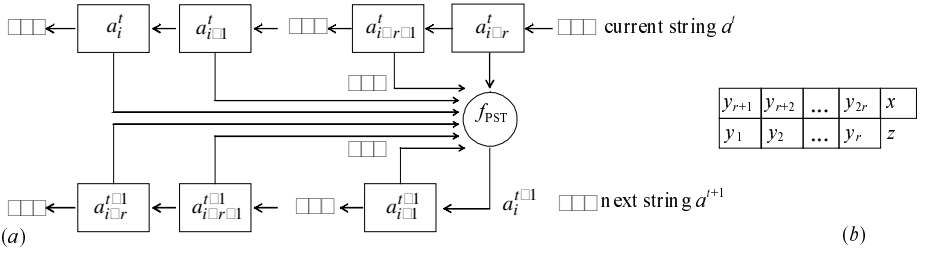


Fig. 2. (a) Window of the PST model slides to the right (τ moments), and implies $q = r$. (b) PST model as an automaton; x – input variable, $y_1, y_2, \ldots, y_{2r}$ – state variables, z – output variable.

For a proof [33, 35], the FCA window is decomposed: x represents input symbols of the automaton F_r, y_1, y_1, ..., y_{2r} represent state variables, and z represents output symbols. Next states follow from the movement of the window, especially $y_r(\tau + 1) = Y_r = z$. The function f_{PST} that computes the outputs z of automaton F_r is $z = y_1 \oplus y_2 \oplus ... \oplus y_{2r} \oplus \bar{x} \oplus \bar{y}_1 \bar{y}_2 \ldots \bar{y}_{2r} \bar{x}$ where $\oplus$ is *XOR* operation, and the variables are from Figure 2 (*b*). The operations negate (N) and accept (A) are performed in cycles.

3.1 Other Systems Based on FCA Window

After publishing the PST string processing mechanism some other models, based on the FCA window, were proposed in the literature. We identify them by the initials of their authors. First, Goldberg [17], analyzed formally the PST filters showing their sequential nature and giving the idea of speeding up their computations (Rapid Updating Rule). The models AKT [1] and TS [49] were derived by modifications of the updating function of PST model. The J filtering, described in [24], used a wider FCA window and a new updating function. New alphabets A, with $|A| > 2$, have been applied with the models AKT [1] and FPS [14] starting the analysis of multi-valued signals (configurations).

Another approach in searching for filter CAs supporting the filtrons exploited algebraic methods. In [10], all possible updating functions were verified (for the FCA window with $r = 2$) and the FM filter supporting filtrons was found. The FPS [14] and F [15] filters used the operations based on algebraic groups. The TS filter from [49] has been described [53] by means of a combinatorial procedure (Young tableaux and stack language) associated with operations over a stack.

Table 1. Automata equivalents of serial SP models with FCA type windows; $y_r(\tau+1) = Y_r = z$.

Model	Ref.	$\|A\|$	r	w	FA	Output function $\beta: A^w \times X \to A$
PST	[30]	2	>0	$2r$	$\in$	$z = (T > 0) \wedge (\overline{P})$
FPS	[14]	m	>0	$2r$	$\in$	Follows from the operation of group $G_m = A$
AKT	[1]	2	>0	$2r$	$\notin$	$z = \overline{y_1}\ \overline{y_2} \ldots \overline{y_r}\ \overline{y_{r+2}}\ \overline{y_{r+3}} \ldots \overline{y_w}\ \overline{x} \oplus P$
J	[24]	2	>0	$2r+1$	$\in$	$z = \overline{y_1}\ \overline{y_2} \ldots \overline{y_r}\ \overline{y_{r+2}}\ \overline{y_{r+3}} \ldots \overline{y_w}\ \overline{x} \oplus y_{r+1} \oplus P$
TS	[49]	2	∞	∞	$\notin$	$z = (y_{r+1} = 0) \wedge ((y_{r+2} + \ldots + y_\infty) > (y_r + \ldots + y_{-\infty}))$
F	[15]	m	>0	$2r$	$\in$	Follows from the operation of group $G_m = A$
BSR-1	[6]	m	2	$2r$	$\notin$	$z = y_3 \oplus y_1 y_4 \oplus y_2 x$
$M_{CF2,1}$	[37]	2	2	$2r$	$\notin$	$z = y_3 \oplus y_2 y_4 \oplus y_1 x$
BSR-2	[6]	m	3	$2r$	$\notin$	$z = y_4 \oplus y_2\ \overline{y_3}\ \overline{y_6} x \oplus y_1\ \overline{y_2}\ \overline{y_5} y_6$
BSR-22	[6]	2	2	$2r$	$\notin$	$z = y_3 \oplus\ \overline{y_1} y_2 \oplus y_4\ \overline{x}$
BSR-23	[6]	3	3	$2r$	$\notin$	$z = y_4 \oplus y_1 y_2 \oplus y_2 y_3 \oplus y_5 y_6 \oplus y_6 x$
$M_{CF3,6}$	[37]	2	3	$2r$	$\notin$	$z = y_4 \oplus y_2 y_6 \oplus y_3 y_5 \oplus y_1 x$
BSR-24	[6]	2	2	$2r$	$\notin$	$z = y_3 \oplus\ \overline{y_1} y_2 \oplus y_4\ \overline{x} \oplus y_2 y_4$
FM	[10]	2	2	$2r$	$\in$	$z = x\ \overline{y_1}\ \overline{y_2}\ \overline{y_3}\ \overline{y_4} \oplus \overline{x} \oplus y_1 \oplus y_3$

Looking for FCA models capable of supporting the filtrons was also undertaken by an analysis of the integrability of some equations of motion or wave equations. This approach is represented in [6] where new families of BSR-j filters have been derived (j refers to the number of defining formula in [6]). In [37] it was shown that a number of versions of the BSR-j filters could be obtained using S_{2r+1}, the group of permutations over arguments of basic updating function. In Table 1 two examples of such versions, $M_{CF2,1}$ and $M_{CF3,6}$, are presented.

In Table 1 [37, 41] we list a description of automata equivalents of all currently known filter CAs supporting the filtrons, except those derived immediately from FA family. All these classical models were originally defined by the recursive formulae. Here we describe them by explicit automaton output functions. Next states follow from window's movement and the substitution $y_r(\tau+1) = Y_r = z$. In the table, P is the *mod* 2 sum of all window's elements (thus $\overline{P}$ denotes parity), and T is the algebraic sum of all these elements.

It is seen also that not all the systems, which apply FCA windows belong to the FA class.

3.2 Fast Rule Algorithms

Fast Rule Theorem (FRT) type algorithms have been given in [17, 27] as an equivalent explicit formulation of parity rule filter CAs. Later, they have been extended to the binary AKT filters and J filters [1, 24, 29].

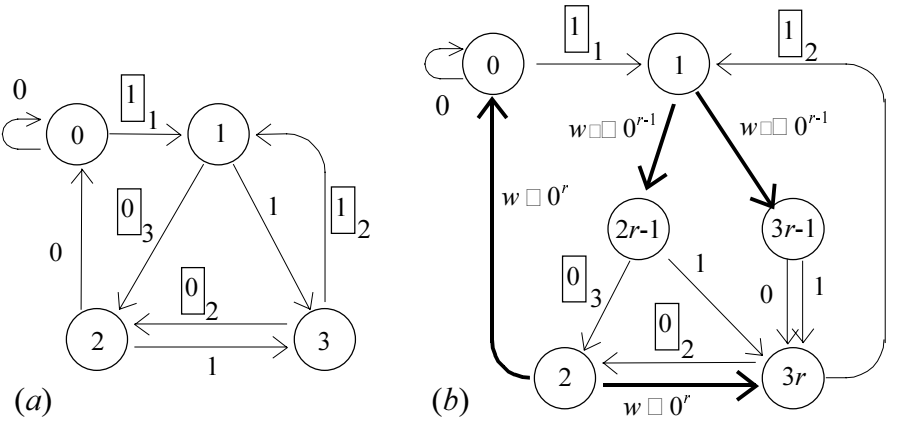


Fig. 3. (a) FRT algorithm for the case $r = 1$. (b) FRT algorithm for the case $r > 1$.

For every configuration a^t the FRT of AKT filter proceeds in two phases. During the first phase a^t is examined from the left, and a subset $B(t) \subset Z$ of positions (so called boxes) is identified. Then, during the second phase, the resulting configuration a^{t+1} is derived simply by complementing the symbols in boxes and shifting all symbols by r positions to the left. Note that this implies $q = r$ for this model.

There are three steps in the first phase of the FRT algorithm. Assume that $A = \{0, 1\}$, positive integer r of AKT filter is given, and the current configuration a^t is analyzed from left to right. The algorithm starts with $B(t) = \varnothing$.

1. Place the first encountered position i such that $a_i^t \neq 0$ in the set $B(t)$, and proceed to Step 2. If there are no such positions go to the phase 2.

2. Check the r-segment w (word) consisting of r symbols to the right from the current box position i;

- if $w \neq 0^r$ then include the position $j = i+r+1$ to the set $B(t)$, and repeat Step 2,
- if $w = 0^r$ (there is the block of r zeros) then go to Step 3.

3. Check the symbol a_i^t;

- if $a_i^t = 1$ then include the position $j = i + r$ to the set $B(t)$, and go to Step 2,

- if $a_i^t = 0$ then return to Step 1.

Looking for the set $B(t)$ lead to the block schemata shown in Figure 3; the step numbers used as box indices indicate how boxes are generated by the algorithm. E.g. for $r = 3$ one has: . . . $0\boxed{1}_1 0\,0\,1\,\boxed{0}_2 0\,0\,0\,0\,\boxed{1}_1 0\,0\,\boxed{0}_3 0\,0\,1\,\boxed{0}_2 0\,0\,1\,\boxed{1}_2 0\,0\,\boxed{0}_3 0\;0\,0\,0 \ldots$.

Having these schemata, the state diagrams of equivalent Mealy automata $M_{\mathrm{AKT}r}$ can be determined. To obtain the function δ of $M_{\mathrm{AKT}r}$ when $r > 1$, one has to 'unroll' bold paths from Figure 3 (b). These are: $w' = 0^{r-1}$ for the transition $\delta(1, w') = (2r\text{-}1)$, $\overline{w}'$ for $\delta(1, \overline{w}') = (3r\text{-}1)$, $w = 0^r$ for $\delta(2, w) = 0$ and also $\overline{w}$ for $\delta(2, \overline{w}) = (3r)$. The resulting automata do not belong to FA class. The details are given in [35, 36].

3.3 Soliton Cellular Automata

The model called soliton cellular automaton has been introduced in [49]. Its underlying FCA window has an infinite length. The model operates as follows.

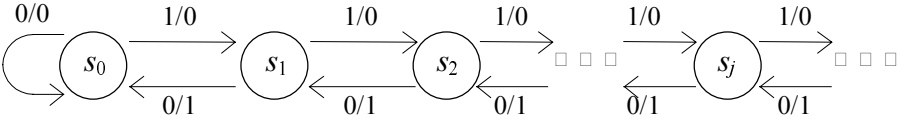


Fig. 4. Infinite Mealy automaton M_{TS} which is equivalent to the TS system.

Assume a configuration $a^t = ...0\, a_1^t \ldots a_i^t \ldots a_{L_t}^t\, 0...$ of elements from $A = \{0, 1\}$ where $0 \le t < \infty$, $-\infty < i < \infty$, $1 \le L_t < \infty$ and $a_1^t \neq 0$. The next configuration is computed for all positions i ($-\infty < i < \infty$) sequentially from the left hand side according to $a_i^{t+1} = f_{\mathrm{TS}}(a_{i-1}^t, ..., a_{i+1}^t, a_i^t, a_{i-1}^{t+1}, ..., a_{i+1}^{t+1})$ where the function f_{TS} is given by:

$$a_i^{t+1} = \begin{cases} 1, & \text{if } a_i^t \neq 0 \text{ and } S_{i,t} > S_{i,t+1}, \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

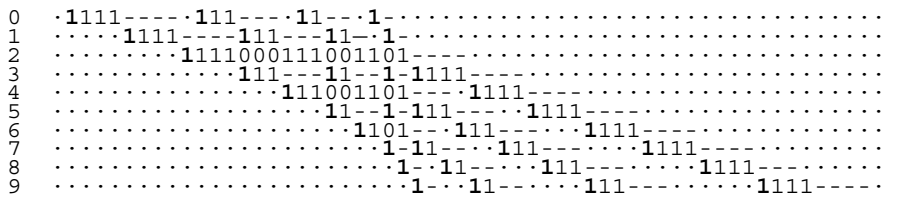
where $S_{i,t} = \sum_{j=1}^{i} a_{i-j}^t$ and $S_{i,t+1} = \sum_{j=1}^{i} a_{i-j}^{t+1}$.

This model has an infinite capacity of memory; we have the following property.

Proposition 2 [39, 41]. The automaton M_{TS} that is equivalent to TS model is a difference counter of the infinite capacity; states are incremented under input 1, and decremented under 0.

Proof. We choose the states $\{s_j\}$ of M_{TS} such that $j = r_{i,t}$, where $r_{i,t} = S_{i,t} - S_{i,t+1}$. Then for $j = 0$ one has $\delta(s_0, a) = s_{0+a}$ and $\beta(s_0, a) = 0$, and for $j > 0$ there is $\delta(s_j, 0) = s_{j-1}$ and $\delta(s_j, 1) = s_{j+1}$ and also $\beta(s_j, a) = \overline{a}$; $a \in A$.

The counter is shown in Figure 4, and the ST diagram of an exemplary associated IAM dynamics is given below.



4 Box-Ball Systems

For the first time the box-ball systems (BBSs) were proposed as models that support the soliton phenomena by D. Takahashi in [44, 45]. Later this class of models was enlarged and discussed, among others in [16, 19, 21, 48, 51, 52]. The basic idea of using BBSs to serial SP is to associate the balls with each symbol a_i in some way, and then to perform one step of evolution $a^t \rightarrow a^{t+1}$ by moving all balls. The processing mechanism runs once along given string from the left hand side to the right and moves all balls once. The new location of a ball is the first encountered empty box; typically zero symbol means an empty box.

4.1 Single Ball Boxes

The simplest BBS models assume that each nonzero symbol a_i of a string a^t over alphabet $A = \{0, 1, ..., m\}$ represents a *single ball* with a label $0 < a_i \leq m$ (symbols represent the variety of balls). This model is denoted B1BS (box-1-ball system).

The B1BS rules to transform a^t into a^{t+1} are as follows [45]. Move every ball exactly once by the procedure (a)-(b), first move the balls number 1, then the balls number 2, and so on:

(a) move the leftmost ball, not yet moved, to the nearest right empty box,

(b) repeat this procedure until all balls are moved.

Note that for $m = 1$, the B1B system reduces to TS system [45, 53]. For $m > 1$, by the definition all balls are moved according to ascending order. Later, in [16] the reformulated BBS with the descending order was proposed. This suggests that any order α of nonzero symbols from A could be applied. Thus, we extend the original B1B system by adding to it some permutation $\alpha: A' \rightarrow A'$, where $A' = \{1, 2, ..., m\}$. We have then B1BS-α model. Now the balls are moved according to the order determined by permutation α.

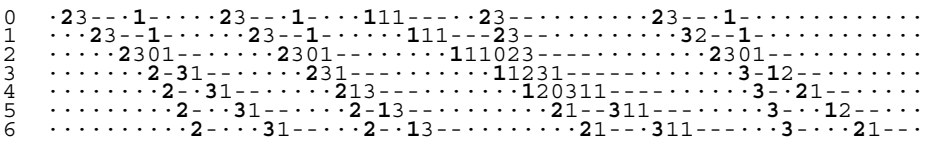
Proposition 3. The automaton equivalent of B1BS-α model, $M_{\text{B1BS-}\alpha}$, is such that its set S of states consists of the sorted lists of symbols from A' already read but not yet returned to the output string. The initial state s_0 represent empty list, and $\delta(s_0, 0) = s_0$. For any $s \neq s_0$ and $a \in A$ the output symbol $\omega = \beta(s, a)$, which is released from list, is either the first element of list greater than a (with respect to α) or it is zero if there is no such element. In this last case symbol a is included into list if $a \neq 0$.

Proof. Looking for a free box to release a ball by means of a one-way sequential procedure requires a memory (or a nonzero state s) to remember all predecessing balls not yet released. The symbols are moved according to some order α, and $m > 1$, thus

any state s must represent a sorted list. Also, the number of consecutive nonzero symbols in a string is unbounded in the model B1BS-α, thus the unbounded lists are necessary. In this case the automata $M_{\text{B1BS-}\alpha}$ have an infinite memory.

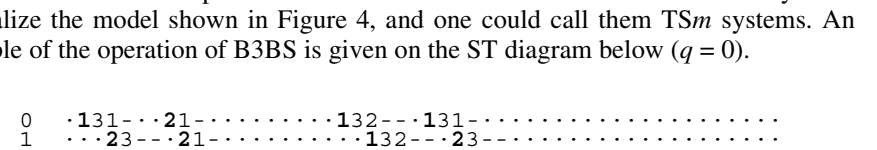
Remark 1. Other special cases are also possible. E.g. no order of symbols is decided, and they are moved in the order as they appear running from the left to the right. This type of memory is known as FIFO (first in, first out) queue. Also, one can decide the reversed order. This implies immediately the memory structure known as stack. Thus we distinguish the automata $M_{\text{B1BS-}i}$, $M_{\text{B1BS-}d}$, $M_{\text{B1BS-}\alpha}$ (increasing, decreasing or given order α) or M_{B1BS} if no order is specified, and M_{B1BSr} if the stack is applied.

Below, we show four ST diagrams (with $q = 0$) of iterated string processing ($A = \{0, 1, 2, 3\}$) performed by automata M_{B1BS}, $M_{\text{B1BS-}i}$, $M_{\text{B1BS-}d}$ and M_{B1BSr}.



4.2 Multiple Ball Boxes

Now, let us consider the models BBSs such that each symbol $0 < a_i \leq m$ is interpreted as the *number* of balls contained in a box at position i, while the capacity of boxes is bounded by the integer m. This interpretation was proposed in [45]. We denote this models by BmBS (box-m-balls system). The BmB systems have simple automata equivalents. Such automata have to use the finite counter as their memory. The counter is decremented immediately when it is possible and as much as it is possible, which means that the operation *min* determines its behavior. Thus BmB systems generalize the model shown in Figure 4, and one could call them TSm systems. An example of the operation of B3BS is given on the ST diagram below ($q = 0$).



We have [39, 41] the following explicit description of automata $M_{\text{B}m\text{BS}}$.

Proposition 4. The automaton equivalent, $M_{\text{B}m\text{BS}}$, of the BmB system over $A = \{0, 1, ..., m\}$ is given by the next state function $s' = \delta(s, \sigma) = s + \sigma - \omega$ and the output function $\omega = \beta(s, \sigma) = \min(s, \overline{\sigma})$, where $\overline{\sigma} = m - \sigma$. The initial state is $s_0 = 0$.

4.3 Box-Ball Systems with Carrier

Yet another family of BBSs has been introduced in [48]. This model is called box-ball system with carrier; let us denote it by BBSC. This is a modification of BmBSs such that the mechanism carrying the balls (called carrier) has the capacity restricted to n. Thus, while the carrier passes the box it puts as many balls into the box as possible and simultaneously takes as many balls from the box as it can.

The action of the carrier over the j-th box is as follows. Assume that the carrier carries $0 \le c_j \le n$ balls and there are $0 \le a_j \le m$ balls in the box. This means that there are $n - c_j$ vacant spaces for balls in the carrier, and $m - a_j$ vacant spaces for balls in the box. Thus the carrier puts $\min (c_j, m - a_j)$ balls into the box and gets $\min (a_j, n - c_j)$ balls from the same box. By this action the number of balls in the j-th box changes from a_j to $a_j + \min (c_j, m - a_j) - \min (a_j, n - c_j)$, and simultaneously the number of balls carried by the carrier changes from c_j to $c_{j+1} = c_j + \min (a_j, n - c_j) - \min (c_j, m - a_j)$.

From the above one can identify immediately the automata $M_{\mathrm{BBSC}}(m, n)$ equivalent to BBSC model with parameters m and n.

Proposition 5 [38, 39]. The automata $M_{\mathrm{BBSC}}(m, n) = (S, \Sigma, \Omega, \delta, \beta, s_0)$ are such that:

- $S = \{0, 1, ..., n\}$ is the finite set of integers—the set of states of $M_{\mathrm{BBSC}}(m, n)$,
- $\Sigma = \Omega = A = \{0, 1, ..., m\}$ is the finite set of integers (unified alphabet),
- $s_0 = 0$ is the initial state of $M_{\mathrm{BBSC}}(m, n)$,
- the next states are given by $s' = \delta(s, \sigma) = s + \min (\bar{s}, \sigma) - \min (s, \bar{\sigma})$, and
- the output symbols are determined by $\omega = \beta(s, \sigma) = \sigma + \min (s, \bar{\sigma}) - \min (\bar{s}, \sigma)$, where $\bar{s} = n - s$ and $\bar{\sigma} = m - \sigma$.

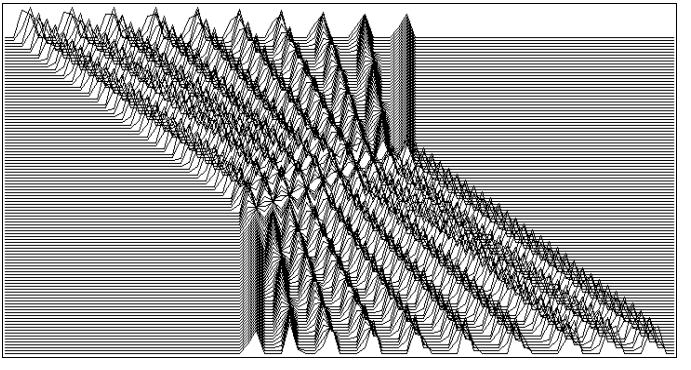


Fig. 5. Multi-filtron nondestructive collision of ten objects of automaton $M_{\mathrm{BBSC}}(17, 25)$; $q = 1$.

An example of IAM performed by the automaton $M_{\mathrm{BBSC}}(17, 25)$, where $a^0 = \ldots 0\mathrm{H}90^4\mathrm{G}90^4\mathrm{F}90^4\mathrm{E}90^4\mathrm{D}90^4\mathrm{C}90^4\mathrm{B}90^4\mathrm{A}90^4990^4890\ldots$, is shown in Figure 5.

5 Crystal Systems

After introducing the box-ball systems [45, 46, 48], a new domain has been opened—crystal base theory—as being closely related to soliton phenomena. Some

studies [16, 19, 20, 21] extended BB systems and applied them or reformulated in crystal theory.

Among the crystal bases that are considered there are two classes called crystals B_k and crystals C_k. These models represent finite systems and lead to finite state automata $B(m, k)$ and $C(m, k)$, respectively. Namely, the states of equivalent automata are represented by sorted lists of symbols from A, but the lengths of lists are bounded by a given positive integer k.

Formal definitions of crystal systems are not presented here, since they need a number of auxiliary notions like tensor algebra and combinatorial R matrices. Let us make a comment that in crystal approach the strings are identified with the set of tableaux of shapes, and some (isomorphic) maps evolve the strings and are used to define the crystal forms. The classes B_k and C_k can be reduced to the finite cases of simple models B1BSs [16, 20, 41]. Thus the automata $B(m, k)$ and $C(m, k)$ (denoted in [41] by $M_{\text{BBS-GF}}$ and $M_{\text{BBS-LF}}$, respectively), can be obtained immediately. We present these automata here without a formal proof.

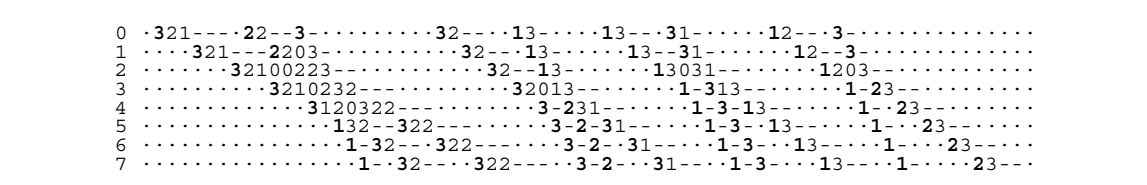
Proposition 6 [39]. The automata equivalents, $B(m, k)$ and $C(m, k)$, for crystal models B_k and C_k have the states represented by the lists L of k symbols from A. Initial state s_0 is the empty list with all entries equal to 0. The lists are sorted in descending (the greatest first) or ascending (the least first) orders, respectively. Next states $L' = \delta(L, \sigma)$ and outputs $\omega = \beta(L, \sigma)$ of automata are determined by the following operations over lists $(L, \sigma) \rightarrow (L', \omega)$:

(a) the automata $B(m, k)$; GF (greatest first) case:
- *remove* an element from the list; e.g. $(124, 0) \rightarrow (012, 4)$,
- *insert* a symbol to the list, e.g. $(013, 1) \rightarrow (113, 0)$,
- *replace* ω by σ (where ω is immediate smaller than σ); e.g. $(124, 3) \rightarrow (134, 2)$,

(b) the automata $C(m, k)$; LF (least first) case:
- *remove* an element from the list; e.g. $(032, 0) \rightarrow (003, 2)$,
- *insert* a symbol to the list, e.g. $(032, 3) \rightarrow (332, 0)$,
- *replace* ω by σ (where ω is immediate bigger than σ), e.g. $(032, 1) \rightarrow (031, 2)$.

Remark 2. Similarly as in systems BmB one can consider also the bounded lists that are sorted according to a given order α. Following this line, if no order is specified, we have the model with FIFO memory bounded to k. And also the automata with a stack memory bounded to k can be constructed easily.

To illustrate the IAM for some (m, n), we show below two ST diagrams generated by automaton $B(3,3)$, and (at the right side) two ST diagrams of automaton $C(3,3)$.



A qualitatively new model of crystal systems has been introduced in [21]. The restricting parameter k was allowed to be different for various symbols. E.g. it was assumed that a symbol 1 can appear in any unbounded number (even infinite) of

copies, while at the same moment the other symbol, 2, was allowed to exist in bounded number of copies. Such symbols were called *bozonic* and *fermionic* symbols, respectively. The automata that are equivalent to such crystal systems form a new class of sequential transducers; they are finite for some inputs, and simultaneously infinite for the others. We have immediately the following.

Proposition 7 [41]. The bosonic-fermionic crystal systems have the automata equivalents with mixed types of counter memory: infinite for bozonic symbols and finite for fermionic symbols.

An example of the automaton that represents crystals with *bozonic* and *fermionic* symbols 1 and 2 is shown in Figure 6.

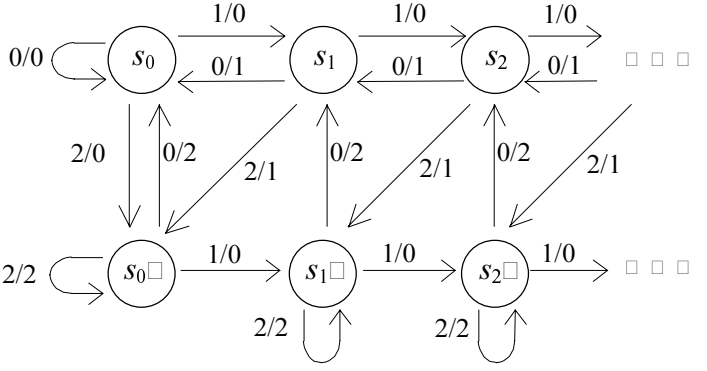


Fig. 6. Automaton with a mixed counter memory: infinite for 1′s and finite for 2′s.

ST diagram below illustrates the IAM performed by the automaton from Figure 6.

```
0 ·211-------1----211--------2----111-----21--2----------------
1 ····211-----1------211-----2------111----2102---------------
2 ·------211---1--------211---2--------111---212--------------
3 ··········21101----------21102----------11100221-----------
4 ·----------2-111----------2121-----------1112021----------
5 ··----------2---111----------2-211-----------1210211--------
6 ··----------2-----111--------2---211----------1-21--211-----
7 ··----------2-------111-----2-----211--------1--21---211--
```

6 Automata for Systems Based on Affine Lie Algebras

The papers [18, 19] have considered another family of systems capable of supporting filtrons where the symbols are treated as types of balls, the carriers have infinite capacity, and some interaction among balls are allowed. These models are called affine Lie algebras g_n. The description of their dynamics is the following [19].

1. The valid alphabets B of symbols, and associated sequences $J = (j_d, \ldots, j_1)$ are assumed as shown in Table 2.

Table 2. Alphabets B and orders of symbols for g_n systems.

g_n	Set B	Sequence $J = (j_d, \ldots, j_1)$	d	
$A_n^{(1)}$	$\{1, 2, \ldots, n+1\}$	$(2, 3, \ldots, n+1)$	n	$n > 0$
$A_{2n-1}^{(1)}$	$\{1, 2, \ldots, n, -n, \ldots, -2, -1\}$	$(2, 3, \ldots, n, -1, -n, \ldots, -3, -2)$	$2n-1$	$n > 2$
$A_{2n}^{(2)}$	$\{1, 2, \ldots, n, -n, \ldots, -2, -1, \varnothing\}$	$(2, 3, \ldots, n, -1, -n, \ldots, -3, -2, \varnothing)$	$2n$	$n > 1$
$B_n^{(1)}$	$\{1, 2, \ldots, n, 0, -n, \ldots, -2, -1\}$	$(2, 3, \ldots, n, 0, -n, -3, -2)$	$2n-1$	$n > 2$
$C_n^{(1)}$	$\{1, 2, \ldots, n, -n, \ldots, -2, -1\}$	$(2, 3, \ldots, n, -1, -n, \ldots -3, -2, -1)$	$2n$	$n > 1$
$D_n^{(1)}$	$\{1, 2, \ldots, n, -n, \ldots, -2, -1\}$	$(2, 3, \ldots, n, -n, \ldots, -3, -2)$	$2n-2$	$n > 3$
$D_{n+1}^{(2)}$	$\{1, 2, \ldots, n, 0, -n, \ldots, -2, -1, \varnothing\}$	$(2, 3, \ldots, n, 0, -n, \ldots, -3, -2, \varnothing)$	$2n$	$n > 1$

2. Symbol 1 is a basic (quiescent) element, and –1 denotes a neutral bound state thus represents merged (particle-antiparticle) pairs (j, j *), or two symbols in a box, such that:

$$j^* = \begin{cases} \boxed{\ j\ } & \text{if } j \in \{0, 1\} \\ \boxed{j^*} & \text{if } j \in J \setminus \{0, 1\} \end{cases} \tag{7}$$

Especially 0 and $\varnothing$ are anti-particles and can form a bound state with one another.

3. The operator K_j is given by:

– Replace each –1 by a pair (j, j *) within a box.
– Move the leftmost j to the nearest right box that is empty or with a single j *.
– Repeat above until all of j's are moved once.
– Replace all pairs (j, j *) by –1.

4. Evolution of a string $b = (\ldots, b_i, \ldots)$, where $b_i \in B$, and $b_i = 1$ for $|i| \gg 1$, is performed by the operation $T = K_{j_d} \ldots K_{j_2} K_{j_1}$ where

$T(b) = K_{j_d} (\ldots (K_{j_2} (K_{j_1} (b)))\ldots)$.

Let us define the automata $M(g_n) = (S, \Sigma, \Omega, \delta, \beta, s_0)$. Input and output alphabets are $\Sigma = \Omega = B$. The states $S = \{(c_1, c_2, \ldots, c_d)\}$ are represented by d-tuples of counter states, $c_j \in \mathbf{Z}_{\geq 0}$. The counters, arranged in a pipeline shown in Figure 7 (*a*), update their contents and workout the outputs according to the state diagrams shown in Figures 7 (*b*), (*c*) and (*d*). Note that all other cases, which are not shown on diagrams, are such that the counter state c_j rests not changed and the output ω_j equals input σ_j. The initial state s_0 of automaton $M(g_n)$ represents zero counts on all d positions.

Proposition 8. The automata $M(g_n)$ are equivalent to the algebras g_n.

Proof. Operation T represents the motion of balls (as in TS model), thus one has to use the counters, and d counters are necessary because there are d types of balls. The pair-annihilation and creation has to be possible through a neutral bound state, and even the order of moving symbols has to be respected. This means that the counters must pass one-way information immediately at any distance, thus Mealy type automata are used. This implies a pipeline of counters. The operation of counters follows immediately from the map L_j given in [19].

Such a memory structure could also be treated as the list (sorted in a way) with some associated processing mechanism (or an algebra of sorting process) to assure the annihilation/creation requirements.

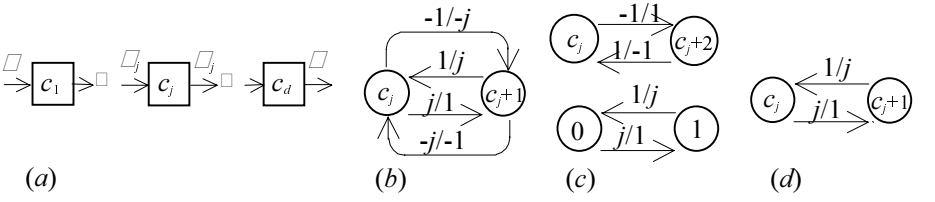


Fig. 7. (a) Pipeline of counters. (b) Case $j \notin \{0, \varnothing, 1, -1\}$. (c) Case $j \in \{0, \varnothing\}$. (d) Case $j = -1$.

Example 2. The operation of pipeline of automata $M(A_4^{(2)})$ which is equivalent to the algebra $g_n = A_4^{(2)}$ is illustrated in Table 3 with the alphabet $B = \{1, 2, -2, -1, \varnothing\}$. The states of counters are shown in boxes, and underlined symbols denote minus signs.

Table 3. Conversion of configuration ...1$\underline{1}$2$\varnothing$2$\varnothing\underline{2}$11111... onto ...111$\varnothing$1$\varnothing\underline{2}$1$\underline{1}$11... .

	σ		ω/σ		ω/σ		ω/σ		ω
τ	$\rightarrow$	$c_{\varnothing}$	$\rightarrow$	$c_{\underline{2}}$	$\rightarrow$	$c_{\underline{1}}$	$\rightarrow$	c_2	$\rightarrow$
0	$\underline{1}$	0	1	0	1	0	1	0	1
1	$\underline{2}$	2	$\underline{2}$	0	1	0	1	0	1
2	$\varnothing$	2	$\varnothing$	1	$\varnothing$	0	$\varnothing$	0	$\varnothing$
3	2	2	2	1	$\underline{1}$	0	1	0	1
4	$\varnothing$	2	$\varnothing$	0	$\varnothing$	1	$\varnothing$	0	$\varnothing$
5	$\underline{2}$	2	$\underline{2}$	0	1	1	$\underline{1}$	0	$\underline{2}$
6	1	2	$\underline{1}$	1	2	0	2	1	1
7	1	0	1	2	$\underline{2}$	0	$\underline{2}$	2	$\underline{1}$
8	1	0	1	1	$\underline{2}$	0	$\underline{2}$	1	$\underline{1}$
9	1	0	1	0	1	0	1	0	1

7 Filtron Phenomena

The interactions of filtrons exhibit very particular and rich collection of phenomena. They are especially important from the point of view of nonlinear (wave) phenomena and wave computations [22, 23, 42, 43, 44]. Let us mention only some striking, and interesting events. The images of these phenomena can be found in previous papers treating filtrons. Among specific images demonstrating the behavior of filtrons there are the following.

- Multifiltron collisions. An example is shown in Figure 5. Note that in general, to classify the collision to be a solitonic one, it is required (among others) that a nondemolition collision should occur for arbitrary number of objects [26].
- Annihilation, shown for the first time in [33], and in [34], is rather a rare event. An interesting question, under what conditions for automata and for initial configurations the annihilation occurs, requires separate studies.
- Fusion of filtrons [33, 34], an event that joins colliding objects into greater entities. Can play some role in building higher-level functional blocks when arranging computations based on filtrons' interactions.
- Calm filtrons, introduced in [37], interact hardly, because their edges are straight, and they move in parallel. Nevertheless, their internal behavior and huge periods are interesting (e.g. in cryptography).
- Vibrating objects (presented at ICCS2002, Nashua, NH) are distinguished by their rapidly changing edge delays (they move on and back within their orbit).
- Breathers [33, 34, 41] belong to a class of complex objects, and seem important because they resemble bound states or orbiting entities.
- Trapped bouncing (TB) filtrons are such that among three objects, the middle filtron is trapped, and bounces between the two others. Also, the colliders can be trapped. These phenomena have been shown for the first time in [35, 36]. What is especially interesting is that TB filtrons can collide solitonically as whole entities. This makes them good candidates for "soft" resonant cavities, not disturbing one another, or even for "memory containers" in solitonic computations.
- Diffracting-like filtrons were shown in [6, 36]. They seem to mimic immediately the basic phenomenon of propagation.
- Attracting and repelling forces. Typically, colliding filtrons change their positions (undergo translational and orbital shifts) in such a way that it can be interpreted as an evidence of attracting forces. In rare cases the filtrons repel [36, 41]. This phenomenon was shown for the first time in [24].
- Quasi-filtrons were shown in [34, 36, 38]. These are objects which can rest localized for a long time, after which they seem to transform or disintegrate "spontaneously". They demonstrate an ability to collide solitonically.

8 Final Remarks

One of the most important features of CAs is that this model explicitly considers the basic impact of fundamental physical laws. Here, we have shown that all known models supporting discrete soliton-like structures can be described by IAMs performed by means of properly chosen automata. Thus, also linearly extended SP media, represented by automata, are closer to basic physical phenomena—automata computations can mimic some of them. This also suggests that nonlinear wave phenomena, especially soliton-like coherent structures, have their fundamental counterparts within the computational models.

This was shown by treating the automata as media (described by excitation, operations, and extinction), and by identifying the types of memory that are applied by basic automata representing the mechanisms capable of supporting discrete coherent structures.

We have distinguished the following memory structures of automata: shift registers associated with FCA windows, circular memory applied by FA family (cycles of processing), finite and infinite counters (TS based models), finite or infinite lists, FIFO and stacks (BmBS-α systems), sorted bounded lists (simple crystal systems B and C), bounded counters of BBSC model, and some combinations of counters (called pipelines of counters, or converted lists of affine Lie algebras).

The definition of filtrons of automata is based on the concept of M-segments. One can easily enlarge this concept, and assume that the beginning and/or end of an M-segment is associated with some specific subsets of automaton states. Especially, such M-segments can be used together with the de Buijn graphs of CA rules, and describe a wide class of CA particles [41].

Let us emphasize also that in related papers some analytical tools have been developed to describe and analyze the filtrons. These tools are based on the concept of so-called ring computation, shown in [33, 34, 38]. Especially, the existence of filtrons for some their (edge) description and for a given automaton can be anticipated using that technique.

To close the remarks: we see the basic potential applications of filtrons in arranging pipelined computations in carefully chosen automaton media.

References

1. Ablowitz, M. J., Keiser, J. M., Takhtajan, L. A.: A class of stable multistate time-reversible cellular automata with rich particle content. *Physical Review* A. **44** no. 10 (1991) 6909-6912.
2. Adamatzky, A. (ed.). *Collision-Based Computing*. Springer-Verlag, London (2002).
3. Atrubin, A. J.: An iterative one-dimensional real-time multiplier. *IEEE Trans. on El. Computers*. **EC-14** (1965) 394-399.
4. Boccara, N., Nasser, J., Roger, M.: Particle-like structures and their interactions in spatio-temporal patterns generated by one-dimensional deterministic cellular-automaton rules. *Physical Review* A. **44** no. 2 (1991) 866-875.
5. Barrett, C. L., Reidys, C. M.: Elements of a theory of computer simulation I: sequential CA over random graphs. *Applied Mathematics and Computation*. **98** (1999) 241-259.
6. Bruschi, M., Santini, P. M., Ragnisco, O.: Integrable cellular automata. *Physics Letters* A. **169** (1992) 151-160.
7. Buning, H. K., Priese, L.: Universal asynchronous iterative arrays of Mealy automata. *Acta Informatica*. **13** (1980) 269-285.
8. Delorme, M., Mazoyer, J. (eds.): *Cellular Automata. A Parallel Model*. Kluwer (1999).
9. Deqin, Y., Quinghu, H.: The enumeration of preimages and Gardens-of-Eden in sequential cellular automata. *Complex Systems*. **12** no. 1 (2000) 83-91.
10. Fokas, A. S., Madala, H. R.: Emerging patterns and computational design of filter cellular automata. *Complex Systems*. **7** (1993) 389-413.
11. Fokas, A. S., Papadopoulu, E. P., Saridakis, Y. G., Ablowitz, M. J.: Interaction of simple particles in soliton cellular automata. *Studies in Appl. Mathematics*. **81** (1989) 153-180.
12. Fokas, A. S., Papadopoulou, E. P., Saridakis, Y. G.: Particles in soliton cellular automata. *Complex Systems*. **3** (1989) 615-633.
13. Fokas, A. S., Papadopoulou, E. P., Saridakis, Y. G.: Soliton cellular automata. Physica D. **41** (1990) 297-321.
14. Fokas, A. S., Papadopoulou, E. P., Saridakis, Y. G.: Coherent structures in cellular automata. *Physics Letters A*. **147** no. 7 (1990) 369-379.

15. Fuchssteiner, B.: Filter automata admitting oscillating carrier waves. *Applied Mathematics Letters*. **4** no. 6 (1991) 23-26.
16. Fukuda, K., Okado, M., Yamada, Y.: Energy functions in box ball systems. *International Journal of Modern Physics* A. **15** no. 9 (2000) 1379-1392.
17. Goldberg, A. H.: Parity filter automata. *Complex Systems*. **2** (1988) 91-141.
18. Hatayama, G., Kuniba, A., Okado, M., Takagi, T.: Combinatorial *R* matrices for a family of crystals: $C_n^{(1)}$ and $A_{2n-1}^{(2)}$ cases. *Progress in Mathematics* Vol. 191, Physical Combinatorics. Birkhauser Boston (2000) 105-139.
19. Hatayama, G., Kuniba, A. Takagi, T.: Simple algorithm for factorized dynamics of the g_n-automaton. *J. of Physics A: Mathematical and General*. **34** (2001) 10697-10705.
20. Hatayama, G., Hikami, K., Inoue, R. Kuniba, A., Takagi, T., Tohikiro, T.: The $A_M^{(1)}$ automata related to crystals of symmetric tensors. *J. Math. Phys*. **42** (2001) 274-308.
21. Hikami, K., Inoue, R.: Supersymmetric extension of the integrable box-ball system. *Journal of Physics A: Mathematical and General*. **33** no. 29 (2000) 4081-4094.
22. Jakubowski, M. H., Steiglitz, K., Squier, R. K.: When can solitons compute? *Complex Systems*. **10** (1996) 1-21.
23. Jakubowski, M. H., Steiglitz, K., Squier, R. K.: Information transfer between solitary waves in saturable Schrödinger equation. *Physical Review* E. **56** no. 6 (1997) 7267-7272.
24. Jiang, Z.: An energy-conserved solitonic cellular automaton. *Journal of Physics A*: *Mathematical and General*. **25** no. 11 (1992) 3369-3381.
25. Lindgren, K., Nordal, M. G.: Universal computation in simple one-dimensional cellular automata. *Complex Systems*. **4** (1990) 299-318.
26. Newell, A.: *Solitons in Mathematics and Physics*. Soc. for Industrial and App. Math., Philadelphia (1985).
27. Papatheodorou, T. S., Ablowitz, M. J., Saridakis, Y. G.: A rule for fast computation and analysis of soliton automata. *Studies in Applied Mathematics*. **79** (1988) 173-184.
28. Papatheodorou, T. S., Fokas, A. S.: Evolution theory, periodic particles and solitons in cellular automata. *Studies in Applied Mathematics*. **80** (1989) 165-182.
29. Papatheodorou, T. S., Tsantanis, N. B.: Fast soliton automata. In: "*Optimal Algorithms*", H. Djidjev (ed.) LNCS vol. 401. Springer-Verlag, Berlin (1989) 41-47.
30. Park, J. K., Steiglitz, K., Thurston, W. P.: Soliton-like behavior in automata. *Physica* D. **19** (1986) 423-432.
31. Paun, G.: On the iteration of gsm mappings. *Revue Roumaine de Mathematiques pures et Appliquees*. **23** (1978) 921-937.
32. Paun, G.: On controlled iterated gsm mappings and related operations. *Revue Roumaine de Mathematiques pures et Appliquees*. **25** (1980) 139-145.
33. Siwak, P.: Introduction to filter automata theory. *Studia z Automatyki*, **18** (1993) 87-110.
34. Siwak, P.: Filtrons and their associated ring computations. *International Journal of General Systems*, **27** nos. 1-3 (1998) 181-229.
35. Siwak, P.: On automata of some discrete recursive filters that support the filtrons. Proceedings of the Fifth International Symposium *MMAR'98*, 25-29 August 1998, Miedzyzdroje, Poland. Eds. S. Domek, R. Kaszynski, L. Tarasiejski. Printed by Wydawnictwo Politechniki Szczecinskiej. Vol. 3 (1998) 1069-1074.
36. Siwak, P.: Iterons, fractals and computations of automata. In: D. M. Dubois (ed.), *CASYS'98 - Second International Conference. Liege, Belgium, August 1998*. AIP Conference Proceedings **465**. Woodbury, New York (1999) 367-394.
37. Siwak, P.: On automaton media supporting the filtrons. In: D. M. Dubois (ed.), *CASYS'99 - Third International Conference. Liege, Belgium, August 1999*. AIP Conference Proceedings **517**. Woodbury, New York (2000) 552-573.
38. Siwak, P.: Soliton-like dynamics of filtrons of cyclic automata, *Inverse Problems*. **17** no. 4 (2001) 897-918.
39. Siwak, P.: Automata and filtrons of box-ball and crystal systems. *International Journal of Computing Anticipatory Systems*. **8** (2001) 386-401.

40. Siwak, P.: Anticipating the filtrons of automata by complex discrete systems analysis. In: D. M. Dubois (ed.), *CASYS'2001 - Fifth International Conference on Computing Anticipatory Systems*., HEC Liege, Belgium, August 13-18, 2001. AIP Conference Proceedings **627**. Woodbury, New York (2002).
41. Siwak, P.: Iterons of automata. In: *Collision-Based Computing*. A. Adamatzky (ed.), Springer-Verlag, London (2002) 299-354.
42. Steiglitz, K., Kamal, I., Watson, A.: Embedding computation in one-dimensional automata by phase coding solitons. *IEEE Transactions on Computers*. **C-37** (1988) 138-145.
43. Steiglitz, K.: Time-gated Manakov spatial solitons are computationally universal. *Physical Review E*. **63**, 016608 (21 December 2000).
44. Steiglitz, K.: Multistable collision cycles of Manakov spatial solitons. *Physical Review E*. **63**, 04407 (26 March 2001).
45. Takahashi, D.: On a fully discrete soliton system. *Proceedings* NEEDS'91, Baia Verde, Gallipoli, Italy, 19-29 June 1991. Eds. M. Boiti, L. Martina, F. Pempinelli. World Scientific, Singapore (1991) 245-249.
46. Takahashi, D.: On some soliton systems defined by using boxes and balls. *International Symposium on Nonlinear Theory and Its Applications* (NOLTA'93), Hawaii, USA, December 5-10. (1993) 555-558.
47. Takahashi, D., Matsukidaira, J.: On discrete soliton equations related to cellular automata. *Physics Letters* A. **209** (1995) 184-188.
48. Takahashi, D., Matsukidaira, J.: Box and ball system with carrier and ultradiscrete modified KdV equation. *Journal of Physics A*: *Mathematical and General*. **30**, no. 21 (1997) L733-L739.
49. Takahashi, D., Satsuma, J.: A soliton cellular automaton. *Journal of the Physical Society of Japan*. **59**, no. 10 (1990) 3514-3519.
50. Takahashi, H.: Information transmission in one-dimensional cellular space and the maximum invariant set. *Information and Control*. **33** (1977) 35-55.
51. Tokihiro, T., Nagai, A., Satsuma, J.: Proof of solitonical nature of box and ball systems by means of inverse ultra-discretization. *Inverse Problems*. **15** (1999) 1639-1662.
52. Tokihiro, T., Takahashi, D., Matsukidaira, J.: Box and ball systems as a realization of ultradiscrete nonautonomous KP equation. *Journal of Physics* A: *Mathematical and General*. **33**, no. 3, (2000) 607-619.
53. Torii, M., Takahashi, D., Satsuma J.: Combinatorial representation of invariants of a soliton cellular automaton. *Physica* D 92. no. 3-4 (1996) 209-220.

A Man and His Computer: An Issue of Adaptive Fitness and Personal Satisfaction

Tommaso Toffoli

Boston University, MA 08544, USA
tt@bu.edu

Abstract. Today man is living in a natural and social ecology extraordinarily different from that in which he evolved. Computers and communication networks have become an integral part of our world's texture; in particular, they provide the "nervous system" for a variety of superorganisms among which and *within which* we live.

Paradoxically, even though computers were introduced by man, the technological, commercial, and social superhuman "animals", thanks to their greater evolutionary plasticity, have been more ready than the human *individual* to take advantage of the computer's capabilities. Though this new environment provides us with a rich assortment of *external* information services and appliances, so far we have not gotten from computers much help in the way of extending our *very selves*—nothing comparable, for example, with the deep, intimate empowerment given by *literacy*.

We argue that an extension of our *personal capabilities* and an attendant enlargement of our own private *information space* are possible. The computer, in one of its many possible impersonations, will provide the additional processing power needed to support these extensions. But material instruments will have to be matched, on the human individual's side, by the acquisition of new competencies. Hardware is not really an issue: what is needed is the crafting of a new *culture* (in the standard sense of 'and integrated set of tools, skills, and traditions'). Thus, we are not thinking as much of a "bionic prothesis" as of a *computational literacy* culture as naturally integrated with one's person as other acquired cultural habits such as living in a home, playing a violin, or reading a book.

The design of a culture is no doubt a delicate engineering task; one that involves both humans and computers will be doubly demanding. In this paper we shall examine and try to arrange some of the pieces of the puzzle: What is desirable? What is possible? Can we identify an evolutionary stable strategy? What are the scientific issues and the technical problems involved? What do we know already? What do have to study next? Who shall do what? Who shall pay for it? Who, if any, will be threatened by it?

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 86–99, 2002.

1 Introduction

NOTE. The title of this paper is a paraphrase, of course, of Thomas Mann's delightful story "A man and his dog" [10].[1] Given the role that I anticipate for language in the interchange between human and computer (cf. §3), and given women's greater facility, on the whole, for language handling, this paper might as well have been "A woman and her computer."

I imagine a conference on non-conventional computing to be in certain respects akin to a science-fiction writers's convention. Everybody will try to impress everyone else with the most imaginative and outlandish tale. Yet, it were a poor science-fiction story that merely consisted of a fancy adventure in an exotic setting. Beside exercising the writer's imagination, a good story must stimulate the *reader*'s—for instance, it may help one preview a technologically or socially relevant issue, give concrete embodiment to a conceptual paradox, or frame an ethical dilemma.

Since I've spent a good part of my life studying fundamental connections between physics and computation, a natural candidate for this talk might have been one more clever way to make nature compute "without even knowing". We've all seen how one can extract useful computational work from water jets [15], from a DNA soup [1], from brush fires [19], even, according to John Reif, from moving a grand piano through narrow corridors [3]. We have realized that, *pace* Thomas Aquinas and the hierarchy of *ad hoc* souls he postulated to account for progressively higher capabilities in the *scala naturae* (cf. [18]), computation-universal hardware is *dirt cheap*. So cheap (a few flip-flops and a tape recorder [12], or practically any kind of excitable medium or board game like the game of "life" [2]) that Norman Margolus recommended not wasting the term 'computer' on anything that is not a *universal* computer to begin with [11]. In matters computational it is not computational-universality (which may be taken for granted) but *practicality* that will "tell the men from the boys".

So I entertained and then abandoned, as not being sufficiently practical, a plan to compute by means of meandering rivers (which can obviously process information: they cut new channels, merge and split, leave behind transient oxbow lakes, etc.).

Next, I drew up the plot for what would indeed have made a good science-fiction story. There is this civilization who owns the most advanced computers. Their machines can fit—not in a nutshell, not on the head of a pin, not inside an ammonia molecule—they can fit in the *nucleus* of an atom!—their logic gates are 10^{-18} meters across and operate in 10^{-24} second. This civilization is threatened by vicious space invaders. Fortunately, their wonderful computers are just what

[1] Charges of 'sexism' or lack of 'political correctness' would be a-historical, since at that time (1918) those memes hadn't emerged yet. At any rate, I'm aware that today, in the West at least, most dogs are purchased by women. In this regard, cf. another delightful booklet which also paraphrases Thomas Mann's title, namely *Dogs and Their Women*.

is needed to wage an effective defence; unfortunately, even though they have excellent software, trained information technologists, etc., and even though their hardware is always kept in perfect working conditions, yet they are helpless in the face of the alien attack: their computers cannot be run for any length of time because *they lack batteries*!

You may think this a contrived predicament. Yet, it is conceivably the dilemma we would find ourselves in if we wanted to exploit interactions of subnuclear particles—quarks, gluons, and all that—as an "obvious" approach to denser and faster computers. The particles made up of those, such as protons, electrons, and other traditional "elementary particles"—are stable because matter on that scale is already near thermal equilibrium: "all of the fast things have already happened" (as Feynman would say) in perhaps the first three minutes of the universe. Already by then, the batteries for those advanced computers were (almost) all gone. Sorry!

With that in mind, the lack of popular enthusiasm for the Superconducting Super Collider is not surprising. Once the laws of a certain kind of interaction are known, setting up a controlled sequence of such interactions is no longer a scientific experiment; it becomes a deliberate *computation* [17]! High-energy particle physics is thus a preliminary form of computer design (cf. [20]). But what's the point of spending enormous resources to design computers that then, because of the difficulty in finding fresh batteries, we have little hope of running?

In the end, I resolved to talk about a more realistic, more pressing, and in the end much more humanly relevant endeavor—something I've been thinking about for a while, beginning to involve others with, and slowly trying to turn into a concrete initiative. In brief, the idea is to develop a *natural* way to extend, by a combination of technological resources and cultural means, our own—that is, the *ordinary human individual's*—natural, tacit, internal computational capabilities. This will certainly be a new, nonconventional way to compute, just as literacy was the introduction of nonconventional—but, as it turned out, extremely effective and productive—ways to remember and communicate.

Instead of asking, as many of you will in this conference, "What makes a piece of hardware a computer?" I'll ask, "How can we turn computers that we already know how to design (and can program at will) into a useful complement to a "legacy" computer we care *a lot* about but which we cannot *redesign* (though we can to a certain extent *teach*)—namely, *ourselves*?

Much emphasis is being placed today on *human* cognitive sciences. To turn the computer into a natural, effective, and faithful extension of ourselves we'll have to develop a complementary discipline, namely, some sort of cognitive science of *computers*.

2 More Mileage for a Legacy Architecture

Our goal, in brief, is to develop a *culture*—a complex of tools, skills, and traditions—that will make it possible for *normal* individuals

- To use the computer as an unobtrusive extension of their own capabilities; and
- To manage and enjoy an extended *personal information space*—a "knowledge home," as it were [21], as an informatic counterpart to their physical home.

I'll readily admit that two objections, quite antithethical to one another, are commonly raised to such a proposal.

- The first says, essentially, Haven't we already achieved those milestones? What about Windows, the Internet, Google, and all that? Today, even little children can effortlessly point-and-click their way through the e-world.
- The other says, How can you even imagine that the great bulk of people will acquire the motivation, the discipline, and the technical skills needed to manage for themselves a complex informatic world? Let a priesthood of dedicated specialists can design and maintain information appliances for the masses!

I'm persuaded that both of these objections are to some extent *fraudulent*, in the same sense that most advertising is (cf. [7]). However, I'll not try to convince you of that—this is a scientific, not a moral, paper. I'll limit myself to arguing, later on, that both objections are factually *wrong*. But first let's get down to some business.

We must presume man to be quasi-optimally adapted to the natural and social environment in which he evolved—conceivably open brushland and small, loosely-coupled nomadic tribes. In a geological instant, however, the former has been replaced by a thoroughly artificial (i.e., man-fabricated and -mantained) world, and the latter has given way to an assortment of intercompenetrating "super-organisms" (corporations, institutions, territorial entities, political bodies, commercial networks). In modern western society, it has become virtually impossible for the human individual to find a habitable space and a way to make a living other than as a *job holder*, i.e., a recognized, specialized "cell' of one of those bodies [16].

The continuing viability of man in such a radically different ecology has made it obvious that man is essentially different from other animals in at least one respect: man is a thoroughly *general-purpose* animal. What constitutes "human nature" must be not so much a specific adaptive mandate, like that of most other species, as something akin to the hardware, firmware, and operating system of a general-purpose computer: well-characterized and vibrantly full of potentialities, for sure, but not really ready to go anywhere until a further level of software—in the case of man, a *role in society*—has been "loaded". Think, for an analogy, of our extraordinary innate capacities for language—the language "organ" [14]: unless a particular language is acquired at an early age they fail to realize themselves, leaving the whole person stunted.

At the same time as we have and instinct for finding and developing a role in society, so we have an instinct for maintaining, exercising, and upgrading our "operating system"—our generic self. We develop a private life and an "interior"

life, we cultivate hobbies and expand interests, we read and listen to music, we turn our home into a general-purpose extension of our person and into a support system for our activities. When we do all this, our instinct rewards us with *satisfaction*.

In the rest of this paper we shall discuss the visible interface between human and computer (§3); some of the structure and "intelligence" that will have to be present underneath (§4); and finally touch on the issue of cultural dissemination of the instrument we propose.

3 A Verbing Language

In the partnership we envisage between human individual and computer, the latter is better viewed as a pragmatic and unpretentious extension of the capabilities of the former, much like a piano to the pianist or a dog to the hunter—rather than an external "artificial intelligence" genie. For the dialogue that will take place between a human individual and his/her "extension" we envisage a **verbing language**. Functionally, this will be somewhat analogous to a *scripting* language: interactive, general-purpose, flexible and robust rather than efficient at any cost, suitable to act as a conversational shell for more specialized packages. However, the verbing language will be designed from the start as a *spoken* medium, capable of maintaining and referring to a substantial implicit "world of discourse;" thus, in terms of internal structure, it will resemble more a remorselessly stripped-down version of *natural* language.

A more formal, text-oriented dialect of the verbing language will be used for batch-mode directives and other noninteractive tasks. Though the oral mode of the verbing language will have conceptual primacy both in its design and its teaching, the written mode will be cultivated (as it happens with English) as the instrument of choice for more deliberate or more technical information-structuring activities.

The term 'verbing' (short for 'verbalizing') suggests an analogy with scripting languages, but reveals a different emphasis and scope. As we said, the verbing language is conceived from the beginning as a *conversational language*, and will be used by the computer as well to talk back to the human. In this interactive mode, the language will heavily rely on semantics. Within the same language, however, one will shift into a more formal "scripting dialect" for one-way, written, batch-mode communication, which has substantially different requirements.

Though the verbal, interactive component of the verbing language retains methodological priority, the pedagogical value of the scripting mode cannot be overemphasized (cf. [13]). A script captures the results of a mental effort so that we don't have to repeat that effort every time we want a complex action done; at the same time, unlike a keyboard macro or a recording, a script is open for inspection, recall, and modification. Unlike a sequence of point-and-click incantations, a script has a syntactical and semantic structure similar to natural language. (Scoping, data-typing, and dependencies are appropriate for

an interpretive language: temporary structures are kept simple, are built as the need arises, and are discarded after use.) The moment a sequence of commands becomes routine, it is canned, given a name, and becomes just one new command; use of modifiers (do this *in an hour*) or iterators (do this *every weekday*) greatly enlarges the scope of a base command form.

Scripts are a vehicle eminently suited for *cultural transmission*, as they provide a structured way not only to specify an action but also to *document* it, i.e., reveal the aspects of *intention* and *design* behind the action. Scripts permit a short design/test/modify cycle, and are much more tolerant of concatenation, deletion, mutation and recombination than conventional programs.[2] Moreover, a scripting language lends itself well to a gradual transition from keyboard/mouse/screen input/output to *verbal dialog*, which we expect will become more and more common in the future in dealing with computers.

Let us imagine a time when the envisaged culture will have taken hold. There are me and this other entity that I'm talking to, which I should rightly call "a piece of my mind" but I will call, for brevity, *my computer*. This is not, let us be clear, my laptop, or a box that sits on my desk, or a cellular-phone terminal to a supercomputer network—though on occasion it may appear to be any of those things, or, in a pinch, may be impersonated by *your* desktop computer, for one. Neither is it some distinguished part of the entire complex of hardware, software, and computer-accessible information that surrounds us (which we may also call 'the computer', as when we say "The computer is changing the world").

'My computer' is something different—it is a *special incantation* that I (and possibly only I) can cast (for definiteness, you may visualize this spell as a collection of programming-language scripts and data pointers) and that will evoke, out of any reasonable hardware box, anyplace, anytime, that "piece of my mind" to which I refer above. The essence of 'my computer' lies, in sum, in my having a blueprint for it, knowing how to evoke the genie described by this blueprint, and, most important of all, knowing how to deal with this genie when it materializes. More on this later on.

And why did I say 'I'm *talking*' to my computer, even though most of the time I'd probably by *typing* to him?[3] Actually, most of the time that I casually claim to be "communicating with my computer" I may well be *editing* a piece of text or a script in a stand-alone mode (say, on my laptop on an airplane); or I may be penciling a note on a bit of paper while lying on the beach; or even hashing bits of ideas in my head while swimming. What's going on here? Who am I really talking to—or whom am I trying to fool?

[2] In the our verbing/scripting/programming distinction, a *program*, in the usual sense of the term, is the software counterpart of a piece of complex, high-precision machinery whose demands for compactness, efficiency, and reliability justify a large technical investment and long design latency. In the same machine-shop metaphor, a *script* is more like a prototype, and a *verbing exchange* is more like a use-once jig.

[3] Cf. Footnote 1.

Well, imagine that my wife leaves a note on the fridge, "Tom, remember to get some milk!" Isn't she in a very real sense talking to me? It is not primarily the uttering of some sounds in the air that constitutes talking, but the fact that the note on the fridge is addressed to an actually existing person who's likely to see it at the right time; that it is certain to be comprehended and likely to be executed; that it didn't require any special coding; in sum, that it will have essentially the same effect as if my wife were directly talking to me.

But these are exactly the requirements for a meaningful conversation to be going on between me and this extra piece of my mind that I call 'my computer'. Certain details—whether the conversation is spoken or in writing, whether a particular exchange is interactive or one-way, whether transmission is immediate or buffered—do of course make a practical difference, but what's more important is that I must feel that I can talk "as it comes" and that "there is somebody there listening to me." The conversation with my computer must be qualitatively close to "thinking to myself" or to giving casual instructions to a longtime servant: that is, I must be speaking in something not unlike my natural language, with a variable amount of detail and redundancy compatible with the interactivity (or lack of) of the exchange, and the recipient must be able to understand, respond in a similar language, take action as appropriate, have a sense of when this action has achieved the intended effects, and let me know that all of that is the case.

The *verbing language*[4] I imagine we will be using to talk to 'our computers' will more closely resemble a stripped down version of natural language than a conventional programming language. Of course, like with natural language, there will be as many distinct flavors as there are speakers; all the same, like with all *cultural* artifacts, there will be much mutual comprehensibility within a community (after all, we'll be learning verbing language from people close to us) and a good deal of social reusability. Think of kitchens and recipes, how they change in going from my home to my neighbor's to a Japanese home, and yet to what extent they are still recognizably "the same".

The above scenario, which is nominally about language, is clearly as much about the *nature* of this entity called 'my computer' that I will be talking to and about *my own* nature.

To be effective, a process aimed at the acquisition of effective "computer husbandry" practices and of the attendant *computational literacy*[5] will have to be designed to leverage universal, innate human cognitive faculties. In addition,

[4] There is a multiple pun here in the word 'verbing'. It wants to recall *scripting* languages, as constrasted to more formal *programming* languages. It wants to stress that the language should be designed from the start as a spoken or "verbalizable" medium (even though one will often use a more formal, written dialect of it suitable for non-interactive directives). Finally, it wants to remind one that, notwithstanding the word 'object' in *object*-oriented programming, *verbs*—and actions—retain a primary role in language.

[5] This term was introduced by Andy diSessa [6] to distinguish a deep cultural acquisition from the skin-deep training in the use of commercial office software etc. peddled by some as "computer literacy".

this process will have to reckon with another set of cognitive aptitudes—those of the raw computing machinery that is impersonating the 'my computer' role, and those that this role itself is assigned to play.

Far from being an independent, self-contained entity with little contents of its own—like a conventional computer language—the verbing language will be a "handle" to two complex, independently-standing structures—me and my computer—both characterized by having a large amount of unique *state*—all that makes me and this extra piece of my mind different from everyone else. Then a meaningful and *efficient* dialogue must entail *four* components, that is,

- Me in its full complexity and ramifications.
- My computer in its full complexity and ramifications (secondary and tertiary storage, links to other computers and repositories, etc.).
- The "model of the world of discourse" that I entertain for the sake of a short-term conversational exchange, and which includes some model of my partner.
- The "model of the world of discourse" that my partner in the conversation likewise entertains and which includes some aspects of me.

In these circumstances, a dialogue will be like a pair of matched sliding-window *convolutions*. What I say will be convolved with the state of my computer, resulting in a new state of the world of discourse that my computer entertains (and in occasional minor modifications of its full state or "long-term memory"). At the same time, I will listen to myself and myself convolve what I say with the "brief model" of my partner that I entertain, resulting in a new state for this model. It is on the basis of this latter state that I will be able to efficiently adapt and compress what I say next, confident that it will be decompressed and understood correctly by my partner. For instance, If I had just mentioned my daughter, I'll presume that my partner will have placed a special token for her in its short-term model, and thus I may unambiguosly say 'her' rather than 'Julia' or 'my daughter' on my next utterance, thus achieving not only concision but greater *semantic clarity* as well.[6]

In a similar way, my computer will transform my state by means of his utterances and at the same time maintain an updated "short model" of me, so that when I say "Delete this file" he need only answer "Done!" to be understood.

In addition to me and my computer, there is a third complex structure that the verbing language must handle in a tacit, automatic, and therefore "intelligent" way, namely, *language itself* (human, computer, or otherwise) seen not as a historical accident, but rather as an adaptively advantageous (and in this sense "universal") solution to a common problem: that of two large and complex systems who need to exchange substantial communication and control but are constrained to using a serial, low-bandwith link.

[6] If I had mentioned *both* my daughter and my wife, then from this short-term model I will gather that just 'her' will be ambiguous without extra indicators or conventions.

Thus the verbing language must intrinsically know a lot about the primitives and the attendant combining rules of three classes of mechanisms; namely, it must handle with the competence of a native

- Primitives and rules of the human cognitive process (e.g., the ordering of spatial and temporal arrangements in terms of "here", "before", and "after"; the "who did what to whom" categorization; questions like "is this affecting that?", preliminary to any system decomposition attempts [8]).
- Primitives and rules of "language". In this general context, How do humans specifically like to say things, and why? and How do computers prefer to hear them, and why?
- Primitives and rules of computer "cognitive aptitudes". Though one can imagine all sorts of computer architectures, for the last forty years the von Neumann paradigm has managed to shrug off all challengers; meanwhile, the repertoire of recognizedly basic computational "organs and tissues" (such as procedure, tree, stack, list, interrupt, cache, pipe) has remained reasonably stable. A survey of popular UNIX shell commands or Perl modules would show that, all considered, computers insist on doing certain things in a way that is most natural to them rather than to us (cf. Lohr [9]).

Because of that all, the designers of a verbing language will have to operate under heavy constraints. Computers have cognitive aptitudes that are quite different from ours; though technical programmers and geeks can relatively easily bridge the gap, normal people won't. Thus, one task of the verbing language will be to hide the cognitive aptitude of computers under the synthetic cognitive aptitutes of 'my computer' (see next section), which will be much closer to my style of cognition. Afer all, even though both are computation-universal, computers can be reprogrammed (and their programs replicated) much more easily than people; natural language, our most powerful and natural way of access to our internal computational modes, is here to stay in spite of the vagaries and idiosyncrasies of a long, undirected adaptive evolution at the genetic level and thick layers of cultural adaptation (cf. [4,8,14]).

On the other hand, humans have extraordinary learning capacities when the right learning sets are exploited, and both the pressure of competition and the reward of personal satisfaction will induce the to put their capabilities to good use. We cannot expect to gain much from a new tool if we are not willing to train ourselves in the use of it, or from a new servant if we are not willing to understand his capabilities and limitations and train him to understand and fulfill our wishes. Using 'my computer' will be a process of *mutual domestication* (cf. [23]).

Designers of the template of 'my computer' and of the accompanying verbing language we will have to judiciously evaluate how people's and computers' "anatomies" differ, and what riding styles can be introduced or paddings devised to attain a comfortable fit.[7]

[7] Incidentally, interfaces like the saddle, the stirrup, and the horse collar had military and economic consequence utterly incommensurable with the humbleness of the inventions themselves.

4 Under the Hood

The term "computer language" is fraught with the same kind of deceiving ambiguity as "speech recognition". A speech recognition program will take your dictation and possibly even correct some of your mistakes; it may be doing a good and useful job, but in no way does it understand what you are saying. It knows about human speech statistics, perhaps including your mannerisms and idiosyncrasies, but has no "model of the world" to constantly compare your speech stream with.

In a similar way, one may think that a computer language is for "talking with a computer"; but really there is "no one there" to talk *with*, no equivalent of a person, no permanent world or model of the world underneath except, just one level down, the raw resources of the computer; likewise, there is no continuity of discourse from one program to the next. In fact, the nature of a computer language is better understood when one one examines the compiler (or the interpreter) that supports and gives life to it. It then appears that a computer language is more like a *power screwdriver*, which lightens and speeds up the chore of assembling a complex structure but, in the end, leaves all the responsibility to design the structure to us.

The verbing language imagined in the previous section aspires to being a *real* language, i.e., a language for talking *to somebody*; somebody who is there, has permanent continuity, knows you, is recognizably the same from one day to the next and yet can learn and grow.

Then, though there still are a number of eminently linguistic issues in designing a proper interface with this entity, in the first place we must resolve what kind of entity is going to be there to do the job of an extension of ourselves.

There will, of course, have to be some intelligence in 'my computer'. But, just as a disembodied "computer language" is not really a language, so there cannot be a disembodied intelligence—intelligence about *what*? On the other hand, life is full of intelligent behavior that is amazingly competent in its realm—bacterial, immunologic, chemical [22]—without having to make recourse to the formal tools of much classical "artificial intelligence".

Besides, too much stress on making of 'my computer' an artifical intelligence project runs two grave risks. On one hand, the quest for *human-like* machine intelligence may take us on a long chase and possibly lead us nowhere soon. On the other hand, who wants it? We already know an easy way to make real people, but then either (a) they start pursuing their own goals, perhaps in competition with ours, or (b) the league for the protection of animals will prevent us from turning them into our servants and slaves. If that were not enough, *super*-human artificial intelligence is already brewing, in corporate networks, advertising combines, and, in general, in the superhuman organisms to which our society is giving birth.

A much better model for what we need in 'my computer' is given by the humble *dog* (thence our title). That is something that we evolved to be a man's (or a woman's) *unconditionally faithful, undemanding* companion: the burglar's dog

will love and protect the burglar as much as the policeman's will do the policeman's biddings. The hunter's dog extends the hunter's reach, the shepherd's dog can drive sheep toward us from afar, a watchdog lends us his ear, nose (as well as a well-organized database of thousands of past smells) and subtle pattern-recognition skills. A dog shows immense satisfaction just at being allowed to bask in his master's company. Note that the 'my' in 'my dog' is not primarily and indicator of *ownership*; a dog has to *grow* with me to really become 'my dog'—and to some extent I with him. Also note that a dog doesn't have to be *so* intelligent (how would an intelligent dog adore the most despicable master?); in fact, domestication systematically tends to lower a species' intelligence as well as its aggressivity. On the other hand, a dog has a world all of its own—of smells, sounds, places, social hierarchies, chases, habits, and memories. A dog will entertain 'doggie thoughts' and have 'doggie dreams' of smelling possums or chasing squirrels .

Let the world of interest and understanding of 'my computer', then, be not my whole world, but a limited world of files, pictures, directories, pattern matches, searches, URLs, drivers, format conversion procedures, etc. This, however, will be a *real* world to which 'my computer' will have direct access, not a vague, nominalistic "semantic network" ("`dog(CHASE,BALL)`" and all that!). When 'my computer' deletes a file, well, that was a real file and it is lost for good!

Whatever intelligence 'my computer' will have will emerge from tireless iteration and reshuffling of certain low-level primitives capable of supporting modest capabilities of discovery and invention by basically *evolutionary* mechanisms (like, after all, *all* kinds of intelligence; cf. once more [22]). Among these primitives

- A built-in *feedback loop* template for solving inverse problems by massive iteration of the corresponding direct problems. If I know how to answer the simple direct question, Does this file contain the string "quark"?, then, by using this loop I'll eventually be able to answer the indirect question, What's the longest file containing "quark"?

  Once capabilities for this kind of loop have been designed in as a basic reflex, details of the iteration order, of the closeness of an acceptable match (Does "Quark" qualify?), of timeout or retry policies, etc. can be left to be hashed out by genetic algorithms, recombination of previous runs, and, in sum, by the "try and retry and keep whatever works" approach.
- A built-in *comparison* template ("scalar product") that given any two structures will return a real number rating their similarity. Again, the adaptive development of a repertoire of features to be used in the comparison, of weighting patterns, etc., will be left to evolution.
- A built-in *tree* template, for synthesizing ever more complex structures (data, commands, production rules) from persistent reuse of a single forking construct (see analogies in natural language [8], protein synthesis, phenotipic development, artificial languages such as Lisp, etc.)

– A built-in "*pleasure center*" template, which will convert to a single currency the values of different commodities like running time, closeness of match, owner's approval, intrinsic simplicity of a solution, and so forth.

The task of *inventing* the right personality, the right role, the right "species" for 'my computer' is one deserving the best efforts of applied cognitive science, computer science, education science, and, to some extent, applied anthropology.

5 Conclusions

At the beginning of the industrial revolution, industrialists attempted at first to have *complete* control over the worker: work from early morning to night, workers' lodgings on company grounds, company stores, blue laws (no drinking on Sunday to have sober workers on Monday), etc. Workers had next to no private life (or time for it, for that matter).

Eventually, by a combination of workers' demands, trade unions, industrialists' philanthropic initiatives, government supervision, and, last but not least, a quest for more productive ways of managing the worker as an economic resource, much of private life was returned to the worker. The corporation gave up meddling with the workers' reproductive policies, their family life, their childrens' schooling, their leisure time, even their political views. The corporation still tries, by and large, to get as much as possible out of the worker (what else could one expect?). However, there is a tacit contract that—ostensibly for the good of the worker but undoubtedly also for the overall good of the corporate world—the making and raising of new workers, their day-to-day "maintenance" (bed, shower, rest and relaxation), and much of what we call "private life" even though it is indispensable to *public* life, is "outsourced" back to the worker: its details become his or her responsibility.

In a similar way, at the beginning of the "computer revolution" the superorganisms of the day—be they Internet providers, software companies, communication conglomerates, e-businesses, and what-nots—have been making a bid for total control of the informatics "customers", mainly through offers they "cannot resist": one-click shopping, browser frames that do not let go, email services where your mail remains with them so that you must continually return to their "water hole" to edit a reply, and so forth. To show just one aspect of this attempt to monopolize the user, I will quote a clear message that Michael Dertouzos, the late director of the MIT Laboratory for Computer Science, gave to DARPA in his last published statement [5]: "[T]ake, for example, the development of a radically new operating system that will increase human productivity three to one, by making machines serve people rather than the other way around. *It is against the interest of large software manufacturers to pursue such research-and-development endeavor.* [italics mine] They'd rather upgrade their systems incrementally so as to preserve a steady revenue stream from a huge base of existing custometrs. By doing so they inadvertently exacerbate the complexity

and difficulty of using these systems. A radically new approach could help all computer users in a big way. . . ."

In this context, the present initiative aims at preparing the terms and the material conditions for a broad *outsourcing contract* to humans as individuals, from societal and corporate organisms, of a substantial part of "informatics space". Resources will have to be devolved to this, just as they are being devoted to universal schooling and other forms of public education. We'll need basic research, curriculum development, creation of infrastructure, teaching aids, documentation, textbooks, and all the paraphernalia for the diffusion of literacy, this time applied to the dissemination of *computational literacy*. In a world of six or ten billion people, the extended knowledge space made available to each of us by 'my computer' will provide much needed *Lebensraum* at the only affordable cost.

References

1. Adleman, Leonard: Computing with DNA: The manipulation of DNA to solve mathematical problems is redefining what is meant by 'computation', in Scientific American **279**:2 (August 1998), 54–61.
2. Berlekamp, Elwyn, Richard Guy, and John Conway: Winning Ways for Your Mathematical Plays, 2nd edition, A. K. Peters 2001.
3. Canny. J., B. Donald, John Reif, and P. Xavier: Kinodynamic motion planning, in J. ACM **40** (1993), 1048–1066.
4. Chomsky, Noam, Language and Mind, Harcourt Brace Jovanovich 1972.
5. Dertouzos, Michael: An open letter to DARPA, in Technology Review, Oct. 2001, 50.
6. diSessa, Andrea: Changing Minds: Computers, learning, and literacy, MIT Press 2000.
7. Feynman, Richard: The Pleasure of Finding Things Out, Perseus 1999.
8. Jackendoff, Ray: Patterns in the Mind: Language and human nature, BasicBooks 1994.
9. Lohr, Steve: Go To: The programmers who created the software revolution, BasicBooks 2001.
10. Mann, Thomas: A Man and His Dog (1918), in Stories of Three Decades (H. T. Lowe–Porter ed.), The Modern Library.
11. Margolus, Norman: Physics and computation, Ph. D. Thesis, MIT Physics Dept., June 1987.
12. Minsky, Marvin: Computation: Finite and Infinite Machines, Englewood Cliffs, NJ: Prentice–Hall, 1967.
13. Ong, Walter: Orality and Literacy—The technologizing of the world, Routledge 1982 (reprinted Methuen 1988).
14. Pinker, Steven: The Language Instinct: How the mind creates language, Harper 2000.
15. Stong, C. L., and Shawn Carlson: A fluidic flip-flop, in Scientific American (May 1966), 128–135 (this is a second-hand reference).
16. Thayer, Lee: The functions of incompetence, in Vistas in Physical Reality: Papers in honor of Henry Margenau (Laszlo and Sellon ed.), Plenum Press 1976, 171–187, republished in electronic form in the Knowledge Home Library (`kh.bu.edu/khl`).

17. Toffoli, Tommaso: Physics and Computation, in Int. J. Theor. Phys. **21** (1982), 165–175.
18. Toffoli, Tommaso: Occam, Turing, von Neumann, Jaynes: How much can you get for how little? (A conceptual introduction to cellular automata), in The Interjournal (October 1994). Republished from ACRI '94: Automi Cellulari per la Ricerca e l'Industria, Rende, Italy, 29–30 September 1994.
19. Toffoli, Tommaso: Non-conventional computers, in Wiley Encyclopedia of Electrical and Computer Science (J. G. Webster ed.), Wiley & Sons, vol. 14 (1998), 455–471.
20. Toffoli, Tommaso: Symbol Super Colliders, in Collision Based Computing (Andrew Adamatzky ed.), Springer 2002.
21. Toffoli, Tommaso: A Knowledge Home: Personal knowledge structuring in a computer world, white paper, draft 5.00 (25 January 2002), `kh.bu.edu`.
22. Vertosick, Frank: The Genius Within: Discovering the intelligence of every living thing, Harcourt 2002.
23. Wilson, Peter: The Domestication of the Human Species, Yale U. Press 1988.

Exploiting the Difference in Probability Calculation between Quantum and Probabilistic Computations

Masami Amano[1], Kazuo Iwama[2], and Rudy Raymond H.P.[2]

[1] IBM Tokyo Research Lab, Tokyo, Japan
amanom@jp.ibm.com
[2] Graduate School of Informatics, Kyoto University/ERATO
{iwama, raymond}@kuis.kyoto-u.ac.jp

Abstract. The main purpose of this paper is to show that we can exploit the difference in the probability calculation between quantum and probabilistic computations to claim the difference in their space efficiencies. It is shown that, for each n, there is a finite language L which contains sentences of length up to $O(n^{c+1})$ such that: (i) There is a one-way quantum finite automaton (qfa) of $O(n^{c+4})$ states which recognizes L. (ii) However, if we try to simulate this qfa by a probabilistic finite automaton (pfa) *using the same algorithm*, then it needs $\Omega(n^{2c+4})$ states. It should be noted that we do not prove real lower bounds for pfa's but show that if pfa's and qfa's use exactly the same algorithm, then qfa's need much less states.

1 Introduction

It is a fundamental rule of quantum computation that if a state q has an amplitude of σ, then q will be observed not with probability $\|\sigma\|$ but with probability $\|\sigma\|^2$. Therefore, if one can increase the amplitude of q twice, i.e., from σ to 2σ, then the corresponding probability increases four times, i.e., $\|\sigma\|^2$ to $4\|\sigma\|^2$. In general, if the amplitude increases k times then the probability increases k^2 times. One observation of the Grover search [Gro96], which is apparently one of the most celebrated quantum algorithms, is that it takes advantage of this fact cleverly, by inventing the (efficient) quantum process whose k iterations increase the amplitude of a designated state roughly k times. As described above, this is equivalent to increasing the probability k^2 times. Thus the Grover search is faster quadratically than the classic randomized search whose k iterations can increase the probability only k times.

In this paper, we also exploit this feature, i.e., the difference in probability calculation between quantum and probabilistic computations, but from a bit different angle: Suppose that there are ten pairs of state $(p_1, q_1), \cdots, (p_{10}, q_{10})$ where, for each $1 \leq i \leq 10$, either p_i or q_i has the amplitude $1/\sqrt{10}$ (we say that p_i is ON if it has the amplitude and OFF otherwise.). We wish to know how many p_i's are ON. This can be done by "gathering" amplitudes by applying a

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 100–114, 2002.

Fourier transform from p_i's to r_i's and by observing r_{10} (see later sections for details). If all ten p_i's are ON, then the amplitude of r_{10} after Fourier transform is one and it is observed with probability one. If, for example, only three p_i's are ON, then the amplitudes of r_{10} is 3/10 and is observed with probability 9/100. In the case of probabilistic computation, we can also gather the probability of p_i's (= 1/10 for each) simply by defining (deterministic) transitions from p_i to r_{10}. If all pairs are ON, then the probability that r_{10} is observed is one again, but if only three p_i's are ON, its probability is 3/10. If the latter case (only three p_i's are ON) is associated with some erroneous situation, this probability, 3/10, is much larger than 9/100 in the quantum case. In other words, quantum computation can enjoy much smaller error-probability due to the difference in the rule of probability calculation.

The question is of course whether we can turn this feature into some concrete result or how we can translate this difference in probability into some difference in efficiency like time and space. In this paper we give an affirmative answer to this question by using quantum finite automata; we prove that there is a finite language L which contains sentences of length up to $O(n^{c+1})$ such that: (i) There is a one-way quantum finite automaton (qfa) of $O(n^{c+4})$ states which recognizes L. (ii) However, if we try to simulate this qfa by a probabilistic finite automaton (pfa) *using the same algorithm*, then it needs $\Omega(n^{2c+4})$ states. It should be noted that we do not prove real lower bounds for pfa's but show that if pfa's and qfa's use exactly the same algorithm (the only difference is the way of gathering amplitudes mentioned above), then qfa's need much less states.

Quantum finite automata have been popular in the literature since its simplicity is nice to understand merits and demerits of quantum computation [AF98,AG00,AI99,ANTV99,KW97,Nay99]. Ambainis and Freivalds [AF98] proved an exponential difference in the size of qfa's and pfa's for a one-letter language. Their technique highly depends on the *rotation of complex amplitudes*, which is exceptionally powerful for a certain situation. Nayak [Nay99] gave a negative side of qfa's by showing $L_n = \{wa|\ w \in \{a,b\}^*\ and\ |w| \leq n\}$ needs exponentially more states for qfa's than for dfa's. L_∞ is a regular set but is not recognizable by qfa's as shown by Kondacs and Watrous [KW97]. [KW97] also introduced 2-way qfa's which can recognize non-regular languages. To our best knowledge, the notion of gathering amplitudes using Fourier transform appeared in this paper for the first time and played an important role in [AI99], too.

2 Problem EQ

Suppose that Alice and Bob have n-bit numbers x and y and they wish to know whether or not $x = y$. This problem, called EQ, is one of the most famous problems for which its randomized communication complexity ($= \Theta(\log n)$) is significantly smaller than its deterministic counterpart ($= n+1$) [KN97]. In this paper, we need a little bit more accurate argument on the value of randomized (and one-way) communication complexity: Consider the following protocol M_{EQ}: (i) Alice selects a single prime p among the smallest N primes. (ii) Then she

divides x by p and sends Bob p and the residue a. (*iii*) Bob also divides his number y by p and compares his residue with a. They accept (x, y) iff those residues coincide.

It is obvious that if $x = y$ then protocol M_{EQ} accepts (x, y) with probability one. Let $E(N)$ be the maximum (error) probability that M_{EQ} accepts (x, y) even if $x \neq y$. To compute $E(N)$, we need the following lemma: (In this paper, $\log n$ always means $\log_2 n$ and $\lceil f(n) \rceil$ for a function $f(n)$ is simply written as $f(n)$.)

Lemma 1. *Suppose that $x \neq y$ and let $S(x, y)$ be a set of primes such that $x \equiv y$ mod p for all p in $S(x, y)$. Also, let s be the maximum size of such a set $S(x, y)$ for a pair of n-bit integers x and y. Then $s = \Theta(n/\log n)$.*

Proof. Let p_i be the i-th largest prime and $\pi(n)$ be the number of different primes $\leq n$. Then the prime number theorem says that $\lim_{n\to\infty} \frac{\pi(n)}{n/\log_e n} = 1$, which means that $p_{n/\log n} = \Theta(n)$. Consequently, there must be a constant c s.t. $p_1 \cdot p_2 \cdots p_{n/\log n} \cdots p_{cn/\log n} > p_{n/\log n} \cdot p_{n/\log n+1} \cdots p_{cn/\log n} > 2^n$ since $n^{n/\log n} = 2^n$. Thus an n-bit integer z has at most $cn/\log n$ different prime factors. Note that $x \equiv y$ mod a iff $|x - y| \equiv 0$ mod a. Hence, $s \leq cn/\log n$. Also it turns out by the prime number theorem that there is an n-bit integer z such that it has $c'n/\log n$ different prime factors for some constant c', which proves that $s \geq c'n/\log n$. □

In this paper, N_0 denotes this number s which is $\Theta(n/\log n)$. Then

Lemma 2. $E(N) = N_0/N$.

For example, if we use $N = n^2/\log n$ different primes in M_{EQ}, its error-rate is $1/n$.

3 Our Languages and qfa's

A one-way qfa is the following model: (*i*) Its input head always moves one position to the right each step. (*ii*) Global state transitions must be unitary. (*iii*) Its states are partitioned into *accepting*, *rejecting* and *non-halting* states. (*iv*) Observation is carried out every step, and if acceptance or rejection is observed, then the computation halts. Otherwise, computation continues after proportionally distributing the amplitudes of accepting and rejecting states to non-halting states. We omit the details, see for example [KW97]. In this paper, we consider the following three finite languages.

$L_0(n) = \{w\sharp w^R \mid w \in \{0,1\}^n\}$,

$L_1(n) = \{w_1\sharp w_2\sharp\sharp w_3\sharp w_4\sharp \mid w_1, w_2, w_3, w_4 \in \{0,1\}^n,$
$(w_1 = w_2^R) \vee ((w_1 w_2) = (w_3 w_4)^R)\}$,

$L_2(n,k) = \{w_{11}\sharp w_{12}\sharp\sharp w_{13}\sharp w_{14}\sharp\sharp\sharp \cdots \sharp\sharp\sharp w_{i1}\sharp w_{i2}\sharp\sharp w_{i3}\sharp w_{i4}\sharp\sharp\sharp \cdots$
$\cdots \sharp\sharp\sharp w_{k1}\sharp w_{k2}\sharp\sharp w_{k3}\sharp w_{k4}\sharp \mid w_{i1}, w_{i2}, w_{i3}, w_{i4} \in \{0,1\}^n, 1 \leq i \leq k$
and $1 \leq \exists j \leq k$ s.t. $(w_{j1} = w_{j2}^R)$
$\wedge$ (for all $1 \leq i \leq j-1$, $(w_{i1} w_{i2}) = (w_{i3} w_{i4})^R$)$\}$.

In the next section, we first construct a qfa M_0^Q, which accepts each string $x \in L_0$ with probability 1 and each string $y \notin L_0$ with probability at most $\frac{1}{n}$. M_0^Q simulates the protocol M_{EQ} in the following way (see Fig.1). Given an input string $\not\in w_1 \sharp w_2 \$$ ($\not\in$ is the leftmost and \$ is the rightmost symbols), M_0^Q first splits into N different states $q_{p_1}, \cdots, q_{p_i}, \cdots, q_{p_N}$ with equal amplitudes by reading $\not\in$. Then from q_{p_i}, submachine M_{1i} starts the task for dividing integer w_1 by the i-th prime p_i. This computation ends up in some state of M_{1i} which corresponds to the residue of the division. This residue is shifted to the next submachine M_{2i}, and then M_{2i} carries out a completely opposite operation while reading w_2. If (and only if) two residues are the same, M_{2i} ends up in some specific state q_i^0. M_0^Q then applies a Fourier transform from q_i^0 to s_i for $1 \leq i \leq N$. M_0^Q thus simulates M_{EQ} by setting s_N as its only accepting state.

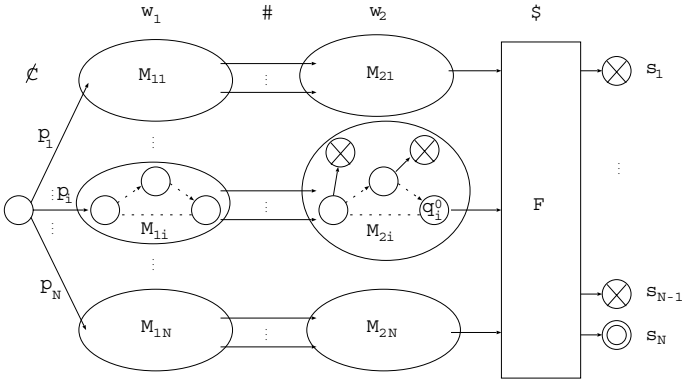


Fig. 1. Qfa M_0^Q.

For the probabilistic counterpart, pfa M_0^P, we can use exactly the same state transition, except for deterministic transitions from q_i^0 to s_N. As mentioned before we can achieve a quadratic difference in the probability of error, like $(1/n)^2$ for M_0^Q v.s. $(1/n)$ for M_0^P. It would be nice if this quadratic difference of error can be traded directly to a quadratic difference in the necessary number of primes or to a quadratic difference in the size of automata. Unfortunately that is not possible: The main reason is that we do not need such a small (like $1/n$ or $1/n^2$) error-rate but something like $1/3$ is enough by definition. Then the quadratic difference in the error is reduced to a difference between, say, $1/3$ and $1/9$, which means only a difference of the constant factor in the necessary number of primes or the necessary number of states.

There is a standard technique to overcome this difficulty, namely, the use of iteration. Consider the following string:

$$w_{11} \sharp w_{12} \sharp\sharp w_{21} \sharp w_{22} \sharp\sharp \cdots \sharp w_{n1} \sharp w_{n2}$$

where the accepting condition is that for some $1 \leq j \leq n$, $w_{j1} = w_{j2}^R$. When all pairs (w_{j1}, w_{j2}) do not satisfy this condition, the (error) probability of accepting such a string is roughly $O\left(\frac{1}{n}\right) \times n = O(1)$, which appears desirable for our purpose.

This argument does not seem to cause any difficulty for pfa's but it does for qfa's for the following reason: After checking w_{11} and w_{12}, the qfa is in a single accepting state if the condition is met, which is completely fine. However, if $w_{11} \neq w_{12}^R$ and the observation is not accepting, then there are many small amplitudes distributed to many different states. Note that we have to continue the calculation for w_{21} and w_{22} which should be started from a single state. (It may be possible to start the new computation from each non-halting state, but that will result in an exponential blow-up in the number of states, which again implies no clear separation in the size of automata.) One can see easily that we cannot use a Fourier transform this time to gather the amplitudes since there are many different patterns in the distribution of states which have a small nonzero amplitudes.

This is the reason why the next language $L_1(n)$ plays an important role. Suppose that $w_1 \neq w_2^R$. Then the resulting distribution of amplitudes is quite complicated as mentioned above. However, no matter how complicated it is, we can completely "reverse" the previous computation for $w_1 \sharp w_2$ by reading $w_3 \sharp w_4$ if $(w_1 w_2) = (w_3 w_4)^R$. This reverse computation should end up in a single state of amplitude one (actually it is a little less than one) since the original computation for $w_1 \neq w_2^R$ starts from the (single) initial state. Now one can use the iteration scheme, which naturally leads us to the third language $L_2(n, k)$.

4 Main Results

As mentioned in the previous section, we construct our qfa's and corresponding pfa's for $L_0(n)$, $L_1(n)$ and $L_2(n, n^c)$ in a step-by-step fashion. Recall that N is the number of primes used in protocol M_{EQ} and $N_0 = \Theta(n/\log n)$.

Lemma 3. *There exists a qfa M_0^Q which accepts strings in L_0 with probability one and strings not in L_0 with probability at most $\left(\frac{N_0}{N}\right)^2$. The number of states in M_0^Q is $\Theta(N^2 \log N)$.*

Proof. M_0^Q has the following states: (*i*) An initial state q_0, (*ii*) $q_{p_k,j_k,1}$ (in sub-machine M_{1i} of Fig.1), (*iii*) $q_{p_k,j_k,2}$ (in M_{2i} of Fig.1), (*iv*) $q_{p_k,j_k,rej}$ (also in M_{2i} of Fig.1), (*v*) s_l, where $1 \leq k \leq N$, $0 \leq j_k \leq p_k - 1$ and $1 \leq l \leq N$. p_k denotes the k-th largest prime > 3 (two is excluded for the reason mentioned later). s_N is the only accepting state, $q_{p_k,j_k,rej}$ and s_l $(1 \leq l \leq N-1)$ are rejecting states and all the others are non-halting states. We give a complete state transition diagram of M_0^Q in Table 1, where $V_\sigma|Q\rangle = \alpha_1|Q_1\rangle + \cdots + \alpha_i|Q_i\rangle + \cdots + \alpha_m|Q_m\rangle$ means that if M_0^Q reads symbol σ in state Q, it moves to each state Q_i with amplitude α_i $(|\alpha_1|^2 + \cdots + |\alpha_m|^2 = 1)$.

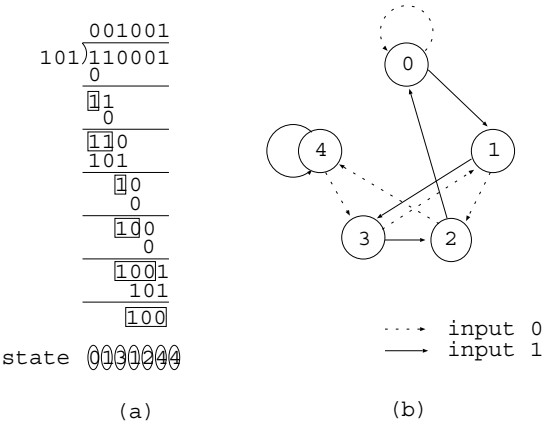


Fig. 2. Division procedure for $w_1 = 110001$ and $p_k = 5$.

When reading $\not\in$ of the input string $\not\in w_1 \sharp w_2 \$$, M_0^Q splits into N submachines (denoted by M_{1i} in Fig.1) with equal amplitudes (see transition (1) of Table 1). The k-th submachine M_{1k} computes the residue when dividing w_1 by p_k (by using transition $(2-a)$ to $(2-d)$ in Table 1). This division can be done simply by simulating the usual division procedure as shown in Fig.2 (a) and (b) for $w_1 = 110001$ and $p_2 = 101$ $(= 5)$. State j in Fig.2 (b) corresponds to $q_{p_2,j,1}$. The starting state is 0 and by reading the first symbol 1 it goes to state 1. By reading the second symbol 1, it goes to state 3 $(= 11)$. Now reading 0, it goes to state 1 since $110 \equiv 1 \bmod 101$. This continues until reading the last symbol 1 and M_0^Q ends up in state 4. It should be noted that these state transitions are reversible: For example, if the machine reaches state 2 $(= 10)$ from some state Q by reading 0, then Q must be state 1 since Q cannot be greater than 2. (Reason: If Q is greater than 2, it means that the quotient will be 1 after reading a new symbol. Since M_0^Q reads 0 as the new symbol, the least significant bit of the residue when divided by 5 must be 1, which excludes state 2 as its next state.) Hence the quotient must have been 0, and so the previous state must be 1. Note that this argument holds because we excluded two, which is the only even prime, from p_k.

Thus, if $w_1 \bmod p_k = j_k$, then M_0^Q is in superposition $\frac{1}{\sqrt{N}} \sum_{k=1}^{N} |q_{p_k,j_k,1}\rangle$ after reading w_1. Then M_0^Q reads $\sharp$ and this superposition is "shifted" to $\frac{1}{\sqrt{N}}$ $\sum_{k=1}^{N} |q_{p_k,j_k,2}\rangle$, where M_0^Q checks if $w_2^R \bmod p_k$ is also j_k by using transition $(4-a)$ to $(4-d)$ in Table 1. This job can be done by completely reversing the previous procedure of dividing w_1 by p_k. Actually, the state transitions are obtained by simply reversing the directions of previous state diagrams. Since previous transitions are reversible, new transitions are also reversible. Now one can see that the k-th submachine M_{2k} is in state $q_{p_k,0,2}$ iff the two residues are the same.

Finally by reading \$, Fourier transform is carried out only from these zero-residue states $q_{p_k,0,2}$ to s_l. From other states $q_{p_k,j,2}$ $(j \neq 0)$ M_0^Q goes to rejecting states $q_{p_k,j,rej}$. If the residues are the same in only t submachines out of the k ones, the amplitude of s_N is computed as

$$\frac{1}{N}\sum_{|t|}\sum_{l=1}^{N}\exp\left(\frac{2\pi i}{N}kl\right)|s_l\rangle = \frac{t}{N}|s_N\rangle + \frac{1}{N}\sum_{|t|}\sum_{l=1}^{N-1}\exp\left(\frac{2\pi i}{N}kl\right)|s_l\rangle,$$

namely that is equal to t/N. Thus the probability of acceptance is $\left(\frac{t}{N}\right)^2$. If the input string is in L_0, then this probability becomes 1. Otherwise, it is at most $(N_0/N)^2$ by Lemma 2. The number of states in M_0^Q is given as

$$1 + 2\sum_{k=1}^{N} p_k + \sum_{k=1}^{N}(p_k - 1) + N = 1 + 3\sum_{k=1}^{N} p_k \leq 1 + 3 \cdot N \cdot p_N = O(N^2 \log N),$$

which completes the proof. □

Table 1. State transition diagram of M_0^Q.

(1)	$V_{\not\in}\|q_0\rangle = \frac{1}{\sqrt{N}}\sum_{k=1}^{N}\|q_{p_k,0,1}\rangle$,	$(4-a)$	$V_0\|q_{p_k,j,2}\rangle = \|q_{p_k,\frac{j}{2},2}\rangle$ $(j:\ even)$,
$(2-a)$	$V_0\|q_{p_k,j,1}\rangle = \|q_{p_k,2j,1}\rangle$ $(0 \leq j < \frac{p_k}{2})$,	$(4-b)$	$V_0\|q_{p_k,j,2}\rangle = \|q_{p_k,\frac{j+p_k}{2},2}\rangle$ $(j:\ odd)$,
$(2-b)$	$V_0\|q_{p_k,j,1}\rangle = \|q_{p_k,2j-p_k,1}\rangle$ $(\frac{p_k}{2} < j < p_k)$,	$(4-c)$	$V_1\|q_{p_k,j,2}\rangle = \|q_{p_k,\frac{j-1+p_k}{2},2}\rangle$ $(j:\ even)$,
$(2-c)$	$V_1\|q_{p_k,j,1}\rangle = \|q_{p_k,2j+1,1}\rangle$ $(0 \leq j < \frac{p_k}{2} - 1)$,	$(4-d)$	$V_1\|q_{p_k,j,2}\rangle = \|q_{p_k,\frac{j-1}{2},2}\rangle$ $(j:\ odd)$,
$(2-d)$	$V_1\|q_{p_k,j,1}\rangle = \|q_{p_k,2j+1-p_k,1}\rangle$ $(\frac{p_k}{2} - 1 < j < p_k)$,	$(5-a)$	$V_\$\|q_{p_k,0,2}\rangle = \frac{1}{\sqrt{N}}\sum_{l=1}^{N}\exp\left(\frac{2\pi i}{N}kl\right)\|s_l\rangle$,
(3)	$V_\sharp\|q_{p_k,j,1}\rangle = \|q_{p_k,j,2}\rangle$,	$(5-b)$	$V_\$\|q_{p_k,j,2}\rangle = \|q_{p_k,j,rej}\rangle$ $(1 \leq j < p_k)$.

Let us consider the pfa whose state transition is exactly the same as M_0^Q of $f(N)$ states excepting that the state transitions from $q_{p_k,0,2}$ to s_l for the Fourier transform are replaced by simple (deterministic) transitions from $q_{p_k,0,2}$ to s_N. We call such a pfa *emulates* the qfa. Suppose that M^P emulates M^Q. Then the size of M^P is almost the same as that of M^Q, i.e., it is also $\Theta(f(N))$ if the latter is $f(N)$, since the Fourier transform does not make much difference in the number of states. The following lemma is easy:

Lemma 4. *Suppose that M_0^P emulates M_0^Q. Then M_0^P accepts strings in L_0 with probability one and those not in L_0 with probability N_0/N.*

Let us set, for example, $N = N_0\sqrt{n}$. Then the error-rate of M_0^Q is $(N_0/N)^2 = \frac{1}{n}$ and its size is $O(n^3/\log n)$. To achieve the same error-rate by a pfa, we have to set $N = N_0 n$, which needs $O(n^4/\log n)$ states.

Remark 1. Suppose that we have once designed a specific qfa M_0^Q (similarly for M_0^P). Then it can work for inputs of any length or it does not reject the input only by the fact that its length is not $2n+1$. The above calculation of the acceptance and rejection rates is only true when our input is restricted to strings $\subseteq \{0,1\}^n \sharp \{0,1\}^n$.

The following lemmas, Lemma 5 and 6 (see Acknowledgment), are important for the analysis of error probability of M_1^Q, a qfa which recognizes the second language $L_1(n)$. Here, $\|\psi\|$ means the norm of a vector ψ and $\|\psi\|_{acc}$ the norm of ψ after being projected onto accepting space, i.e., the accepting probability is $\|\psi\|_{acc}^2$. $\langle\psi|\phi\rangle$ denotes the inner product between ψ and ϕ.

Lemma 5. *Let ψ be a quantum state such that applying a unitary transformation U followed by a measurement to ψ causes acceptance with probability* 1*, i.e., $\|U\psi\|_{acc}^2 = 1$. If ψ can be decomposed into two orthogonal states ψ_1 and ψ_2 s.t. $\psi = \psi_1 + \psi_2$, then $\|U\psi_1\|_{acc}^2 \geq \|\psi_1\|^4$.*

Proof. Let $H = span\{\phi \mid \|U\phi\|_{acc} = 1 \text{ and } \|\phi\| = 1\}$, i.e., H is obtained by applying U^{-1} to the subspace spanned by accepting states only. The accepting probability of $U\psi_1$ is equal to the squared projection of $U\psi_1$ on the subspace spanned by accepting states. Since U is unitary and any unitary transformation preserves the inner product, it turns out that this projection is the same as the projection of ψ_1 onto H. Let $H' = span\{\psi\}$. Since $\|U\psi\|_{acc}^2 = 1$, we have $H' \subseteq H$. Therefore the projection of ψ_1 to H is at least the projection of ψ_1 to H', namely at least $\| \langle\psi_1|\psi\rangle \| = \|\psi_1\|^2$. To summarize, $\|U\psi_1\|_{acc}^2 \geq \| \langle\psi_1|\psi\rangle \|^2 = \|\psi_1\|^4$. □

Lemma 6. *Let ψ be a quantum state such that applying a unitary transformation U followed by a measurement to ψ causes acceptance with probability at most α^2, i.e., $\|U\psi\|_{acc}^2 \leq \alpha^2$. If ψ can be decomposed into two orthogonal states ψ_1 and ψ_2 s.t. $\psi = \psi_1 + \psi_2$, then*

$$\|U\psi_1\|_{acc}^2 \leq \|\psi_1\|^2 \left(\alpha\|\psi_1\| + \|\psi_2\|\right)^2.$$

Proof. Let H be the Hilbert space spanned by ψ_1 and ψ_2. Then, ψ_1 can also be written as

$$\psi_1 = \langle\psi|\psi_1\rangle \ \psi + \left\langle\bar{\psi}|\psi_1\right\rangle \ \bar{\psi},$$

where $\bar{\psi}$ is a normalized vector in H and perpendicular to ψ. Note that $\|\psi\|$ is also 1. Again $\| \langle\psi|\psi_1\rangle \| = \|\psi_1\|^2$ and from the above equation we obtain that $\langle\psi_1|\psi_1\rangle = \| \langle\psi|\psi_1\rangle \|^2 + \| \left\langle\bar{\psi}|\psi_1\right\rangle \|^2$, which implies that $\| \left\langle\bar{\psi}|\psi_1\right\rangle \|^2 = \|\psi_1\|^2(1 - \|\psi_1\|^2)$. Note that $\|\psi_1\|^2 + \|\psi_2\|^2 = 1$. Thus, $\| \left\langle\bar{\psi}|\psi_1\right\rangle \| = \|\psi_1\| \ \|\psi_2\|$. Since

$U\psi_1 = \langle\psi|\psi_1\rangle\ U\psi + \langle\bar{\psi}|\psi_1\rangle\ U\bar{\psi}$ and our observation is a simple projection, it follows by triangular inequality that

$$\begin{aligned}\|U\psi_1\|_{acc} &\leq \|\langle\psi|\psi_1\rangle\|\ \|U\psi\|_{acc} + \|\langle\bar{\psi}|\psi_1\rangle\|\ \|U\bar{\psi}\|_{acc}\\ &\leq \|\psi_1\|^2\alpha + \|\psi_1\|\ \|\psi_2\|\\ &= \|\psi_1\|\,(\alpha\|\psi_1\| + \|\psi_2\|)\,.\end{aligned}$$

This proves the lemma. □

Now we shall design a qfa M_1^Q which recognizes the second language $L_1(n)$.

Lemma 7. *There exists a qfa M_1^Q which accepts strings in L_1 with probability at least $1 - (\frac{N_0}{N_1})^2 + (\frac{N_0}{N_1})^4$ and strings not in L_1 with at most $(\frac{N_0}{N_1})^2 + (1 - (\frac{N_0}{N_1})^2)(\frac{N_0'}{N_2} + \frac{N_0}{N_1})^2$. M_1^Q has $\Theta((N_1N_2)^2 \log N_1 \cdot \log N_2)$ states. Here N_0' denotes the number s in Lemma 1 but for x and y of length $2n$.*

Proof. Again a complete state transition diagram is shown in Table 2, where accepting states are $s_{N_1,0,p_l,f}$ such that $0 \leq f \leq p_l - 1$ and t_{N_1}. Rejecting states are $q_{p_k,e,p_l,f,rej}$ such that $e \neq 0$ or $f \neq 0$, $0 \leq e \leq p_k - 1$, $0 \leq f \leq p_l - 1$, $t_{p_k,0,y}$ such that $1 \leq y \leq N_2 - 1$, and t_z such that $1 \leq z \leq N_1 - 1$. All other states are non-halting.

M_1^Q checks whether $w_1 = w_2^R$ using N_1 primes and also whether $(w_1w_2) = (w_3w_4)^R$ using N_2 primes. Note that those two jobs have to be done at the same time using composite automata while reading $w_1\sharp w_2$. Hence M_1^Q first splits into $N_1 \cdot N_2$ submachines, each of which is denoted by $M(k,l)$, $1 \leq k \leq N_1$, $1 \leq l \leq N_2$. As shown in Fig.3, $M(k,l)$ has six stages, from stage 1 thorough stage 6. It might be convenient to think that each state of $M(k,l)$ be a pair of state (q_L, q_R) and to think $M(k,l)$ be a composite of M_L and M_R. In stages 1 and 2, M_L has similar state transitions to those of Table 1 for checking $w_1 \neq w_2^R$. M_R has also similar transitions but only for the first part of it, i.e., to compute $w_1w_2 \bmod p_l$. This portion of transitions are given in (2) to (4) of Table 2.

Now we go to stage 3. Here M_L, reading the first $\sharp$, carries out the Fourier transform exactly as M_0^Q (see $(5-a)$ in Table 2). After that M_L, reading the second $\sharp$, execute Inverse Fourier transform from states $s_{m,0,p_l,f}$ $(1 \leq m \leq N_1)$, which is shown in $(6-a)$ of Table 2. In this stage, M_R does nothing; it just shifts the state information about $(w_1w_2) \bmod p_l$ (but only when $w_1 \neq w_2^R$) to stage 4.

Stages 4 and 5 are for the complete reverse operation of stages 2 and 1, respectively. By doing this, the amplitudes for state q_L, which were once in turmoil after stage 2, are reorganized and gathered in specific states, namely $q_{p_k,0,p_l,0,4}$ if $(w_1w_2) = (w_3w_4)^R$. Therefore, what we do is to gather the amplitude of $q_{p_k,0,p_l,0,4}$ to $t_{p_k,0,N_2}$ by Fourier transform reading $\sharp$. Now reading the rightmost symbol, we do another Fourier transform, which gathers the amplitudes of $t_{p_k,0,N_2}$ to t_{N_1}.

For the analysis of error probability, Lemma 5 and 6 are convenient. The basic idea is as follows: When $w_1 \neq w_2^R$, a small amplitude, $\frac{1}{\sqrt{N_2}}\frac{N_0}{N_1}$ is "taken"

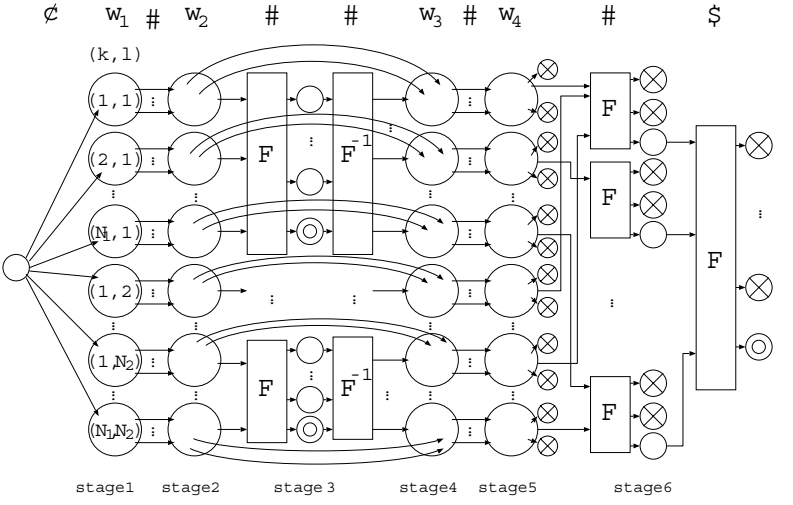


Fig. 3. Qfa M_1^Q.

by each of the N_2 accepting states in stage 3. This is basically the same as M_0^Q since its probability of observing acceptance is $\sum_{l=1}^{N_2} \left(\frac{1}{\sqrt{N_2}} \cdot \frac{N_0}{N_1}\right)^2 = \left(\frac{N_0}{N_1}\right)^2$. So, the problem is how much of the remaining amplitudes distributed on the other states in this stage can be retrieved in the final accepting state t_{N_1} when $(w_1w_2) = (w_3w_4)^R$.

Suppose that we construct a new qfa M' which is exactly the same as M_1^Q but all the N_2 halting states of M_1^Q in stage 3 are changed to non-halting states. Thus M' only checks the longer strings, whether $(w_1w_2) = (w_3w_4)^R$ or not. It is clear that M' accepts with probability exactly one when $(w_1w_2) = (w_3w_4)^R$ and with probability at most $(\frac{N_0'}{N_2})^2$ when $(w_1w_2) \neq (w_3w_4)^R$.

Note that Lemma 5 and 6 also hold for any sequence of unitary transformations and measurements since we can delay measurements and replace them with a single unitary transformation U followed by a measurement [ANTV99]. Next, consider $(i)\psi$, $(ii)\psi_1$ and $(iii)\psi_2$ in Lemma 5 and 6 as $(i)$the quantum state, $(ii)$the superposition of non-halting states and $(iii)$the superposition of accepting states in stage 3 of M_1^Q right after Fourier transform, respectively. In our case, $\|\psi_1\|^2 = (1 - (\frac{N_0}{N_1})^2)$, $\|\psi_2\|^2 = (\frac{N_0}{N_1})^2$ and $\alpha^2 = (\frac{N_0'}{N_2})^2$. Thus, from Lemma 5 we can obtain that when $(w_1w_2) = (w_3w_4)^R$, M_1^Q accepts with probability at least $\left(\frac{N_0}{N_1}\right)^2 + \left(1 - \left(\frac{N_0}{N_1}\right)^2\right)^2$. Also from Lemma 6, when $(w_1w_2) \neq (w_3w_4)^R$, M_1^Q accepts with probabil-

ity at most $\left(\frac{N_0}{N_1}\right)^2 + \left(1-\left(\frac{N_0}{N_1}\right)^2\right)\left(\left(1-\left(\frac{N_0}{N_1}\right)^2\right)^{\frac{1}{2}}\frac{N_0'}{N_2}+\frac{N_0}{N_1}\right)^2 \leq \left(\frac{N_0}{N_1}\right)^2 + \left(1-\left(\frac{N_0}{N_1}\right)^2\right)\left(\frac{N_0'}{N_2}+\frac{N_0}{N_1}\right)^2$.

Finally, we count the number of states in M_1^Q. This is not hard since the whole machine is a composition of two machines, one for using N_1 different primes and the other for N_2 different primes. Therefore the size of the composite machine is $O((N_1N_2)^2 \log N_1 \cdot \log N_2)$. □

Suppose that we set $N_1 = N_0\sqrt{n}$, and $N_2 = dN_0'$. Then $\frac{N_0}{N_1} = \frac{1}{\sqrt{n}}$ and $\frac{N_0'}{N_2} + \frac{N_0}{N_1} \leq \frac{1}{2}$ if we select a sufficiently large constant d, e.g. $d = 3$ when $n \geq 36$. Namely, M_1^Q accepts strings in L_1 with probability at least $1 - 1/n + 1/n^2$ and those not in L_1 with probability at most $\frac{1}{4} + \frac{3}{4n}$. The number of states is $\Theta(\frac{n^5}{\log^2 n})$. The probability distribution for each state of M_1^Q is illustrated in Fig.4 (for example, above $1-1/n+1/n^2$ is the sum of $1/n$ and $1-2/n+1/n^2$).

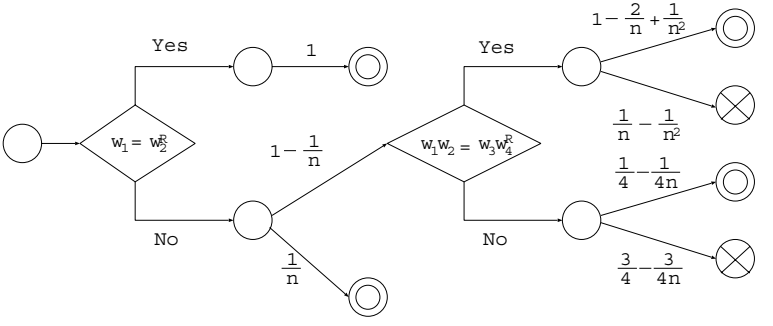


Fig. 4. Probability distribution when $N_1 = N_0\sqrt{n}$,$N_2 = dN_0'$.

Let us consider pfa M_1^P which recognizes $L_1(n)$. The state transition of M_1^P is the same as that of M_1^Q except for Fourier transform and Inverse Fourier transform only M_1^Q performs. If string x satisfies $w_1 \neq w_2^R$, then M_1^P accepts x with probability at most $\frac{N_0}{N_1}$ after reading $w_1\sharp w_2$ instead of with at most $(\frac{N_0}{N_1})^2$ in the case of M_1^Q. There are subtle differences between M_1^P and M_1^Q. For example, in the case of M_1^P, the distributed small amplitudes after reading $w_3\sharp w_4$ can be collected completely (there is some loss due to Inverse Fourier transform in the case of M_1^Q). This causes a slight difference in the accepting probability of the next lemma (proof is omitted).

Lemma 8. *Suppose that M_1^P emulates M_1^Q. Then M_1^P accepts strings in L_1 with probability 1 and those not in L_1 with probability at most $\frac{N_0}{N_1} + \left(1-\frac{N_0}{N_1}\right)\cdot$*

$\frac{N'_0}{N_2}$. *The number of states is approximately the same as the one of* M_1^Q, *i.e.,* $\Theta((N_1N_2)^2 \log N_1 \log N_2)$.

If we set $N_1 = N_0 n$ and $N_2 = dN'_0$, then strings such that $w_1 \neq w_2^R$ are accepted with probability at most $\frac{1}{n}$ after reading $w_1 \sharp w_2$. Thus this probability is the same as qfa M_1^Q such that $N_1 = N_0\sqrt{n}$ and $N_2 = dN'_0$, but the lemma says that we need $\Omega\left(\frac{n^6}{\log^2 n}\right)$ states. See Fig.5 for a probability distribution.

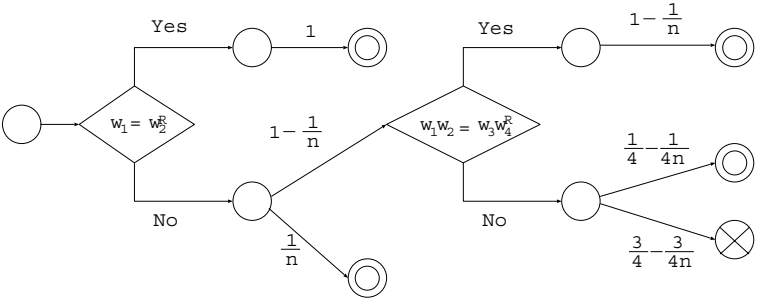


Fig. 5. Probability distribution when $N_1 = N_0 n$, $N_2 = dN'_0$.

Now we are ready to give our main theorem:

Theorem 1. *For any integer c, there is a qfa* M^Q *such that* M^Q *recognizes* $L_2(n, n^c)$ *and the number of states in* M^Q *is* $O\left(\frac{n^{c+4}}{\log^2 n}\right)$.

Proof. The construction of M^Q is easy: We just add a new deterministic transition from the last accepting state in stage 6 of M_1^Q to its initial state by reading $\sharp$, by which we can manage iteration. Also, we need some small changes to handle the very end of the string: Formally speaking, transition (11) in Table 2 is modified into

$$V_\sharp |t_{p_k,0,N_2}\rangle = \frac{1}{\sqrt{N_1}} \sum_{z=1}^{N_1} \exp\left(\frac{2\pi i}{N_2} kz\right) |t_z\rangle,$$

t_{N_1} is now not an accepting state but a non-halting state and two new transitions

$$(10-c)\ V_\$ |q_{p_k,e,p_l,f,4}\rangle = |q_{p_k,e,p_l,f,rej}\rangle$$
$$(12)\qquad V_\sharp |t_{N_1}\rangle = |q_1\rangle$$

are added.

We set $N_1 = 2N_0 n^{c/2}$ and $N_2 = dN'_0$. Then $N_0/N_1 = \frac{1}{2n^{c/2}}$ and $\frac{N'_0}{N_2} + \frac{N_0}{N_1} < \frac{1}{2}$ if we select a sufficiently large constant as d. Suppose that M^Q has not halted yet and is now reading the i-th block $w_{i1}\sharp w_{i2}\sharp\sharp w_{i3}\sharp w_{i4}$. Then, we can conclude the following by Lemma 7: (i) If $w_{i1} = w_{i2}^R$, then M^Q accepts the input with

probability one. (ii) If $(w_{i1}w_{i2}) = (w_{i3}w_{i4})^R$, then $(ii-a)$ M^Q also accepts the input with probability at most $1/4n^c$ and $(ii-b)$ rejects the input with at most $\frac{1}{4n^c} - \frac{1}{16n^{2c}}$ and $(ii-c)$ goes back to the initial state with at least $1 - \frac{2}{4n^c} + \frac{1}{16n^{2c}}$. (iii) If $(w_{i1}w_{i2}) \neq (w_{i3}w_{i4})^R$, then $(iii-a)$ M^Q accepts the input with at most $\frac{1}{4n^c}$, $(iii-b)$ rejects it with at least $\frac{3}{4} - \frac{3}{16n^c}$ and $(iii-c)$ goes back to the initial state with at most $\frac{1}{4} - \frac{1}{16n^c}$. The number of state is $O(n^{c+4}/\log^2 n)$.

Recall that the number of iteration is n^c. Now suppose that the input x is in $L_2(n, n^c)$. Then, the probability that x is rejected is equal to the probability that $(ii-b)$ happens before (i) happens. The probability that $(ii-b)$ happens is at most $\frac{1}{4n^c}$ per iteration, and so the probability that $(ii-b)$ happens in some iteration is at most $n^c \cdot \frac{1}{4n^c} = \frac{1}{4}$. Therefore the probability that x is finally accepted is well larger than $1/2$. Suppose conversely that x is not in $L_2(n, n^c)$. Then the probability that $(ii-a)$ happens in some iteration is the same as above and is at most $\frac{1}{4}$. If M^Q does not meet a block such that $(w_{i1}w_{i2}) \neq (w_{i3}w_{i4})^R$ until the end, then the accepting probability is at most this $1/4$. If M^Q does meet such a block in some iteration, it rejects x with probability at least $(1 - \frac{1}{4})(\frac{3}{4} - \frac{3}{16n^c})$ which is again well above $1/2$. Thus M^Q recognizes $L_2(n, n^c)$. □

Theorem 2. *Suppose that M^P which emulates M^Q recognizes $L_2(n, n^c)$. Then the number of states of M^P is $\Omega(n^{2c+4}/\log^2 n)$.*

Proof. M^P is constructed by applying the same modification (as given in the above proof) to M_1^P. Then it turns out that we must set $N_0'/N_2 \leq 1/d$, where d is a sufficiently large constant, to reject the strings such that $(w_{i1}w_{i2}) \neq (w_{i3}w_{i4})^R$ since M_1^P accepts such bad strings with probability at least $\frac{N_0}{N_1} + (1 - \frac{N_0}{N_1}) \cdot \frac{N_0'}{N_2}$ by Lemma 8. So we have to set $N_2 = dN_0'$ and suppose that we set $N_1 = \frac{1}{a}N_0 n^c$. Then, as shown below, M^P does not recognize $L_2(n, n^c)$ if a is large. That means we have to set $N_1 = \frac{1}{a}N_0 n^c$ for a sufficiently small $a > 0$, which implies, from Lemma 8, that we need $\Omega(n^{2c+4}/\log^2 n)$ states.

Now suppose that the input x includes a long repetition of blocks such that $(w_{i1}w_{i2}) = (w_{i3}w_{i4})^R$. Then x is accepted in each iteration with probability a/n^c. Therefore the probability that this happens in the first k iterations is

$$\sum_{i=1}^{k} \left(1 - \frac{a}{n^c}\right)^{i-1} \cdot \frac{a}{n^c} = 1 - \left(1 - \frac{a}{n^c}\right)^k.$$

Since the number of repetitions $(= k)$ can be as large as n^c,

$$\lim_{n\to\infty} \left(1 - \frac{a}{n^c}\right)^{n^c} = \frac{1}{e^a}.$$

Thus if we select a sufficiently large constant a, then the probability of acceptance can be arbitrarily close to one. Such an M^P does not recognize $L_2(n, n^c)$ obviously, which proves the theorem. □

Table 2. State transition diagram of M_1^Q.

$$
\begin{array}{lll}
(1) & V_{\not\in}|q_0\rangle & = \frac{1}{\sqrt{N_1 N_2}} \sum_{k=1}^{N_1} \sum_{l=1}^{N_2} |q_{p_k,0,p_l,0,1}\rangle, \\
(2-a) & V_0|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,2e,p_l,2f,1}\rangle \quad (0 \le e < \frac{p_k}{2}, \quad 0 \le f < \frac{p_l}{2}), \\
(2-b) & V_0|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,2e,p_l,2f-p_l,1}\rangle \quad (0 \le e < \frac{p_k}{2}, \quad \frac{p_l}{2} < f < p_l), \\
(2-c) & V_0|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,2e-p_k,p_l,2f,1}\rangle \quad (\frac{p_k}{2} < e < p_k, \quad 0 \le f < \frac{p_l}{2}), \\
(2-d) & V_0|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,2e-p_k,p_l,2f-p_l,1}\rangle \quad (\frac{p_k}{2} < e < p_k, \quad \frac{p_l}{2} < f < p_l), \\
(2-e) & V_1|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,2e+1,p_l,2f+1,1}\rangle \quad (0 \le e < \frac{p_k}{2} - 1, \quad 0 \le f < \frac{p_l}{2} - 1), \\
(2-f) & V_1|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,2e+1,p_l,2f+1-p_l,1}\rangle \quad (0 \le e < \frac{p_k}{2} - 1, \quad \frac{p_l}{2} - 1 < f < p_l), \\
(2-g) & V_1|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,2e+1-p_k,p_l,2f+1,1}\rangle \quad (\frac{p_k}{2} - 1 < e < p_k, \quad 0 \le f < \frac{p_l}{2} - 1), \\
(2-h) & V_1|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,2e+1-p_k,p_l,2f+1-p_l,1}\rangle \ (\frac{p_k}{2} - 1 < e < p_k, \frac{p_l}{2} - 1 < f < p_l), \\
(3) & V_\sharp|q_{p_k,e,p_l,f,1}\rangle & = |q_{p_k,e,p_l,f,2}\rangle, \\
(4-a) & V_0|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,\frac{e}{2},p_l,2f,2}\rangle \quad (e: \ even, \quad 0 \le f < \frac{p_l}{2}), \\
(4-b) & V_0|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,\frac{e}{2},p_l,2f-p_l,2}\rangle \quad (e: \ even, \quad \frac{p_l}{2} < f < p_l), \\
(4-c) & V_0|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,\frac{e+p_k}{2},p_l,2f,2}\rangle \quad (e: \ odd, \quad 0 \le f < \frac{p_l}{2}), \\
(4-d) & V_0|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,\frac{e+p_k}{2},p_l,2f-p_l,2}\rangle \quad (e: \ odd, \quad \frac{p_l}{2} < f < p_l), \\
(4-e) & V_1|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,\frac{e-1+p_k}{2},p_l,2f+1,2}\rangle \quad (e: \ even, \quad 0 \le f < \frac{p_l}{2} - 1), \\
(4-f) & V_1|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,\frac{e-1+p_k}{2},p_l,2f+1-p_l,2}\rangle \quad (e: \ even, \quad \frac{p_l}{2} - 1 < f < p_l), \\
(4-g) & V_1|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,\frac{e-1}{2},p_l,2f+1,2}\rangle \quad (e: \ odd, \quad 0 \le f < \frac{p_l}{2} - 1), \\
(4-h) & V_1|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,\frac{e-1}{2},p_l,2f+1-p_l,2}\rangle \quad (e: \ odd, \quad \frac{p_l}{2} - 1 < f < p_l), \\
(5-a) & V_\sharp|q_{p_k,0,p_l,f,2}\rangle & = \frac{1}{\sqrt{N_1}} \sum_{m=1}^{N_1} \exp\left(\frac{2\pi i}{N_1} km\right) |s_{m,0,p_l,f}\rangle, \\
(5-b) & V_\sharp|q_{p_k,e,p_l,f,2}\rangle & = |q_{p_k,e,p_l,f}\rangle \quad (1 \le e < p_k), \\
(6-a) & V_\sharp|s_{m,0,p_l,f}\rangle & = \frac{1}{\sqrt{N_1}} \sum_{r=1}^{N_1} \exp\left(-\frac{2\pi i}{N_1} mr\right) |q_{p_r,0,p_l,f,3}\rangle \ (1 \le m \le N_1), \\
(6-b) & V_\sharp|q_{p_k,e,p_l,f}\rangle & = |q_{p_k,e,p_l,f,3}\rangle \quad (1 \le e < p_k), \\
(7-a) & V_0|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,2e,p_l,\frac{f}{2},3}\rangle \quad (0 \le e < \frac{p_k}{2}, \quad f: \ even), \\
(7-b) & V_0|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,2e,p_l,\frac{f+p_l}{2},3}\rangle \quad (0 \le e < \frac{p_k}{2}, \quad f: \ odd), \\
(7-c) & V_0|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,2e-p_k,p_l,\frac{f}{2},3}\rangle \quad (\frac{p_k}{2} < e < p_k, \quad f: \ even), \\
(7-d) & V_0|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,2e-p_k,p_l,\frac{f+p_l}{2},3}\rangle \quad (\frac{p_k}{2} < e < p_k, \quad f: \ odd), \\
(7-e) & V_1|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,2e+1,p_l,\frac{f-1+p_l}{2},3}\rangle \quad (0 \le e < \frac{p_k}{2} - 1, \quad f: \ even), \\
(7-f) & V_1|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,2e+1,p_l,\frac{f-1}{2},3}\rangle \quad (0 \le e < \frac{p_k}{2} - 1, \quad f: \ odd), \\
(7-g) & V_1|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,2e+1-p_k,p_l,\frac{f-1+p_l}{2},3}\rangle \quad (\frac{p_k}{2} - 1 < e < p_k, \quad f: \ even), \\
(7-h) & V_1|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,2e+1-p_k,p_l,\frac{f-1}{2},3}\rangle \quad (\frac{p_k}{2} - 1 < e < p_k, \quad f: \ odd), \\
(8) & V_\sharp|q_{p_k,e,p_l,f,3}\rangle & = |q_{p_k,e,p_l,f,4}\rangle, \\
(9-a) & V_0|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,\frac{e}{2},p_l,\frac{f}{2},4}\rangle \quad (e: \ even, \quad f: \ even), \\
(9-b) & V_0|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,\frac{e}{2},p_l,\frac{f+p_l}{2},4}\rangle \quad (e: \ even, \quad f: \ odd), \\
(9-c) & V_0|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,\frac{e+p_k}{2},p_l,\frac{f}{2},4}\rangle \quad (e: \ odd, \quad f: \ even), \\
(9-d) & V_0|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,\frac{e+p_k}{2},p_l,p_l,\frac{f+p_l}{2},4}\rangle \quad (e: \ odd, \quad f: \ odd), \\
(9-e) & V_1|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,\frac{e-1+p_k}{2},p_l,\frac{f-1+p_l}{2},4}\rangle \quad (e: \ even, \quad f: \ even), \\
(9-f) & V_1|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,\frac{e-1+p_k}{2},p_l,\frac{f-1}{2},4}\rangle \quad (e: \ even, \quad f: \ odd), \\
(9-g) & V_1|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,\frac{e-1}{2},p_l,\frac{f-1+p_l}{2},4}\rangle \quad (e: \ odd, \quad f: \ even), \\
(9-h) & V_1|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,\frac{e-1}{2},p_l,\frac{f-1}{2},4}\rangle \quad (e: \ odd, \quad f: \ odd), \\
(10-a) & V_\sharp|q_{p_k,0,p_l,0,4}\rangle & = \frac{1}{\sqrt{N_2}} \sum_{y=1}^{N_2} \exp\left(\frac{2\pi i}{N_2} ly\right) |t_{p_k,0,y}\rangle, \\
(10-b) & V_\sharp|q_{p_k,e,p_l,f,4}\rangle & = |q_{p_k,e,p_l,f,rej}\rangle \quad (1 \le f < p_l), \\
(11) & V_\$|t_{p_k,0,N_2}\rangle & = \frac{1}{\sqrt{N_1}} \sum_{z=1}^{N_1} \exp\left(\frac{2\pi i}{N_2} kz\right) |t_z\rangle,
\end{array}
$$

5 Concluding Remarks

The question in this paper is whether or not we can exploit the difference in probability calculation between quantum and probabilistic computations. We have shown that the answer is yes using quantum finite automata. However, what remains apparently is whether or not we can exploit this property for other types of models and/or for other types of problems which are preferably less artificial. Also it should be an important future research to obtain a general lower bound for the number of states which is needed to recognize $L_2(n, n^c)$ by pfa's.

Acknowledgment. We are grateful to Mario Szegedy for his many valuable comments to this research. We also thank the anonymous reviewer for Lemma 5. In the earlier version of this paper, the proof of Lemma 7 was very lengthy. Using this lemma and Lemma 6 which is developed under the same idea as Lemma 5, the proof of Lemma 7 was greatly simplified.

References

[AF98] A. Ambainis and R. Freivalds, "1-way quantum finite automata: strengths, weaknesses and generalizations," *Proceedings of the 39th IEEE Conference on Foundations of Computer Science*, 332-341, 1998.

[AG00] F. Ablayev and A. Gainutdinova, "On the Lower Bounds for One-Way Quantum Automata", *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, 132-140, 2000.

[AI99] M. Amano and K. Iwama, "Undecidability on Quantum Finite Automata", *Proceedings of the 31st ACM Symposium on Theory of Computing*, 368-375, 1999.

[ANTV99] A. Ambainis, A. Nayak, A. Ta-Shma and U. Vazirani, "Dense quantum coding and a lower bound for 1-way quantum automata", *Proceedings of the 31st ACM Symposium on Theory of Computing*, 376-383, 1999.

[Gro96] L. Grover, "A fast quantum mechanical algorithm for database search," *Proceedings of the 28th ACM Symposium on Theory of Computing,* 212-219, 1996.

[KN97] E. Kushilevitz and N. Nisan, "Communication Complexity", *Cambridge University Press*, 1997.

[KW97] A. Kondacs and J. Watrous, "On the power of quantum finite state automata," *Proceedings of the 38th IEEE Conference on Foundations of Computer Science,* 66-75, 1997.

[Nay99] A. Nayak, "Optimal lower bounds for quantum automata and random access codes", *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 369-376, 1999.

Implementing Bead-Sort with P Systems

Joshua J. Arulanandham

Department of Computer Science
University of Auckland, Auckland, New Zealand
hi_josh@hotmail.com

Abstract. In this paper, we implement *Bead–Sort*, a natural sorting algorithm that we introduced in [1], with the new, biochemically inspired *P systems*. In *Bead–Sort*, positive integers are represented by a set of *beads* (like those used in an *Abacus*). The beads representing integers to be sorted are allowed to slide through the *rods* of the Abacus. In this process, the smaller "numbers" always emerge above the larger ones and this creates a natural *comparison* and thus a natural sorting action. This natural sorting phenomenon is "implemented" with a special type of P system — a *tissue P system* that computes by means of communication (using *symport/antiport* rules) only. Beads are represented by *objects* placed within *membranes* of a *tissue P system*; a rod is represented by a 'group' of *membranes* that can communicate with one another by means of symport/antiport rules. The "flow" of objects between the group of membranes representing a rod (using communication rules) reflects the actual flow of beads in the physical system.

1 *Bead-Sort* Algorithm

Bead–Sort is a natural sorting algorithm for positive integers. See [1] where we introduced the new sorting algorithm *Bead–Sort* along with a proof of correctness, analyzed its complexity and discussed different possible implementations in detail. Here, we implement Bead–Sort using the new, biochemically inspired *P systems*. See [7] (where Păun first introduced P systems) and [3] for a detailed discussion on P systems. A brief description of *Bead–Sort* algorithm follows.

We represent positive integers by a set of *beads* (like those used in an *Abacus*) as illustrated in Figure 1; the beads slide through *rods*.

Figure 1 (a) shows the numbers 4 and 3 (represented by beads) attached to rods; beads displayed in Figure 1 (a) appear to be suspended in the air, just before they start sliding down. Figure 1 (b) shows the state of the *frame* (a *frame* is a structure with the rods and beads) after the beads are 'allowed' to slide down. The row of beads representing number 3 has 'emerged' on top of the number 4 (the 'extra' bead in number 4 has dropped down one 'level'). Figure 1 (c) shows numbers of different sizes, suspended one over the other (in a random order). We allow beads (representing numbers 3, 2, 4 and 2) to slide down to obtain the same set of numbers, but in a sorted order again (see

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 115–125, 2002.

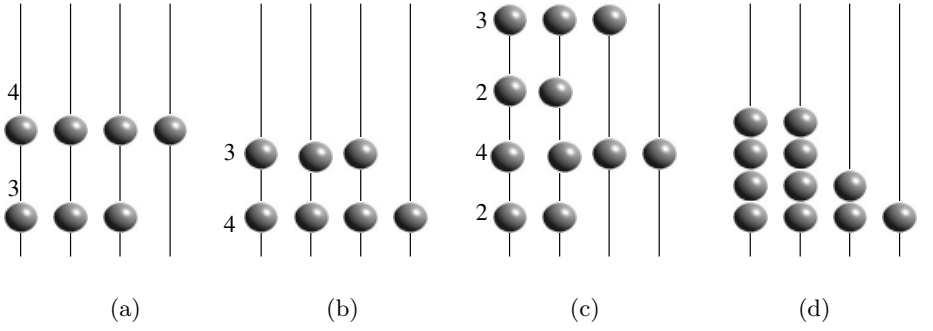


Fig. 1. Bead–Sort examples.

Figure 1 (d)). In this process, the smaller numbers emerge above the larger ones which creates a natural *comparison* (see [2] for online animation).

Rods (vertical lines) are counted always from left to right and *levels* are counted from bottom to top as shown in Figure 2. A *frame* is a structure consisting of *rods* and *beads*.

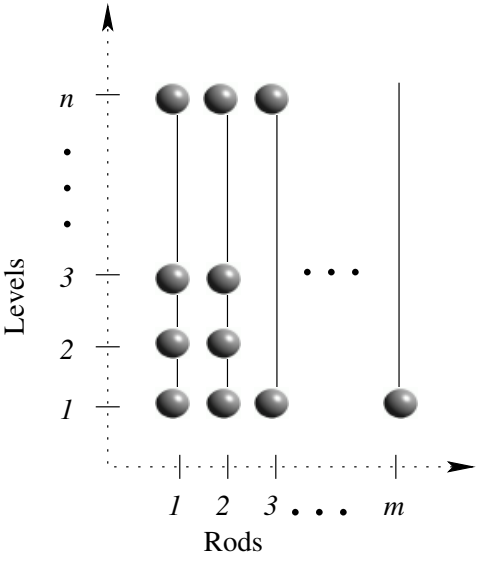


Fig. 2. Bead–Sort conventions.

Consider a set A of n positive integers to be sorted and assume the biggest number in A is m. Then, the frame should have at least m rods and n levels. The Bead–Sort algorithm is the following:

The Bead–Sort Algorithm

> For all $a \in A$ drop a beads (one bead per rod) along the rods, starting from the 1^{st} rod to the a^{th} rod. Finally, the beads, seen level by level, from the n^{th} level to the first level, represent A in ascending order.

The algorithm's run–time complexity ranges from $O(1)$ to $O(S)$ (S is the sum of the input integers) depending on the nature of implementation chosen.

2 Objects = *Beads*, Membranes = *Rods*

A *tissue P system with symport/antiport* is used to implement Bead-Sort. (See Figure 3.) For literature on *tissue P systems* and P systems with *symport/antiport*, see [4], [5], [6], [8] and [9]. Beads are represented by objects x placed within the membranes; a rod is represented by a 'group' of membranes that can communicate with one another by means of symport/antiport rules. Note that the object '$-$' represents "absence of bead". Thus, the objects x and '$-$' together will reflect bead–positions in the initial state of the frame.

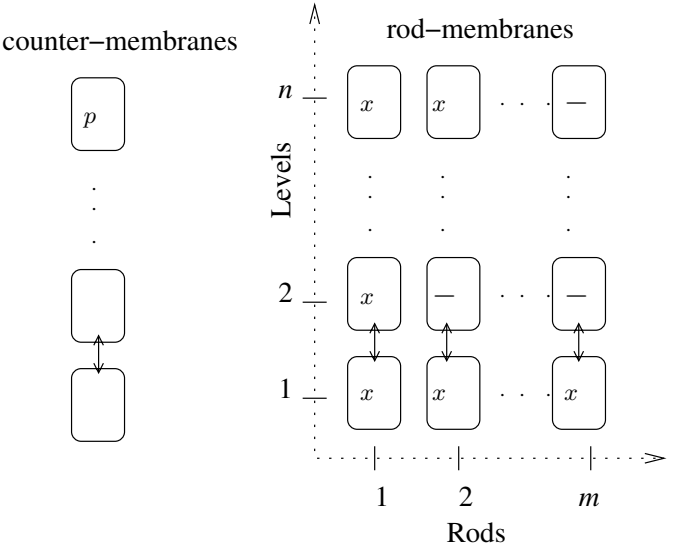


Fig. 3. Membranes = *Rods*, Objects = *Beads*.

Morover, the "flow" of objects between the group of membranes representing a rod (using communication rules) will reflect the actual flow of beads in the physical system. The 'counter membranes' (see Figure 3) along with the object

p will serve the purpose of generating 'clock pulses'; they are useful especially to synchronize, while ejecting the output in the desired sequence. We distinguish these from the other set of membranes — the 'rod membranes'.

More formally, we have a *tissue P system with symport/antiport* of degree $(m \times n) + n$ ($m \times n$ membranes to represent m rods with n levels; extra n membranes to serve as counters). Note that m rods with n levels can sort n positive integers, the biggest among them being m.

In the following sections, we discuss the various symport/antiport rules used to simulate Bead–Sort. A simple P system that can sort 3 integers (biggest among them is 3) is used for illustrative purpose. To explain the necessity of each rule, we initially start with a system having only the 'basic rule', then gradually add more and more complex rules until we arrive at the complete system. We formally define the system only at the final stage.

3 Making Bead-Objects "Fall Down"

As outlined in the previous section, one can set–up the initial state of the frame using a tissue P system. Now, the objects x (representing the beads) should be made to fall down/flow like the real beads in the physical system. This is achieved through a simple antiport rule. See Figure 4 which demonstrates the rule with a simple tissue P system representing 3 rods with 3 levels; we start with the unsorted set $\{2, 1, 3\}$. The antiport rule initiates an exchange of objects x and '$-$'; for instance, according to the rule $(8, x \mid -, 5)$ inside membrane 8, x from membrane 8 will be exchanged with '$-$' in membrane 5. This "simulates" the action of beads falling down. Figure 4 (c) shows the sorted state of the frame; note that the antiport rule can no longer be applied. Also, one can see that not more than $(n - 1)$ steps will be consumed to reach the sorted state using the application of this rule.

Observe that the multiplicity of the object x in the membranes comprising the same *level* taken together (e.g. membranes 1,2 and 3 in Figure 4 comprise *level–1*) denotes an integer. And, these integers are now in the sorted order, viewed level–wise, as seen in Figure 4(c).

Now, we discuss the rules for "reading" the output in the proper sequence, in the following section.

4 Reading the Output

The output can be read in the proper sequence using only symport/antiport rules. But, the rules are a little more complex. The idea is to first "know" when the P system has reached the sorted state; then, the objects from the rod–membranes can be ejected into the environment, starting from *level–1* to *level–n*. The objects from membranes comprising the same *level* will be ejected

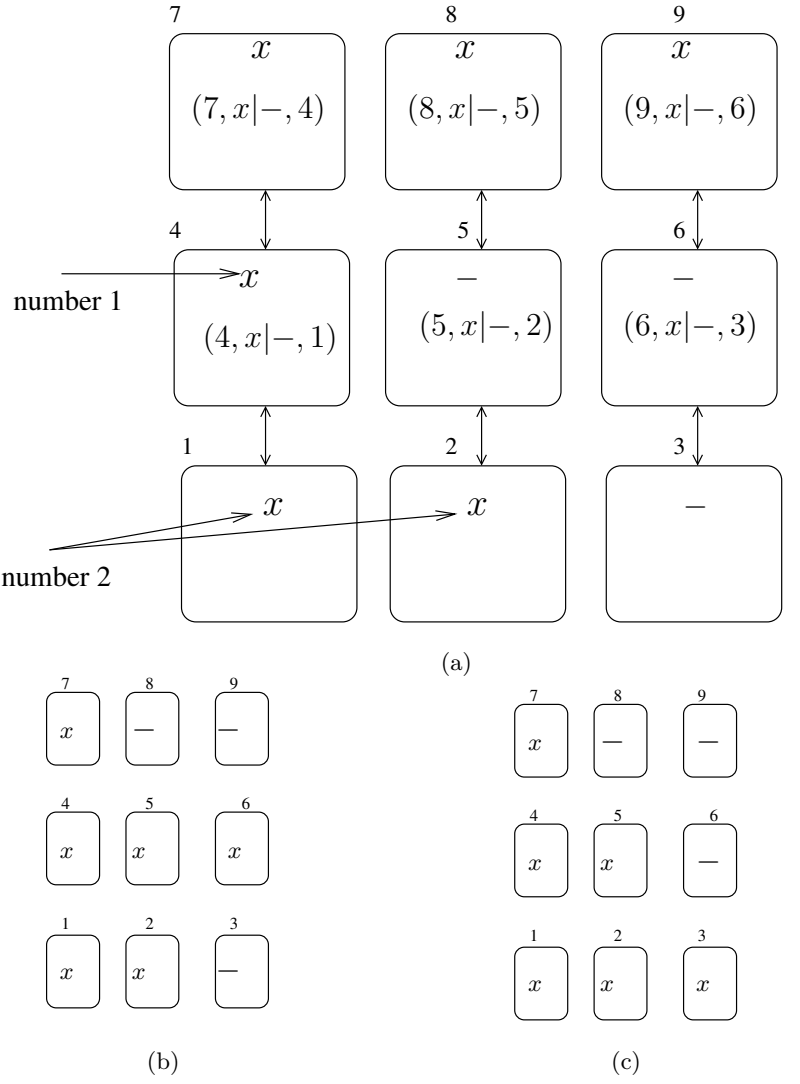


Fig. 4. Making bead–objects "fall down".

simultaneously as a 'single string' to be externally interpreted as representing a distinct integer. Thus the ejected output will form the following language:

{*String of objects from level-1 membranes*, *String of objects from level-2 membranes*, ...}

Note that, externally, the multiplicity of x is "calculated" separately for each string in the language which are ejected at different points of time.

Now, let us include new rules that accomplish these actions related to reading the output. Figure 5 shows the inclusion of new symport rules in the rod–membranes. Observe the inclusion of new objects c_1, c_2, c_3 in the rules. These rules would eject (symport) the x objects from the membranes into the environment along with the "prompting" objects c_1, c_2, c_3, but only if they (c_1, c_2, c_3) are present. It is clear that, one has to first send c_1, c_2, c_3 into the rod–membranes in order to "prompt" the ejection of the output. In what follows, we discuss the rules that do the "prompting".

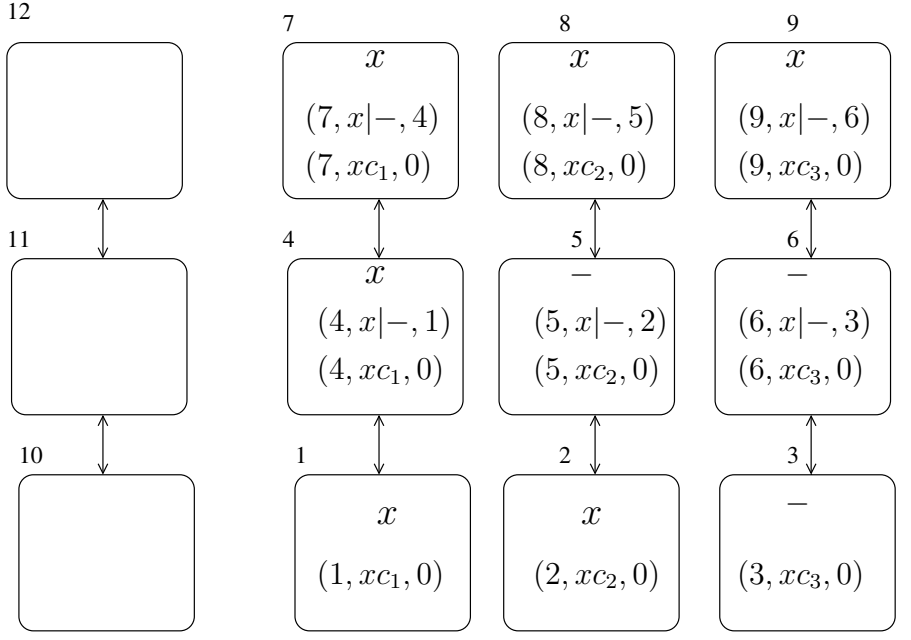


Fig. 5. Ejecting the output.

Remember, before prompting the ejection of output, the system has to first "know" (and ensure) that it has reached the sorted state. The new rules to be discussed help ensure this. Figure 6 shows new rules added to the counter–membranes. Note the presence of object p within membrane 12. These new symport rules would "move" p from one counter membrane to the other, driven by the global clock. After $n-1$ time units of the global clock (2 units in our example), p would have been symported to the counter–membrane denoting *level–1* (membrane 10). Recall from previous section, not more than $(n-1)$ steps will be consumed for the bead–objects to "fall down" and reach the sorted state.

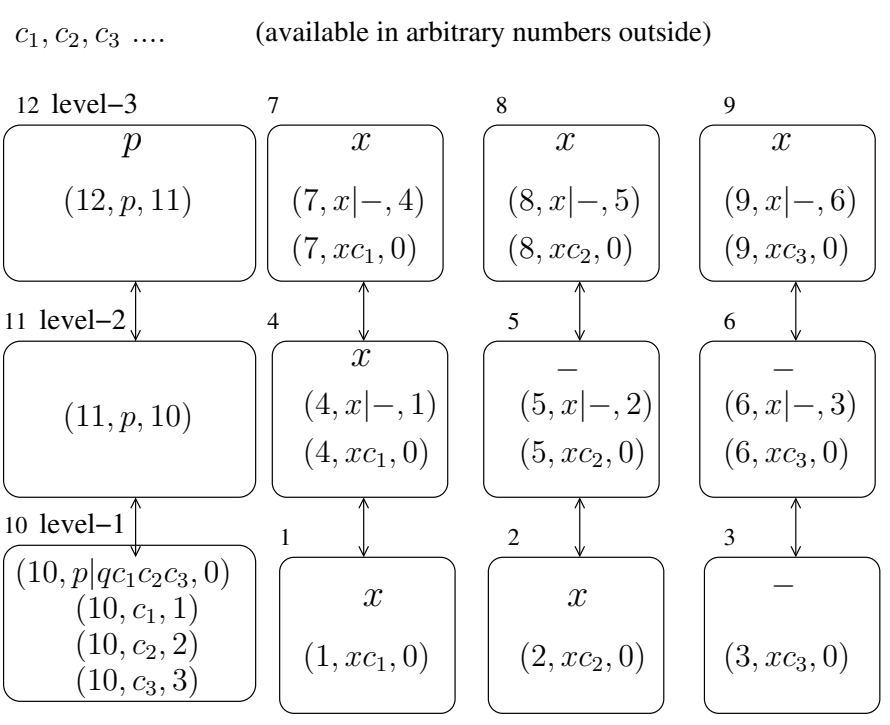


Fig. 6. Prompting the ejection of output from level–1.

Thus, the presence of p within *level–1* counter–membrane (membrane 10) would ensure that the bead–objects have settled down in a sorted state.[1] output. Note that the other rules written in membrane 10 (*level–1* counter–membrane) would start prompting ejection of the output, after p is availabe in membrane 10. The rules get c_1, c_2, c_3 (and another object q) from the environment and transfer them into the group of membranes representing *level–1* (1, 2 and 3 in our case) thus prompting the ejection of x objects as output. (Assume the presence of arbitrary number of c_1, c_2, c_3 in the environment. The need for object q will be discussed shortly.)

Still we need to add more rules which prompt the ejection of (output) strings from *level–2* upto *level–n*. The idea is to move two new objects q and r (alternately) through the counter–membranes, now from *level–2* upto *level–n* ("upward")[2]. The presence of objects q and r in the counter–membranes would trig-

[1] Note that we are forced to adopt this method because, there seems to be no better way to ensure whether the system has reached the final state or not; for instance, we can not deduce this from the 'states' of individual membranes.

[2] We can not use p again as it has already been used by earlier rules; p would activate the same set of actions as before, which is undesirable.

ger certain new rules that would subsequently prompt output from *level–2* upto *level–n* rod–membranes, one by one. (Note, in our case, $n = 3$.) These rules have been included in Figure 7. (Assume the presence of arbitrary number of q's and r's in the environment.)

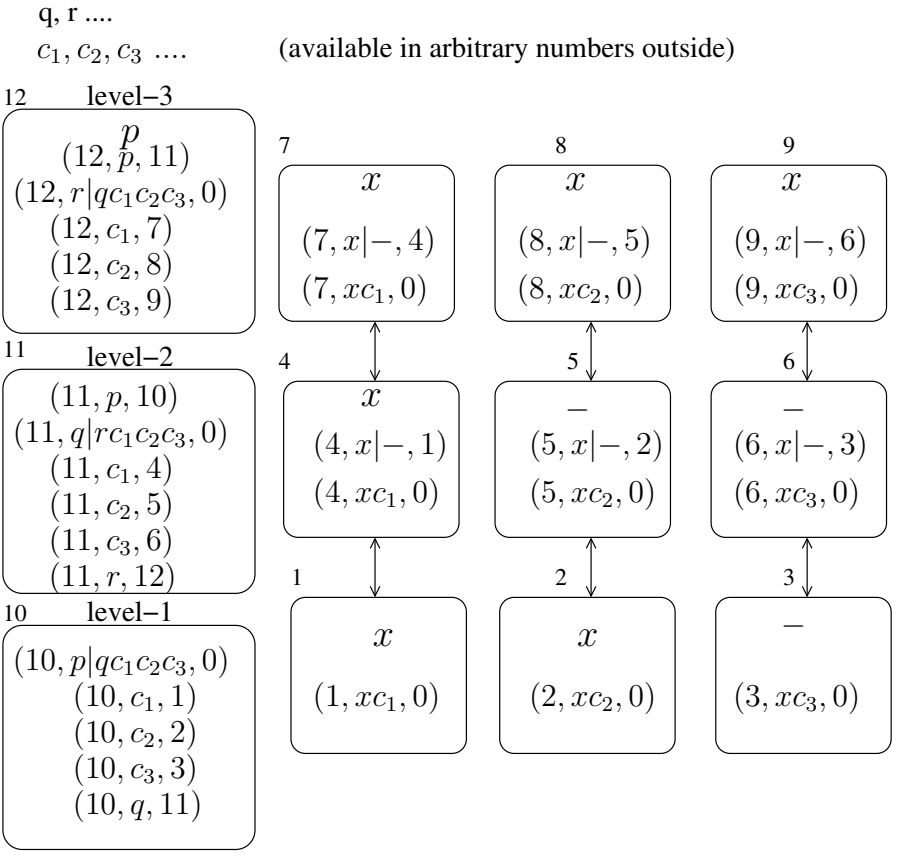


Fig. 7. Prompting the ejection of output from levels 2 & 3.

5 A Sample Illustration

We first formally define a *tissue P system* of degree 12 (3 rods × 3 levels + 3 counter membranes) with *symport/antiport* rules which can sort a set of three positive integers, the maximum among them being three:

$\Pi = (V, T, \omega_1, \omega_2, ..., \omega_{12}, M_0, R)$

where:

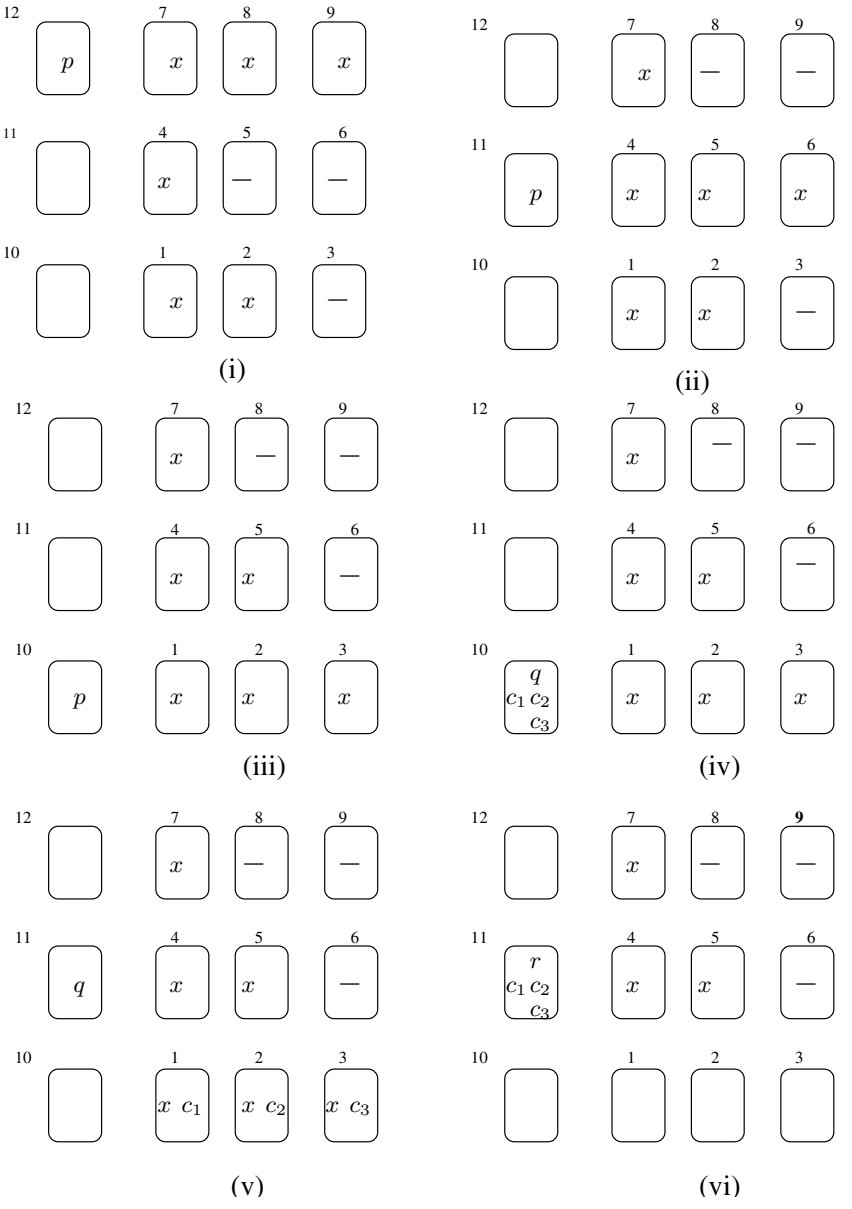


Fig. 8. Sample illustration — intermediate steps (i) to (vi).

(i) V (the alphabet) $= \{x, -, p, q, r, c_1, c_2, c_3\}$

(ii) T (the output alphabet) $\subset V$

$T = \{x, c_1, c_2, c_3\}$ (the multiplicity of c_1, c_2, c_3 is to be ignored in the end)

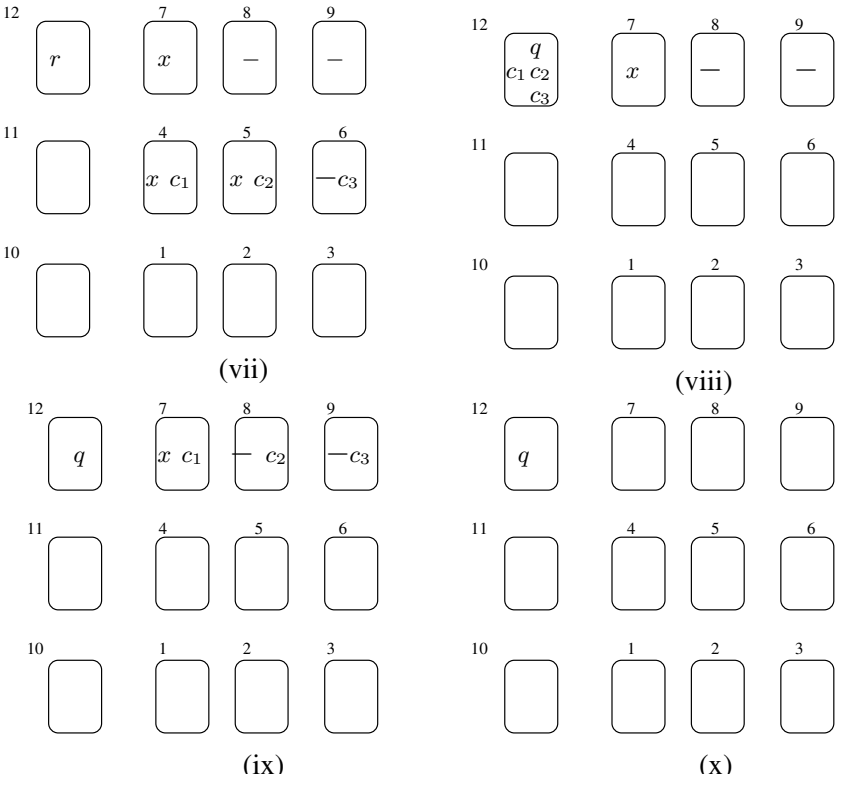


Fig. 9. Sample illustration — steps (vii) to (x).

(iii) ω_1, ...,ω_{12} are strings representing the multisets of objects initially present in the regions of the systems. These are shown in Figure 8 (i).
(iv) $M_0(x) = M_0(-) = M_0(p) = 0$;
$M_0(q) = M_0(r) = M_0(c_1) = M_0(c_2) = M_0(c_3) = \infty$
(v) R is a finite set of (symport/antiport) rules. They are enumerated in Figure 7. Note that we do not use a separate 'output membrane', but prefer to "eject" the output into the environment.

Figures 8 and 9 illustrate sorting {2, 1, 3} with snapshots of all the intermediate configurations, clearly showing the evolution of the system, step by step, until output is "read". Note that steps (i) to (vi) are shown in Figure 8 and steps (vii) to (x) in Figure 9.

Note that the output strings from membranes in levels 1, 2 and 3 will be ejected during steps (vi), (viii) and (x) (of Figures 8 and 9) respectively. One has to ignore the c_1, c_2, c_3s that accompany x objects and take into account only the multiplicity of x in each output string.

in **bold** letters.

6 Conclusion

Bead–Sort, a natural algorithm for sorting positive integers has been implemented with a *tissue P system* that uses only *symport/antiport* rules. The complexity of the algorithm is $O(n)$. As the system uses only communication rules (symport/antiport), the procedure to read the output has been a bit tedious. The P system built for sorting three numbers can be generalized for sorting any n positive integers by simply adding more membranes with similar rules. Some of the antiport rules used involve 5 symbols; a future task could be to try to avoid this, and keep the number of symbols involved close to 2, as it is the case in biology. This could be important from the point of view of a possible bio–implementation.

References

1. J. J. Arulanandham, C. S. Calude, M. J. Dinneen. Bead–Sort: A natural sorting algorithm, *EATCS Bull.*, 76 (2002), 153–162.
2. J. J. Arulanandham. *The Bead–Sort.* Animation, `www.geocities.com/natural_algorithms/josh/beadsort.ppt`.
3. C. S. Calude, Gh. Păun. *Computing with Cells and Atoms: An Introduction to Quantum, DNA, and Membrane Computing*, Taylor and Francis, New York, 2000.
4. C. Martin-Vide, Gh. Păun, J. Pazos, A. Rodriguez-Paton. *Tissue P systems*, Technical Report 421, Turku Center for Computer Science, September 2001.
5. C. Martin-Vide, A. Păun, Gh. Păun. On the power of P systems with symport rules, *J. Univ. Computer Sci.*, 8, 2 (2002), 317–331.
6. C. Martin-Vide, A. Păun, Gh. Păun, G. Rozenberg. Membrane systems with coupled transport: Universality and normal forms, *Fundamenta Informaticae*, 49, 1-3 (2002), 1–15.
7. Gh. Păun. Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
8. A. Păun, Gh. Păun. The power of communication: P systems with symport/antiport, *New Generation Computers*, to appear.
9. A. Păun, Gh. Păun, G. Rozenberg. Computing by communication in networks of membranes, submitted, 2001.

Specification of Adleman's Restricted Model Using an Automated Reasoning System: Verification of Lipton's Experiment*

C. Graciani Díaz, F.J. Martín Mateos, and Mario J. Pérez Jiménez

Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla, Spain Carmen.Graciani@cs.us.es

Abstract. The aim of this paper is to develop an executable prototype of an unconventional model of computation. Using the PVS verification system (an interactive environment for writing formal specifications and checking formal proofs), we formalize the *restricted model*, based on DNA, due to L. Adleman. Also, we design a formal molecular program in this model that solves **SAT** following Lipton's ideas. We prove using PVS the soundness and completeness of this molecular program. This work is intended to give an approach to the opportunities offered by mechanized analysis of unconventional model of computation in general. This approach opens up new possibilities of verifying molecular experiments before implementing them in a laboratory.

1 Introduction

Using formal notations does not ensure us that specifications will be correct. They still need to be validated by permanent reviews but, on the other hand, they support formal deduction; thus, reviews can be supplemented by mechanically checked analysis.

PVS[1] is a verification system: a specification language tightly integrated with a powerful theorem prover and other tools. We present in this paper a formalization, in the PVS verification system [3], of an abstract model of molecular computation: the *restricted model* due to L. Adleman [2]. This work is motivated by the results obtained using ACL2 in [6], where ACL2 is an automated reasoning system that provides both a programming language in which one can model computer systems and a tool that provides assistance to prove properties of these models.

In a molecular model the data are, in general, *tubes* (abstract structures representing a test tube in a laboratory) over a prefixed alphabet. The elements of these tubes encode a collection of DNA strands associating to each symbol of the alphabet an oligonucleotide, under certain conditions. The restricted model

* This work has been supported by DGES/MEC: Projects TIC2000-1368-C03-02 and PB96-1345

[1] The PVS Specification and Verification System http://pvs.csl.sri.com/

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 126–136, 2002.

is based on filtering. In such a model, computations have as input an initial tube containing all possible solutions to the problem to be solved (a coding of them). Then, by performing separations, a tube with only the correct solutions of the problem is obtained.

In order to establish the formal verification of a program designed in the restricted model to solve a decision problem, we must prove two basic results:

• Every molecule in the output tube encodes a valid solution to the problem. That is, if the output is **YES** then the problem has a correct solution (*soundness* of the program).

• Each molecule in the input tube that encodes a correct solution to the problem is in the output tube. That is, if there is such a molecule in the input tube then the output is **YES** (*completeness* of the program).

The paper is organized as follows. In section 2, we briefly introduce the PVS verification system. In section 3, we present Adleman's restricted model and describe how this model is formalized in PVS. Section 4 sets up the SAT problem and how we deal with it in PVS. In section 5 we develop the molecular solution due to Lipton and we provide, in a compact way, a description of the formal verification of the program designed, obtained using PVS. The complete developed theories for this paper are available on the web at `http://www.cs.us.es/cgdiaz/investigacion`.

2 PVS

The *Prototype Verification System (PVS)* is a proof checker based on higher–order logic where types have semantics according to Zermelo–Fraenkel set theory with the axiom of choice [7]. In such a logic one can quantify over functions which take functions as arguments and return them as values.

Specifications are organized into *theories*. They can be parameterized with semantic constructs (constant or types). Also they can import other theories. A prelude for certain standard theories is preloaded into the PVS system. As an example we include in figure 1 the PVS theory `epsilons` (it appears in the prelude) which provides a "choice" function that does not have a nonemptiness requirement. Given a predicate over the type `T`, epsilon produces an element satisfying that predicate if one exists, and otherwise produces an arbitrary element of that type. Note that the type parameter is given as nonempty, which means that there is an nonempty `ASSUMPTION` automatically generated for this theory.

Before a theory may be used, it is typechecked. The PVS typechecker analyzes the theory for semantic consistency and adds semantic information to the internal representation built by the parser. Since this is not a decidable process, the checks which cannot be resolved automatically are presented to the user as assertions called type–correctness conditions.

The PVS prover is goal–oriented. Goals are sequents consisting of antecedents and consequents, e.g. $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$. The conjunction of the antecedents should imply the disjunction of consequents, i.e. $A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$. The proof starts with a goal of the form $\vdash A$, where A is the

```
epsilons [T: NONEMPTY_TYPE]: THEORY
 BEGIN
  p: VAR pred[T]
  x: VAR T
  epsilon(p): T
  epsilon_ax: AXIOM (EXISTS x: p(x)) => p(epsilon(p))
 END epsilons
```

Fig. 1. A PVS Theory

theorem to be proved. The user may type proof commands which either prove the current goal, or result in one or more new goals to prove. In this manner a proof tree is constructed. The original goal is proved when all leaves of the proof tree are recognized as true propositions. Basic proof commands are also combined into strategies.

3 Adleman's Restricted Model in PVS

In this section Adleman's restricted model is described and some considerations about the PVS version of the formalization are given.

Definition 1. *An **aggregate** over an alphabet Σ is a finite multiset of symbols from Σ. A **tube** is a multiset of aggregates over Σ.*

```
AGGREGATES: TYPE = MULTISETS[SIGMA]
TUBES: TYPE = MULTISETS[AGGREGATES]
```

The following are the basic molecular instructions in the Adleman's restricted model.

• `separate`(T, s): Given a tube T and a symbol $s \in \Sigma$ it produces two tubes: $+(T, s)$ (resp. $-(T, s)$) is the tube of all the aggregates of T containing (resp. not containing) the symbol s. As the `separate` operation produces two values, we use two functions to implement it in PVS.

```
sep_p(TT, symb): TUBES =
    LAMBDA (gamma): IF TT(gamma) = 0 OR NOT ms_in(symb, gamma)
                      THEN 0
                    ELSE TT(gamma) ENDIF
sep_n(TT, symb): TUBES =
    LAMBDA (gamma): IF TT(gamma) = 0 OR ms_in(symb, gamma)
                      THEN 0
                    ELSE TT(gamma) ENDIF
```

• `merge`(T_1, T_2): Given tubes T_1 and T_2 it produces their union $T_1 \cup T_2$, considered as multisets.

```
merge(T1, T2): TUBES = ms_union(T1, T2)
```

• `detect`(T): Given a tube T, says **YES** if T contains at least one aggregate, and says **NO** otherwise.

```
DECISION: TYPE = {YES, NO}
detect(TT): DECISION =
  IF ms_empty?(TT) THEN NO ELSE YES ENDIF
```

4 The Satisfiability Problem

SAT Problem: *Given a propositional formula in conjunctive normal form, to determine if there is a truth assignment, whose domain contains all the propositional variables occurring in the formula, such that this assignment satisfies the formula.*

Let φ be a formula in conjunctive normal form where $\varphi = c_1 \wedge \cdots \wedge c_p$ and each clause is a disjunction of literals, $c_i = l_{i,1} \vee \cdots \vee l_{i,r_i}$ for $1 \leq i \leq p$. A literal is a variable or the negation of a variable. Let $Var(\varphi) = \{x_1, \ldots, x_n\}$ the propositional variables occurring in φ.

Conjunctions of clauses and disjunctions of literals will be, in PVS, finite sequences of clauses and literals, respectively. To describe literals in PVS we represent them as a `(marker, propositional variable)` ordered pair. There will be two markers: `positive` and `negative`.

```
MARKERS: TYPE = {positive, negative}
PLIT: TYPE = [MARKERS, PVAR]
PCL: TYPE = finseq[PLIT]
PFORM_fnc: TYPE = finseq[PCL]
```

For a nonempty sequence $S = \{e_0, ..., e_n\}$ we denote $S = S' \circ \{e\}$ where $S' = \{e_0, ..., e_{n-1}\}$ and $e = e_n$. We denote, $e \in S \equiv \exists k(e = e_k)$.

The finite sequence of propositional variables occurring in φ is constructed recursively, in a natural way, over the finite sequence of variables that occurs in the clauses of φ. These sequences are constructed over the variables that appear in the literals occurring in each clause. We define in PVS the functions `PVar` to implement these constructions.

Definition 2.

• *If* $l = (mk, x)$ *then* $Var(l) = x$.

```
PVar((S, PV)): PVAR = PV
```

• *If* $c = \{l_1, \ldots, l_r\}$ *then* $Var(c) = \{Var(l_1), \ldots, Var(l_r)\}$.

```
PVar(PC): finseq[PVAR] = LET L = PC`length IN
      (# length := L,
         seq := LAMBDA (i: below[L]): PVar(PC`seq(i)) #)
```

- $Var(\varphi) = \begin{cases} \{\} & \text{if } \varphi = \{\} \\ Var(c) \circ Var(\varphi') & \text{if } \varphi = \varphi' \circ \{c\} \end{cases}$

```
PVar(PF): RECURSIVE finseq[PVAR] =
  IF PF`length = 0 THEN empty_seq
  ELSE fs_concat(PVar(fs_last(PF)), PVar(fs_red(PF)))
  ENDIF MEASURE PF`length
```

We define in PVS a truth assignment or valuation as an application between propositional variables and truth values, 0 or 1. The opposite value, $\bar{v}$, of a truth value, v, is defined as usual.

```
TRUTH_VALUES: TYPE = {zero, one}
VV: VAR TRUTH_VALUES
opp_value(VV): TRUTH_VALUES =
    IF VV = zero THEN one ELSE zero ENDIF

VALUATIONS: TYPE = [PVAR -> TRUTH_VALUES]
```

For a given valuation the truth value of a literal agrees with the truth value of its variable if the marker is `positive`; otherwise the truth value of the literal is the opposite one. The truth value of a clause in relation to a valuation can be computed recursively from the truth values of its literals. It will be 1 if there is, at least, one literal whose truth value is 1 and 0 otherwise. Similarly, the truth value for a given propositional formula can be computed recursively from the truth values of the clauses occurring in it. It will be 1 if all of them are 1; 0 otherwise.

We define the functions `PVal` in PVS to compute those values.

Definition 3.

- $\pi(l) = \begin{cases} \pi(Var(l)) & \text{if } l = (positive, x) \\ \neg\pi(Var(l)) & otherwise. \end{cases}$

```
PVal(PI, PL): TRUTH_VALUES =
    IF plit_positive?(PL) THEN PI(PVar(PL))
    ELSE opp_value(PI(PVar(PL))) ENDIF
```

- $\pi(c) = \begin{cases} 0 & \text{if } c = \{\} \\ \text{if } c = c' \circ \{l\} & \begin{cases} 1 & \text{if } \pi(l) = 1 \\ \pi(c') & otherwise. \end{cases} \end{cases}$

```
PVal(PI, PC): RECURSIVE TRUTH_VALUES =
  IF length(PC) = 0 THEN zero
  ELSIF PVal(PI, fs_last(PC)) = one THEN one
  ELSE PVal(PI, fs_red(PC)) ENDIF
  MEASURE length(PC)
```

- $\pi(\varphi) = \begin{cases} 1 & \text{if } \varphi = \{\} \\ \text{if } \varphi = \varphi' \circ \{c\} & \begin{cases} 0 & \text{if } \pi(c) = 0 \\ \pi(\varphi') & \text{otherwise.} \end{cases} \end{cases}$

```
PVal(PI, PF): RECURSIVE TRUTH_VALUES =
  IF length(PF) = 0 THEN one
  ELSIF PVal(PI, fs_last(PF)) = zero THEN zero
  ELSE PVal(PI, fs_red(PF))
  ENDIF MEASURE length(PF)
```

Nevertheless, to establish the value of a formula usually it is considered that the application domain is restricted to the set of variables that occur in the formula. We consider, in this sense the following approximation.

Definition 4. *A valuation σ, over a finite sequence S, is an application $\sigma : Ran(S) \to \{0, 1\}$, where $Ran(S)$ is the range of S.*

```
RES_VALUATIONS: TYPE =
  [# domain: finseq[PVAR],
     PVal: [{PV: PVAR | fs_in(PV, domain)} -> TRUTH_VALUES] #]
```

The truth value of a variable for a valuation σ over S is $\sigma(x)$ if $x \in S$, 0 otherwise. The definition of the truth value of a literal, clause or formula for a valuation σ over S is done in the same way as before.

5 Lipton's Solution to SAT Problem

In this regard we follow [5] closely. Given a propositional formula φ, we consider the following directed graph $G_\varphi = (V_\varphi, E_\varphi)$; where $V_\varphi = \{a_i, x_i^j, a_{n+1} |\ 1 \leq i \leq n \wedge (j = 0 \vee j = 1)\}$ and $E_\varphi = \{(a_i, x_i^j), (x_i^j, a_{i+1}) |\ 1 \leq i \leq n \wedge (j = 0 \vee j = 1)\}$.

There is a natural bijection between the set of simple paths starting at a_1 and ending at a_{n+1}, and the set of valuations defined over $Var(\varphi)$. Let $\gamma = a_1 x_1^{j_1} \ldots x_n^{j_n} a_{n+1}$ be such a path. The associated valuation σ_γ is characterized by the following relation: $\sigma_\gamma(x_i) = j_i$ for $1 \leq i \leq n$.

5.1 Design of the Molecular Program

Let us consider $\Sigma = \{a_i, x_i^j |\ i \in \mathbb{N} \wedge x \in \texttt{PVAR} \wedge (j = 0 \vee j = 1)\}$. Given a formula φ with $Var(\varphi) = \{x_1, \ldots, x_n\}$, the initial tube $T_\varphi = \{\!\{\{\!\{a_1 x_1^{j_1} \ldots x_n^{j_n} a_{n+1}\}\!\} |\ \forall i (1 \leq i \leq n \to j_i = 0 \vee j_i = 1)\}\!\}$ (we use the double braces to denote a multiset) can be generated in a laboratory. It is formed in the same way that the test tube of all paths to find the Hamiltonian Path is elaborated in [1].

To describe the symbols x^j for $j \in \{0, 1\}$ in PVS we represent them as a `(propositional variable, truth value)` ordered pair. Let us consider $\Sigma =$ `PVAR` $\times \{0, 1\}$ the alphabet for the computation model. Given a propositional formula, we represent the aggregates for the initial tube by omitting the symbols a_i (they were used in the original experiment as auxiliary vertices of the directed graph considered to construct T_φ).

Initial tube: To construct the initial tube for a given finite sequence of propositional variables, in PVS, we define the following functions.

Definition 5.

$$\texttt{ins}(\texttt{x},\texttt{v},\gamma) = \begin{cases} \gamma & \textit{if } \exists v'((x,v') \in \gamma) \\ \gamma \cup \{(x,v)\} & \textit{otherwise.} \end{cases}$$

```
ins(PV, VV, gamma): AGGREGATES =
    IF EXISTS VV_p: ms_in((PV, VV_p), gamma) THEN gamma
    ELSE ms_incl((PV, VV), gamma) ENDIF
```

$$T_S = \begin{cases} \{\!\{\}\!\} & \textit{if } S = \{\} \\ \{\!\{\{\!\{(x,1)\}\!\}, \{\!\{(x,0)\}\!\}\}\!\} & \textit{if } S = \{x\} \\ \{\!\{\texttt{ins}(x,1,\gamma'), \texttt{ins}(x,0,\gamma') |\ \gamma' \in T_{S'}\}\!\} & \textit{otherwise } (S = S' \circ \{x\}). \end{cases}$$

```
make_tube(S): RECURSIVE TUBES =
  IF S`length = 0 THEN ms_empty
  ELSE LET SR = fs_red(S), SU = fs_last(S) IN
      IF SR`length = 0
        THEN LAMBDA (gamma):
             IF gamma = ms_incl((SU, one), ms_empty) OR
                 gamma = ms_incl((SU, zero), ms_empty)
               THEN 1 ELSE 0 ENDIF
      ELSE LET TT = make_tube(SR) IN LAMBDA (gamma):
         IF EXISTS gamma_p:
              ms_in(gamma_p, TT) AND
               (gamma = ins(SU, one, gamma_p) OR
                 gamma = ins(SU, zero, gamma_p))
           THEN 1 ELSE 0 ENDIF
      ENDIF
  ENDIF MEASURE S`length
initial_tube(PF): TUBES = make_tube(PVar(PF))
```

The specification of $\gamma|x$ states that x appears in γ associated with only one truth value.

Definition 6. $\gamma|x \equiv \exists v((x,v) \in \gamma \wedge (x,\bar{v}) \notin \gamma)$.

```
  pv?_aggregate(gamma, PV): bool =
      EXISTS VV: ms_in((PV, VV), gamma) AND
         NOT ms_in((PV, opp_value(VV)), gamma)
```

Lemma 1. $\gamma \in T_S \wedge x \in S \rightarrow \gamma|x$

```
 initial_tube_carac: LEMMA
  ms_in(gamma, make_tube(S)) AND fs_in(PV, S)
   IMPLIES pv?_aggregate(gamma, PV)
```

Definition 7. *Given $S \subseteq S'$ and $\gamma \in T_{S'}$ we define the associated valuation as $\sigma_\gamma^S : Ran(S) \to \{0,1\}$ where $\sigma_\gamma^S(x) = v$ for a truth value v such that $(x,v) \in \gamma$.*

```
asoc(S)(gamma: {gamma_p| EXISTS SL: fs_subseq(S, SL) AND
                          ms_in(gamma_p, make_tube(SL))}):
  RES_VALUATIONS =
      (# domain := S,
         PVal := LAMBDA (PV: {PV_p| fs_in(PV_p, S)}):
                   epsilon! VV: ms_in((PV, VV), gamma) #)
```

The following properties characterize the above definition.

Lemma 2. $\gamma \in T_S \wedge (x,v) \in \gamma \to \sigma_\gamma^S(x) = v$

```
asoc_carac: LEMMA
  ms_in(gamma, make_tube(S)) AND ms_in((PV, VV), gamma)
     IMPLIES asoc(S)(gamma)`PVal(PV) = VV
```

Lemma 3. *For each valuation $\sigma : Ran(S) \to \{0,1\}$ such that S is nonempty there exists an aggregate $\gamma \in T_S$ such that $\sigma_\gamma^S = \sigma$.*

```
make_tube_sound: LEMMA
  (sigma`domain = S AND S`length > 0) IMPLIES
     EXISTS gamma: ms_in(gamma, make_tube(S)) AND
       asoc(S)(gamma) = sigma
```

Following [4], for each literal $l_{i,j}$ occurring in φ we will denote $l_{i,j}^v = x_m^v$ if $l_{i,j} = x_m$; otherwise $l_{i,j}^v = x_m^{\bar{v}}$. That is, if an aggregate in the initial tube contains the $l_{i,j}^v$ symbol then, for the associated valuation of that aggregate, the literal $l_{i,j}$ has assigned the truth value v.

```
l_symb(PL, VV): [PVAR, TRUTH_VALUES] =
    IF plit_positive?(PL) THEN (PVar(PL), VV)
    ELSE (PVar(PL), opp_value(VV)) ENDIF
```

The operation over the initial tube is done as follows: Let T_1 be the tube whose aggregates encode a valuation that assigns the truth value 1 to the clause c_1. We construct this tube as follows:

Consider the tube that consists of the aggregates in T_0 whose associated valuation assign 1 to the literal $l_{1,1}$. Then, for those aggregates whose associated valuation assigns 0 to $l_{1,1}$, extract the ones that assign 1 to $l_{1,2}$. For the remainder aggregates (encoding a truth assignment that assigns 0 to $l_{1,1} \vee l_{1,2}$) extract those that assign 1 to $l_{1,3}$, and so on.

From T_1, in the same way as before, construct T_2, the tube whose aggregates assign 1 to the clause c_2, and so on.

According to this idea, a molecular program in the restricted model solving SAT is the following one.

```
Procedure sat_lipton(φ)
input: T ← T_φ
      for i = 1 to p do
          T' ← ∅
          for j = 1 to r_i do
              T'' ← +(T, l^1_{i,j})
              T ← −(T, l^1_{i,j})
              T' ← merge(T', T'')
          end for
          T ← T'
      end for
      detect(T)
```

To implement the program `sat_lipton` in PVS, we will define two recursive functions, `inner_l` and `main_l`, one for each `loop`.

Definition 8.

$$\texttt{inner\_l}(c, T, T') = \begin{cases} T' & \text{if } c = \{\} \\ \texttt{inner\_l}(c', -(T, l^1), T' \cup +(T, l^1)) & \text{if } c = c' \circ \{l\} \end{cases}$$

$$\texttt{main\_l}(\varphi, T) = \begin{cases} T & \text{if } \varphi = \{\} \\ \texttt{main\_l}(\varphi', \texttt{inner\_l}(c, T, \{\!\{\}\!\})) & \text{if } \varphi = \varphi' \circ \{c\} \end{cases}$$

```
inner_l(PC, TT, TRes): RECURSIVE TUBES =
  IF PC`length = 0 THEN TRes
  ELSE inner_l(fs_red(PC),
         sep_n(TT, l_symb(fs_last(PC), one)),
         merge(TRes, sep_p(TT, l_symb(fs_last(PC), one))))
  ENDIF MEASURE PC`length
inner_l(PC, TT): TUBES = inner_l(PC, TT, ms_empty)

main_l(PF, TT): RECURSIVE TUBES =
  IF PF`length = 0 THEN TT
  ELSE main_l(fs_red(PF), inner_l(fs_last(PF), TT))
  ENDIF MEASURE PF`length
main_l(PF): TUBES = main_l(PF, initial_tube(PF))

sat_lipton(PF): DECISION = detect(main_l(PF))
```

5.2 Completeness and Soundness in PVS

Next we present results needed to prove, in PVS, the completeness and soundness of the program.

Theorem 1 (completeness). *Given a formula φ such that $Var(\varphi)$ is nonempty we have that* $\exists\pi(\pi(\varphi) = 1) \rightarrow \texttt{sat\_lipton}(\varphi) = \texttt{YES}$

```
completeness: THEOREM
  PVar(PF)`length > 0 AND (EXISTS PI: PVal(PI, PF) = one))
      IMPLIES sat_lipton(PF) = YES
```

To prove this theorem it is sufficient to prove the next one.

Theorem 2 (sat_lipton_compl_gen). $Var(\varphi) \subseteq S \wedge T \subseteq T_S \wedge \gamma \in T \wedge \sigma_\gamma^S(\varphi) = 1 \rightarrow \gamma \in \texttt{main\_l}(\varphi, T)$.

```
sat_lipton_compl_gen: THEOREM
  fs_subseq(PVar(PF), S) AND ms_subset(TT, make_tube(S))
    AND ms_in(gamma, TT) AND PVal(asoc(S)(gamma), PF) = one
      IMPLIES ms_in(gamma, main_l(PF, TT))
```

A similar result for clauses is needed.

Theorem 3.
$Var(c) \subseteq S \wedge T \subseteq T_S \wedge \gamma \in T \wedge \sigma_\gamma^S(c) = 1 \rightarrow \gamma \in \texttt{inner\_l}(c, T, T')$

```
sat_lipton_compl_pcl: LEMMA
  fs_subseq(PVar(PC), S) AND ms_subset(TT, make_tube(S)) AND
  ms_in(gamma, TT) AND PVal(asoc(S)(gamma), PC) = one
    IMPLIES ms_in(gamma, inner_l(PC, TT, TRes))
```

Theorem 4 (soundness). $\texttt{sat\_lipton}(\varphi) = \texttt{YES} \rightarrow \exists\pi(\pi(\varphi) = 1)$.

```
soundness: THEOREM
 sat_lipton(PF) = YES IMPLIES EXISTS PI: PVal(PI, PF) = one
```

To prove this theorem it is sufficient to prove the next one.

Theorem 5. $Var(\varphi) \subseteq S \wedge T \subseteq T_S \wedge \gamma \in \texttt{main\_l}(\varphi, T) \rightarrow \sigma_\gamma^S(\varphi) = 1$.

```
sound_pform: LEMMA
  fs_subseq(PVar(PF), S) AND ms_subset(TT, make_tube(S)) AND
  ms_in(gamma, main_l(PF, TT))
     IMPLIES PVal(asoc(S)(gamma), PF) = one
```

Again a similar result for clauses is also needed.

Theorem 6. $Var(c) \subseteq S \wedge T \subseteq T_S \wedge \gamma \notin T' \wedge \gamma \in \texttt{inner\_l}(\varphi, T, T') \rightarrow \sigma_\gamma^S(c) = 1$.

```
sound_pcl: LEMMA
  fs_subseq(PVar(PC), S) AND ms_subset(TT, make_tube(S))
   AND (NOT ms_in(gamma, TRes)) AND
   ms_in(gamma, inner_l(PC, TT, TRes)))
     IMPLIES PVal(asoc(S)(gamma), PC) = one
```

6 Conclusions

A great part of our work within molecular computing is related to the formalization of the different models that have appeared. Also we have designed and verified programs within these models, to solve classical intractable problems. During this effort we have drawn the conclusion that the next step should be the implementation of these works in an automated reasoning system. Moreover, this approach gives us the possibility of obtaining executable models.

As stated before, the present formalization has been motivated by the results obtained in [6] using ACL2. One advantage of PVS is that it has sets and functions as types and that it is based on a higher–order logic so we gain expressiveness. We also have list data types and tools to deal with recursive definitions and proofs by induction over them.

In this paper a formalization in PVS of the restricted model has been presented. Also a verification using PVS of the Lipton's experiment solving the SAT problem has been obtained. We think that using automated reasoning systems, molecular experiments can be verified before realizing these experiments in a laboratory.

We plan to implement in PVS a more natural specification of this and other molecular models. We also want to establish the formal verification, in the mentioned terms, of the programs designed in those models. We believe and expect that this process would be especially useful to develop a fully or semi automatic strategy to prove the completeness and soundness of programs in unconventional models of computation.

References

1. Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 11, 1994.
2. Leonard M. Adleman. On constructing a molecular computer. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996. DNA Based Computers.
3. Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, and Mandayam Srivas. A tutorial introduction to PVS. Presented at WIFT '95: Workshop on Industrial-Strength Formal Specification Techniques, Boca Raton, Florida, April 1995. Available, with specification files, at `http://www.csl.sri.com/papers/wift-tutorial/`.
4. M.J. Pérez Jiménez, F. Sancho Caparrini, M.C. Graciani Díaz, and A. Romero Jiménez. Soluciones moleculares del problema SAT de la Lógica Proposicional. In *Lógica, Lenguaje e Información, JOLL'2000*, pages 243–252. Ed. Kronos, 2000.
5. Richard J. Lipton. Using DNA to solve NP-complete problems. *Science*, 268:542–545, April 1995.
6. F.J. Martín-Mateos, J.A. Alonso, M.J. Pérez-Jiménez, and F. Sancho-Caparrini. Molecular computation models in ACL2: a simulation of Lipton's experiment solving SAT. In *Third International Workshop on the ACL2 Theorem Prover and Its Applications*, Grenoble, 2002.
7. Sam Owre and Natarajan Shankar. The formal semantics of PVS. Technical Report SRI-CSL-97-2, Computer Science Laboratory, SRI International, Menlo Park, CA, August 1997.

Data Structure as Topological Spaces

Jean-Louis Giavitto and Olivier Michel

LaMI, umr 8042 du CNRS, Université d'Evry – GENOPOLE
523 Place des terasses de l'agora, Tour Evry-2 91000 Evry, France
{giavitto,michel}@lami.univ-evry.fr

Abstract. In this paper, we propose a topological metaphor for computations: computing consists in moving through a path in a data space and making some elementary computations along this path. This idea underlies an experimental declarative programming language called `MGS`. `MGS` introduces the notion of *topological collection*: a set of values organized by a neighborhood relationship. The basic computation step in `MGS` relies on the notion of *path* : a path C is substituted for a path B in a topological collection A. This step is called a *transformation* and several features are proposed to control the transformation applications. By changing the topological structure of the collection, the underlying computational model is changed. Thus, `MGS` enables a unified view on several computational mechanisms. Some of them are initially inspired by biological or chemical processes (Gamma and the CHAM, Lindenmayer systems, Paun systems and cellular automata).

Keywords: Topological collection, declarative and rule-based programming language, rewriting, Paun system, Lindenmayer system, cellular automata, Cayley graphs, combinatorial algebraic topology.

1 Introduction

Our starting point is the following intuitive meaning of a data structure: a data structure s is an *organization* o performed on a data set D. It is customary to consider the pair $s = (o, D)$ and to say that s is a structure o of D (for instance a *list* of *int*, an *array* of *float*, etc.) and to use set theoretic constructions to specify o. However, here, we want to stress the structure o as a set of *places* or *positions*, independently of their occupation by elements of D. Following this perspective, a data structure in [Gia00] is a function from a set of positions to a set of values: this is the point of view promoted by the *data fields* approach. Data fields have been mainly focussed on arrays and therefore on $\mathbb{Z}^n$ as the set of positions [Lis93]. One of our motivations is to define in the same framework the set of positions representing a tree, an array or a multiset independently of the set of values.

Data Structure and Neighborhood. To define a data organization, we adopt a *topological* point of view: *a data structure can be seen as a space*, the set of positions between which *the computation moves*. This topological approach relies

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 137–150, 2002.

on the notion of *neighborhood* to specify a move from one position to one of its neighbor. Although speaking of neighborhood in a data structure is not usual, the relative accessibility from one element to another is a key point considered in a data structure definition:

1. In a simply linked list, the elements are accessed linearly (the second after the first, the third after the second, etc.).
2. In a circular buffer, or in a double-linked list, computation goes from one element to the following *or* to the previous one.
3. From a node in a tree, we can access the sons.
4. The neighbors of a vertex V in a graph are visited after V when traveling through the graph.
5. In a record, the various fields are locally related and this localization can be named by an identifier.
6. Neighborhood relationships between array elements are left implicit in the array data-structure. Implementing neighborhood on arrays relies on an index algebra: index computations are used to code the access to a neighbor. The standard example of index algebra is integer tuples with linear mappings $\lambda x.x \pm 1$ along each dimension (called "Von Neumann" or "Moore" neighborhoods).

This accessibility relation defines a logical neighborhood. And the list of examples can be continued to convince ourselves that a notion of logical neighborhood is fundamental in the definition of a data structure.

Elementary Shifts and Paths. The concept of logical neighborhood in a data structure is not only an abstraction perceived by the programmer and vanishing at the execution, but it does have an actual meaning for the computation. Very often the computation indeed complies with the logical neighborhood of the data elements. For example, the recursive definition of the `fold` function on lists propagates an action to be performed from the the tail to the head of the list. More generally, recursive computations on data structures respect so often the logical neighborhood, that standard high-order functions (e.g. *primitive recursion*) can be automatically defined from the data structure organization (think about catamorphisms and others polytypic functions on inductive types [MFP91]).

These considerations lead to the idea of *path*: in a sequential computation, elements of the data structure are visited one after the other. We assume that if element e' is visited just after element e in a data structure s, then e' must be a neighbor of e. The move from e to e' is called a *shift* and the succession of visited elements makes a path in s. The idea of sequential path can be extended to include parallel modes of computations: multi-dimensional paths must be used instead of one-dimensional paths [GJ92].

Paths and Computations. At this point we can summarize our presentation: we assume that a computation induces a path in a space defined by the neighborhood relationship between the elements of a data structure. At each shift,

some elementary computation is done. Each topological operation used to build a path can then be turned into a new control structure that composes program fragments.

This schema is presented in an imperative setting but can be easily rephrased into the declarative programming paradigm by just specifying the linking of computational actions with path specifications. When a path specification matches an actual path in a data structure, then the corresponding action is triggered. It is very natural, especially in our topological framework, to require that the results of the computational action be *local* : the corresponding data structure transformation is restricted to the value of the the elements involved in the path and eventually to the organization of the path elements and their neighborhood relationships. Such transformation is qualified as local.

This declarative schema induces a rule-oriented style of programming: a rule defines a local transformation by specifying the path to be matched and the corresponding action. A program run consists in the transformation of a whole data structure by the simultaneous application of local transformations to non-intersecting paths. Obviously, such *global* transformation can then be iterated.

Organization of the paper. In section 2 we introduce the `MGS` programming language. `MGS` is used as a vehicle to experiment our topological ideas. We start by the definition of several types of topological collections. The notions underlying the selection of a path and path substitution are then sketched. Section 3 illustrates the previous constructions with two examples taken from the domain of molecular computing and cellular automata. All examples given are real `MGS` programs running on top of one or the other of the two available interpreters. In the last section, we review some related works and some perspectives opened by this research.

2 The `MGS` Programming Language

The topological approach sketched in section 1 is investigated through an experimental declarative programming language called `MGS`. `MGS` is aimed at the representation and manipulation of local transformations of entities structured by *abstract topologies* [GM01c,GM02]. A set of entities organized by an abstract topology is called a *topological collection*. Topological means here that each collection type defines a neighborhood relation specifying both the notion of *locality* and the notion of *sub-collection*. A sub-collection B of a collection A is a subset of elements of A defined by some path and inheriting its organization from A. The *global transformation* of a topological collection C consists in the parallel application of a set of *local transformations*. A local transformation is specified by a rewriting rule r that specifies the change of a sub-collection. The application of a a rewrite rule $r = \beta \Rightarrow f(\beta, ...)$ to a collection A:

1. selects a sub-collection B of A whose elements match the *path pattern* β,
2. computes a new collection C as a function f of B and its neighbors,
3. and specifies the insertion of C in place of B into A.

`MGS` embeds the idea of topological collections and their transformations into the framework of a simple dynamically typed functional language. Collections are just new kinds of values and transformations are functions acting on collections and defined by a specific syntax using rules. Functions and transformations are first-class values and can be passed as arguments or returned as the result of an application. `MGS` is an applicative programming language: operators acting on values combine values to give new values, they do not act by side-effect. In our context, dynamically typed means that there is no static type checking and that type errors are detected at run-time during evaluation. Although dynamically typed, the set of values has a rich type structure used in the definition of pattern-matching, rule and transformations.

2.1 Collection Types

There are several predefined collection types in `MGS`, and also several means to construct new collection types. The collection types can range in `MGS` from totally unstructured with sets and multisets to more structured with sequences and GBFs [GMS95,Mic96,GM01a] (other topologies are currently under development and include Voronoï partitions and abstract simplicial complexes). This paper focuses on two families of collection types: *monoidal collection* and *GBF*.

For any collection type `T`, the corresponding empty collection is written `():T`. The name of a type is also a predicate used to test if a value has this type: `T(`v`)` returns true only if v has type `T`. Each collection type can be subtyped:

```
collection U = T;;
```

introduces a new collection type `U`, which is a subtype of `T`. These two types share the same topology but a value of type `U` can be distinguished from a value of type `T` by the predicate `U`. Elements in a collection `T` can be of any type, including collections, thus achieving *complex objects* in the sense of [BNTW95].

Monoidal Collections. Set, multiset (or bag) and sequences are members of the monoidal collection family. As a matter of fact, a sequence (resp. a multiset) (resp. a set) of values taken in V can be seen as an element of the free monoid V^* (resp. the commutative monoid) (resp. the idempotent and commutative monoid). The join operation in V^* is written by a comma "`,`" and induces the neighborhood of each element: let E be a monoidal collection, then elements x and y in E are neighbors iff $E = u\texttt{,}x\texttt{,}y\texttt{,}v$ for some u and v. This definition induces the following topology:

- for sets (type `set`), each element in the set is neighbor of any other element (because the commutativity, the term describing a set can be reordered following any order);
- for multiset (type `bag`), each element is also neighbor of any other (however, the elements are not required to be distinct as in a set);
- for sequence (type `seq`), the topology is the expected one: an element not at one end has a neighbor at its right.

The comma operator is overloaded in `MGS` and can be used to build any monoidal collection (the type of the arguments disambiguate the collection built). So, the expression `1, 1+1, 2+1, ():set` builds the set with the three elements $1, 2$ and 3, while the expression `1, 1+1, 2+1, ():seq` makes a sequence s with the same three elements. The comma operator is overloaded such that if x and y are not monoidal collections, then x`,`y builds a sequence of two elements. So, the expression `1, 1+1, 2+1` evaluates to the sequence s too.

Group-Based Data Field. Group-based data fields (GBF in short) are used to define organizations with *uniform* neighborhood. A GBF is an extension of the notion of array, where the elements are indexed by the elements of a group, called the *shape* of the GBF [GMS95,GM01a]. For example:

```
gbf Grid2 = < north, east >
```

defines a gbf collection type called `Grid2`, corresponding to the Von Neuman neighborhood in a classical array (a cell above, below, left or right – not diagonal). The two names `north` and `east` refer to the directions that can be followed to reach the neighbors of an element. These directions are the *generators* of the underlying group structure. The right hand side (r.h.s.) of the GBF definition gives a finite presentation of the group structure. The list of the generators can be completed by giving equations that constraint the displacements in the shape:

```
gbf Hexagon = <east, north, northeast; east+north=northeast>
```

defines an hexagonal lattice that tiles the plane, see. figure 1. Each cell has six neighbors (following the three generators and their inverses). The equation `east + north = northeast` specifies that a move following `northeast` is the same has a move following the `east` direction followed by a move following the `north` direction.

A GBF value of type `T` is a partial function that associates a value to some group elements (the group elements are the positions of collection and the the empty GBF is the everywhere undefined function). The topology of `T` is easily visualized as the Cayley graph of the presentation of `T`: each vertex in the Cayley graph is an element of the group and vertices x and y are linked if there is a generator `g` in the presentation such that $x + \mathtt{g} = y$.

A presentation starting with `<` and ending with `>` introduces an *Abelian* organization: they are implicitly completed with the equations specifying the commutation of the generators `g+g' = g'+g`. Currently only free and Abelian groups are allowed: free groups with n generators correspond to n-ary trees and Abelian GBF corresponds to twisted and circular grids (the free Abelian group with n generators generalizes n-dimensional arrays).

2.2 Matching a Path

Path patterns are used in the left hand side (l.h.s) of a rule to match a sub-collection to be substituted. We give only a fragment of the grammar of the patterns:

$$Pat ::= \texttt{x} \mid \texttt{<undef>} \mid p\,\texttt{,}\,p' \mid p\,\texttt{|g>}\,p' \mid p\texttt{*} \mid p\texttt{/}exp \mid p\,\texttt{as x}$$

where p, p' are patterns, `g` is a GBF generator, `x` ranges over the pattern variables and *exp* is an expression evaluating to a boolean value.

Informally, a path pattern can be flattened into a sequence of basic filters and repetition specifying a sequence of positions with their associated values. The order of the matched elements can be forgotten to see the result of the matching as a sub-collection. A pattern variable `x` matches exactly one element (somewhere in the collection) and the identifier `x` can be used in the rest of the rule to denote the value of the matched element. More generally, the naming of the value of a sub-path is achieved using the construction `as`. The constant `<undef>` is used to match an element with an undefined value (i.e., a position with no value). The pattern p`,`p' stands for a path beginning like p and ending like p' (i.e., the last element in path p must be a neighbor of the first element in path p'). For example, `x,y` matches two connected elements (i.e., `y` must be a neighbor of `x`). The neighborhood relationship depends of the collection kind and is decomposed in several sub-relations in the case of a GBF. The comma operator is then refined in the construction p `|g>` p': the first element of p' is the `g`-neighbor of the last element in path p. The pattern p`*` matches a (possibly empty) repetition p`, ... ,`p of path p. Finally, p`/`exp matches the path p only if *exp* evaluates to true. For example

$$\big(\texttt{s/seq(s)}\big)\texttt{+ as S / size(S) == 5}$$

selects a sub-collection `S` of size 5, each element of `S` being a sequence. If this pattern is used against a set, S is a subset, if this pattern is used against a sequence, S is a sub-sequence (that is, an interval of contiguous elements), etc.

2.3 Path Substitution and Transformations

There are several features to control the application of a rule: rules may have priority or a probability of application, they may be guarded and depend on the value of local variables, they "consume" their arguments or not, ... , see [GM01b] for more details.

Substitutions of Sub-collections. A rule $\beta \Rightarrow c$ can be seen as a rule for substituting a path or a sub-collection (recall that a path can be seen as a sub-collection by simply forgetting the order of the elements in the path). For example the rule

$$\texttt{(x / x<3)+ as S} \Rightarrow \texttt{3,4,5,():set}$$

applied to the set `1,2,3,4,():set` returns the set `3,4,5,():set` because `S` matches the subset `1,2,():set` and is replaced by the set `3,4,5,():set`. The final result is computed as `(3,4,():set)` $\cup$ `(3,4,5,():set)`.

Substitutions of Paths. Because the matched sub-collection is also a path, that is a sequence of elements, the `seq` type has a special role when appearing in the r.h.s. of a rule. If the r.h.s. evaluates to a sequence, and if this sequence has the same length as the matched path, then the first element of the sequence is used to replace the first element of the matched path, and so on. This convention is coherent with the sub-collection substitution point of view and simplifies the building of the r.h.s.

For example, suppose that in a GBF of type `Grid2`, we want to model the random walk of a particle `x`. Then, two neighboring elements, one being `x` the other undefined, must exchange their values. This is achieved with only one simple rule

```
x, <undef>  ⇒  <undef>, x
```

without the need to mention the precise neighborhood relationships between the two elements.

Newtonian and Leibnizian Collections. We have mentioned above that the result of replacing a sub-set by a set is computed using set union. More generally, the insertion of a collection C in place of a sub-collection B depends on the "borders" of the involved collections. For example, in a sequence, the sub-collection B defines in general two borders which are used to glue the ends of collection C. The gluing strategy may admit several variations. The programmer can select the appropriate behavior using the rule's attributes.

We discuss here only the *flattening/nesting behavior* linked with the *Leibnizian/Newtonian kind* of the involved collection. Consider the rule:

```
x ⇒ x, x
```

Intuitively, it defines the substitution of one element by two copies of it. However the evaluation of the r.h.s. gives a couple and then, there are two possibilities to replace `x`: one may replace the element matched by `x` by one element which is a couple, *or*, one may "merge" the couple in place of `x` preserving the neighborhood of `x`. For example, if this rule is used on the sequence `1,2,3`, the first interpretation gives the result `(1,1), (2,2), (3,3)` (a sequence of sequences of integers) and the second interpretation returns `1,1,2,2,3,3` (a flat sequence of integers).

The two possibilities, exemplified here for a sequence, hold for any monoidal collection. For a GBF, e.g. `Grid2`, this rule has no meaning, because we cannot insert arbitrary positions between two others without changing the topology of `Grid2`. The set of positions of a GBF exists independently of the values involved in the collection. GBF are *Newtonian* space: the positions exist *a priori* and can be occupied or left empty by the values. In the opposite, monoidal collections have a *Leibnizian* character in the sense that their topology exist only as a relation between the actual values. A consequence is that there is no position with an undefined value in a Leibnizian collection.

Transformations. A transformation `R` is a set of rules:

```
trans R = { ...   rule;  ... }
```

For example, the transformation `trans Mf = { x ⇒ f(x); }` defines a function `Mf` similar to the `map(f)` operator. The expression `Mf(c)` denotes the application of one transformation step to the collection c and a transformation step consists in the parallel application of the rules (modulo the rule application's features). Thus `Mf(c)` computes a new collection where each element e of collection `c` is replaced with `f(`e`)`. Transformations may have parameters, which enables, e.g., the writing of a generic `map`: the transformation `trans M[fct] = { x ⇒ fct(x); }` requires an additional argument when applied. The arguments between brackets are passed to the transformation using a name as in [GAK94]. So, expression `M[fct=\x.x+1](c)` returns a collection where each element of`c` is increased by one. This transformation is polytypic in the sense that it can be applied to any collection type. A transformation step can be easily iterated:

`T[iter=n](c)`	denotes the application of n transformation steps
`T[iter=fixpoint](c)`	application of `T` until a fixpoint is reached
`T[iter=fixrule](c)`	the fixpoint is detected when no rule applies

3 Examples

Because the lack of space, we present here only two simple examples. However, more examples can be found in [GM01b,GM01c,GGMP02] including the tokenization of a sequence of letters, the Eratosthene's sieve, primitive recursion operators on sequences and sets, the computation of the convex hull of a set of points, the maximal segment sum and some other optimization problems, the computation of the disjunctive normal form of a logical formula, direct coding of Lindenmayer systems and Paun systems, Turing-like diffusion-reaction processes, the simulation of a spatially distributed biochemical interaction networks, examples in population dynamics, paradigmatic examples in the field of artificial chemistry and cellular automata, etc.

3.1 Restriction Enzymes

This example shows the ability to nest different topologies to achieve the modeling of a biological structure. We want to represent the action of a set of restriction enzymes on the DNA. The DNA structure is simplified as a sequence of letters `A`, `C`, `T` and `G`. The DNA strings are collected in a multiset. Thus we have to manipulate a multiset of sequences. The following declarations

```
collection DNA = seq;;
collection TUBE = bag;;
```

introduce a subtype called `DNA` of `seq` and a subtype of multisets called `TUBE`.

A restriction enzyme is represented as a rule that splits the DNA strings; for instance a rule like:

```
EcoRI = x+ as X,
        (cut+ as CUT / CUT = "G","A","A","T","T","C",():DNA),
        y+ as Y
      ⇒ (X,"G") :: ("A","A","T","T","C",Y) :: ():TUBE ;
```

corresponds to the *EcoRI* restriction enzyme with recognition sequence `G^AATTC` (the point of cleavage is marked with `^`). The `x+` pattern filters the part of the `DNA` string before the recognition sequence and the result is named X (the `+` operator denotes repetition of neighbors). Identically, Y names the part of the string after the recognition sequence. The r.h.s. of the rule constructs a `TUBE` containing the two resulting `DNA` subsequences (the `::` operator indicates the "consing" of an element with a sequence).

We need an additional rule `Void` for specifying that a `DNA` string without a recognition sequence must be inserted wrapped in a `TUBE`. The two rules are collected into one transformation:

```
trans Restriction = {
    EcoRI = ... ;
    Void = x+ as X ⇒ X :: ():TUBE ;
}
```

In this way, the result of applying the transformation *Restriction* on a `DNA` string is systematically a sequence with only one element which is a `TUBE`. Note that the rule `Void` is applied only when the rule `EcoRI` cannot be applied.

The transformation `Restriction` can then be applied to the `DNA` strings floating in a `TUBE` using the simple transformation:

```
trans React = { dna ⇒ hd(Restriction(dna)) }
```

The operator `hd` gives the head of the result of the transformation *Restriction*, i.e. a `TUBE` containing one or two `DNA` strings. These elements are then merged with the content of the enclosing `TUBE`. The transformation can be iterated until a fixpoint is reached :

```
React[fixpoint]((
    ("C","C","C","G","A","A","T","T","C","A","A",():DNA),
    ("T","T","G","A","A","T","T","C","G","G","G",():DNA),
    ():TUBE ));;
```

returns the tube `("A","A","T","T","C","A","A",():DNA), ("T","T","G",():DNA), ("C","C","C","G",():DNA), ("A","A","T","T","C","G","G","G",():DNA), ():TUBE`.

3.2 The Eden Model

We start with a simple model of growth sometimes called the Eden model (specifically, a type B Eden model [YPQ58]). The model has been used since the 1960's as a model for such things as tumor growth and growth of cities. In this model, a 2D space is partitioned in empty or occupied cells (we use the value `true` for

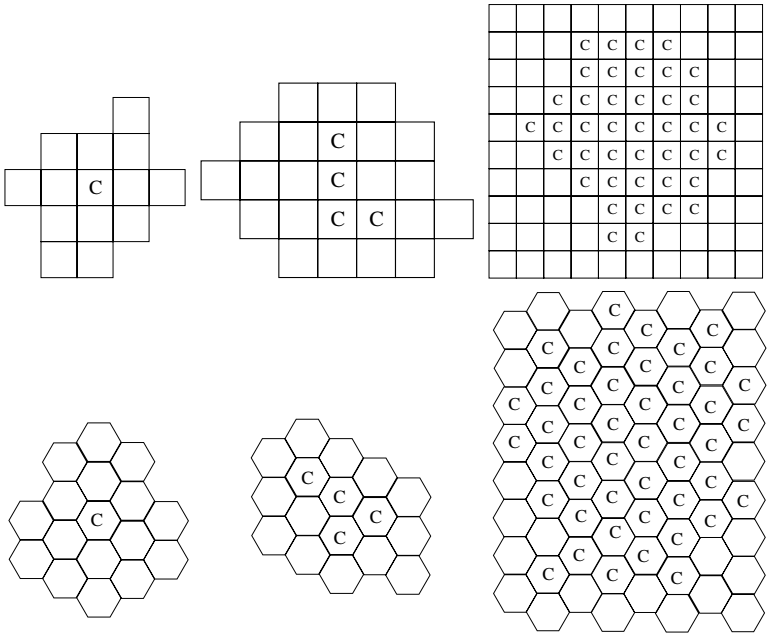


Fig. 1. Eden's model on a grid and on an hexagonal mesh (initial state, and states after the 3 and the 7 time steps). *Exactly the same* transformation is used for both cases. These shapes correspond to a Cayley graph of `Grid2` and `Hexagon` whit the following conventions: a vertex is represented as a face and two neighbors in the Cayley graphs share an edge in this representation. An empty cell has an undefined value. Only a part of the infinite domain is figured.

an occupied cell and left undefined the unoccupied cells). We start with only one occupied cell. At each step, occupied cells with an empty neighbor are selected, and the corresponding empty cell is made occupied.

The Eden's aggregation process is simply described as the following transformation:

```
trans Eden = {   x,<undef> / x  ⇒  x,true ;   }
```

We assume that the boolean value `true` is used to represent an occupied cell, other cells are simply left undefined. Then the previous rule can be read: an occupied element x and an undefined neighbor are transformed into two occupied elements. The transformation `Eden` defines a function that can then be applied to compute the evolution of some initial state. One of the advantages of the `MGS` approach, is that this transformation can apply indifferently on grid or hexagonal lattices, or *any* other collection kind.

It is interesting to compare transformations on GBFs with the genuine cellular automata (CA) formalism. There are several differences. The notion of GBF

extends the usual square grid of CA to more general Cayley graphs. The value of a cell can be arbitrary complex (even another GBF) and is not restricted to take a value in a finite set. Moreover, the pattern in a rule may match an arbitrary domain and not only one cell as it is usually the case for CA. For example the transformation:

```
gbf G2 = <X, Y >;;
trans Turn = { a|X> b |Y-X> c |-X-Y> d |X-Y> e ⇒ a,e,b,c,d; }
```

specify the 90°-turn of a cross in GBF `G2` (see illustration 2). The pattern fragment `b |Y-X> c` specifies that `c` is at the north-west of element `b` if we take the `X` dimension as the *east* direction and the `Y` dimension as the *north* direction.

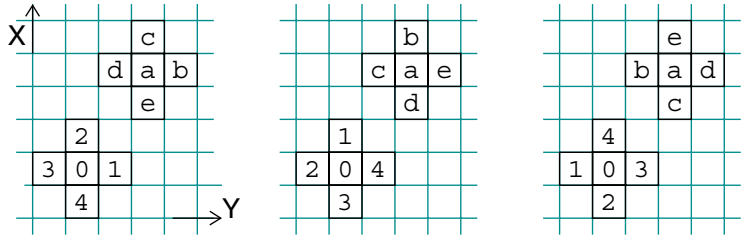


Fig. 2. First and second iteration of transformation `Turn` on the GBF to the left (only defined values are pictured). In contrast with cellular automata, the evolution of a multi-cell domain can be easily specified by a single rule.

4 Related and Future Work

This topological approach formalizing the notion of collection is part of a long term research effort [GMS95] developed for instance in [Gia00] where the focus is on the substructure and in [GM01a] where a general tool for uniform neighborhood definition is developed.

Related Works. Seeing a computation as a path in some abstract space is hardly new: the representation of the execution of a concurrent program as a trajectory in the Cartesian product of the sequential processes dates back to the sixties(in this representation, semaphore operations create topological obstructions and one can study the topology of theses obstructions to decide if a deadlock may occur). However, note that the considered space is based on the elementary computations, not on the involved data structure.

In the same line, the methods for building domains in denotational semantics have clearly topological roots, but they involve the *topology of the set of values*, not the *topology of a value*.

Another example of topological inspiration is the approach taken in [FM97], that rephrased in our perspective, uses a *regular language* to model the displacements resulting from following pointers in `C` data structures.

There exists strong links between GBF and cellular automata, especially considering the work of Z. Róka which has studied CA on Cayley graphs [Rók94]. However, our own works focus on the construction of Cayley graphs as the shape of a data structure and we develop an operator algebra and rewriting notions on this new data type. This is not in the line of Z. Róka which focuses on synchronization problems and establishes complexity results in the framework of CA.

Obviously, Lindenmayer systems [Lin68] correspond to transformations on sequences, and basic Paun systems [Pau00] can be emulated using transformations on multisets.

Formalizations and Implementations. A unifying theoretical framework can be developed [GM01b,GM02], based on the notion of *chain complex* developed in algebraic combinatorial topology. However, we do not claim that we have achieved a useful theoretical framework encompassing the cited paradigm. We advocate that few (topological) notions and a single syntax can be consistently used to allow the merging of these formalisms *for programming* purposes.

Currently, two versions of an `MGS` interpreter exist: one written in `OCAML` (a dialect of `ML`) and one written in `C++`. There are some slight differences between the two versions. For instance, the `OCAML` version is more complete with respect to the functional part of the language. These interpreters are freely available (see url `http://www.lami.univ-evry.fr/mgs`).

Perspectives. The perspectives opened by this preliminary work are numerous. We want to develop several complementary approaches to defines new topological collection types. One approach to extend the GBF applicability is to consider monoids instead of groups, especially automatic monoids which exhibits good algorithmic properties. Another direction is to handle general combinatorial spatial structures like simplicial complexes or G-maps [Lie91].

At the language level, the study of the topological collections concepts must continue with a finer study of transformation kinds. Several kinds of restriction can be put on the transformations, leading to various kind of pattern languages and rules. The complexity of matching such patterns has to be investigated. The efficient compilation of a `MGS` program is a long-term research. We have considered in this paper only one-dimensional paths, but a general n-dimensional notion of path exists and can be used to generalize the substitution mechanisms of `MGS`.

From the applications point of view, we are targeted by the simulation of developmental processes in biology [GGMP02]. Another motivating application is the case of a spatially distributed biochemical interaction networks, for which some extension of rewriting as been advocated, see [FMP00].

Acknowledgments. The authors would like to thanks the members of the "Simulation and Epigenesis" group at Genopole for fruitful discussions and biological motivations. They are also grateful to C. Godin, P. Prusinkiewicz, F. Delaplace and J. Cohen for numerous challenging questions and useful comments. This research is supported in part by the CNRS, the GDR ALP, IMPG and Genopole/Evry.

References

[BNTW95] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, 18 September 1995.

[FM97] P. Fradet and D. Le Metayer. Shape types. In *Proc. of Principles of Programming Languages*, Paris, France, Jan. 1997. ACM Press.

[FMP00] Michael Fisher, Grant Malcolm, and Raymond Paton. Spatio-logical processes in intracellular signalling. *BioSystems*, 55:83–92, 2000.

[GAK94] Jacques Garrigue and H. A¨-Kaci. The typed polymorphic label-selective lambda-calculus. In *Principles of Programming Languages*, Portland, 1994.

[GGMP02] J.-L. Giavitto, C. Godin, O. Michel, and P. Prusinkiewicz. *Biological Modeling in the Genomic Context*, chapter Computational Models for Integrative and Developmental Biology. Hermes, July 2002.

[Gia00] Jean-Louis Giavitto. A framework for the recursive definition of data structures. In *Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-00)*, pages 45–55. ACM Press, September 20–23 2000.

[GJ92] E. Goubault and T. P. Jensen. Homology of higher-dimensional automata. In *Proc. of CONCUR'92*, Stonybrook, August 1992. Springer-Verlag.

[GM01a] J.-L. Giavitto and O. Michel. Declarative definition of group indexed data structures and approximation of their domains. In *Proceedings of the 3nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-01)*. ACM Press, September 2001.

[GM01b] J.-L. Giavitto and O. Michel. `MGS`: a programming language for the transformations of topological collections. Technical Report 61-2001, LaMI – Université d'Évry Val d'Essonne, May 2001. 85p.

[GM01c] Jean-Louis Giavitto and Olivier Michel. Mgs: a rule-based programming language for complex objects and collections. In Mark van den Brand and Rakesh Verma, editors, *Electronic Notes in Theoretical Computer Science*, volume 59. Elsevier Science Publishers, 2001.

[GM02] J.-L. Giavitto and O. Michel. The topological structures of membrane computing. *Fundamenta Informaticae*, 49:107–129, 2002.

[GMS95] J.-L. Giavitto, O. Michel, and J.-P. Sansonnet. Group based fields. In I. Takayasu, R. H. Jr. Halstead, and C. Queinnec, editors, *Parallel Symbolic Languages and Systems (International Workshop PSLS'95)*, volume 1068 of *Lecture Notes in Computer Sciences*, pages 209–215, Beaune (France), 2–4 October 1995. Springer.

[Lie91] P. Lienhardt. Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59–82, 1991.

[Lin68] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[Lis93] B. Lisper. On the relation between functional and data-parallel programming languages. In *Proc. of the 6th. Int. Conf. on Functional Languages and Computer Architectures*. ACM, ACM Press, June 1993.

[MFP91] E. Meijer, M. Fokkinga, and R. Paterson. Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire. In *5th ACM Conference on Functional Programming Languages and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 124–144, Cambridge, MA, August 26–30, 1991. Springer, Berlin.

[Mic96] O. Michel. *Représentations dynamiques de l'espace dans un langage déclaratif de simulation*. PhD thesis, Université de Paris-Sud, Centre d'Orsay, December 1996. N°4596, (in french).

[Pau00] G. Paun. From cells to computers: Computing with membranes (P systems). In *Workshop on Grammar Systems*, Bad Ischl, Austria, July 2000.

[Rók94] Zsuzsanna Róka. One-way cellular automata on Cayley graphs. *Theoretical Computer Science*, 132(1–2):259–290, 26 September 1994.

[YPQ58] Hubert P. Yockey, Robert P. Platzman, and Henry Quastler, editors. *Symposium on Information Theory in Biology*. Pergamon Press, New York, London, 1958.

The Blob: A Basic Topological Concept for "Hardware-Free" Distributed Computation

Frédéric Gruau[1,2] and Philippe Malbos[3]

[1] Université Paris Sud, Laboratoire de Recherche en Informatique, UMR 8623, F-91405 ORSAY Cedex, France, FaxFred@lri.fr,
[2] LIRMM, UMR 5506, 161, rue Ada, F-34392 Montpellier Cedex, France.
[3] Université Montpellier II, Laboratoire Géométrie-Topologie-Algèbre, UMR 5030, F-34095 MONTPELLIER Cedex, France, malbos@math.univ-montp2.fr

Abstract. This paper is part of the Blob Computing Project, a project focusing on the development of an original model of parallelism. The model includes a language and parallel machine that is an asynchronous network of automata. The machine is to handle the placement problem. To accomplish this, it is necessary to represent data structure in a distributed "hardware-free" manner. Our solution is to encode a data structure as a system of color blobs, sharing relationships, and to ensure that these relationships are preserved during the evolution.

1 Introduction

This paper describes one step realized within a long-term project called the Blob Computing Project. The presented blob concept is central to this project. We believe that the blob concept is sufficiently simple and general to interest a wider community outside the scope of the blob computing project. However, to understand blob, it is necessary to provide a short summarization of the project itself. At this stage, we cannot offer proof or simulation demonstrating the ideas of the project, we can only offer proof concerning the specific step made. However, we believe the ideas of the project are clear and appealing enough to motivate a long-term effort.

The Blob Computing Project aim to develop a new model of parallelism encompassing: a language; cellular encoding that codes the development of a graph of cells [4]; and a machine -the blob machine- with a state that represents such a graph and instructions that are able to develop it.

1.1 The Language

Cellular encoding codes the development of a graph of cells, starting from a single ancestor cell applying cell division instructions. Each cell has a copy of the code and reads instructions at a specific location using a reading head. Apart from dividing, cells can also exchange signals and perform computation on these signals, using a Finite State Machine; finite memory is crucial. For example, a cell can simulate a simplified neuron, thus a cellular code can develop and

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 151–163, 2002.

simulate an ANN (Artificial Neural Network). Cellular encoding was designed in 1991 for use in genetic evolution of ANN. The goal was to automatically discover architecture with regularity matching the problem at hand. This was demonstrated on several difficult problems [4].

Cells can also merge, therefore the size of the cell graph can increase and decrease, depending on the requirements for memory and parallelism. Using this feature, a compiler has been designed for compilation from Pascal to cellular code [5]. It keeps the size of the code constant up to a small factor. This shows that cellular encoding is as expressive as Pascal for specifying general purpose computation problems. It is of particular interest given the fact that each cell has a finite memory (a few registers). There is no arbitrary large memory as in the traditional von Neumann model. As soon as an array or whatever large data structure is needed, we just divide cells to create extra memory and memory is thus not inert.

1.2 The Machine

This is the focus of our current research. A preliminary outline can be found in [6]. To obtain a parallel machine, we must solve the mapping problem: which processors will simulate which cell of the cell graph. The mapping must minimize communications by placing neighboring cells on nearby processors. The mapping is the central problem of parallelism, it is NP-hard and thus is addressed using heuristics [1]. We got our idea for the machine by looking at how nature solves this problem. Nature is able to store a few billion neurons and their 100 000 billion connections into the 20 cm^3 of our skull, that is certainly a quite astonishing placement achievement. How does it work? During cell development, the movement of cells is influenced by chemical gradient, electrostatic force that may be genetically programmed. However, behind this, if we go to the hardware mechanism, the first basic law that determines cell position is the everpresent law of physics: pressure and elastic membrane force exerted between cells. A machine trying to reproduce this placement principle can be summarized in the following four ideas.

- The machine is a fine-grain simulation of a physical model, like the one used for weather predictions or fluid dynamics. Fine-grain means its structure can be as elementary as a cellular automaton.
- Each vertex cell and each edge of the cell graph is attributed hardware support consisting of a connected component of the automata network, as if it were a physical object being simulated as such.
- This support is allowed to change through time, with as much freedom as possible, as if the physical object was moving through the space of the network (hardware-free representation).
- This freedom is exploited to optimize the placement of the soft graph on the hard network, continuously, during the development of the soft graph for a cell. For example, communicating cells must be placed nearby. This optimization can be handled by the machine itself simulating laws of pressure and elasticity.

1.3 The Contribution to the Machine

In this paper we solve the problem of obtaining a "hardware-free" representation of data structures that can move through a network of automata. We illustrate this with two examples: simple graphs and hierarchical multisets. The data structure of a cell graph used in cellular encoding is a combination of those two and has been fully addressed in our research report [7].

The difficulty of the problem is that only local interaction is allowed between automata of the network, while a global property needs to be preserved to ensure that the representation movement does not alter the representation itself. Automata have only a few states (three states in the graph example) and they all execute the same rules. Furthermore, since our objective is a truly scalable machine, we drop two constraints, classical in the community of cellular automata modeling of physical models.

- We do not impose synchronicity, thus eliminating the need for a global clock. The only requirement we put on execution order is that two neighboring automata cannot make a transition simultaneously (local mutual exclusion).
- We do not impose regularity in the interconnection patterns, such as a two-dimensional grid. This greatly simplifies multichip implementation. In fact, we accept an arbitrary topology and this can potentially exploit new non-VLSI manufacturing technologies.

Starting from this kind of barebones context, our first concern is to devise rules that will establish deterministic invariants on this disorderly substrate. The blob rule presented will precisely serve this purpose by dividing the network into regions called blobs. While their hardware support moves through the network, blobs persist through time and share topological relationships, able to represent the data structure associated to the developing graph of cells used in cellular encoding [4]. Of course, this is only one step of the simulation. We also need to obtain a hardware-free and distributed representation of each cell's internal code, register and finite state machinery, plus a mechanism to divide blobs into model cell division. The report [6] sketches a proposed solution.

The machine we use can be classified as a distributed system. Usually, such machines implement specific coordination algorithms, such as voting, synchronizing, spanning tree and coloring. Our quite specific goal is to build an algorithm that allows efficient general purpose computation to occur. This achievement, which at first glance may seem totally unrealistic, is in fact a goal that naturally goes along with the idea of developing a graph of cells. Once experience with programming in cellular encoding has been gained, at some point, one realizes that this way of programming is indeed as powerful as a general purpose programming language, the Pascal compiler we previously mentioned proves it.

1.4 The Contribution of the Paper

The simplicity of the blob concept allows us to formulate this contribution beyond the context of the blob machine for a larger community of researchers studying networks of automata. We distinguish three orientations.

- In mathematics, the networks of automata are seen as discretized versions of continuous differential equations [9].
- From a purely computational viewpoint, emphasis is on solving a specific problem, efficiently using synchronous pipe-lined, systolic style models [2,8].
- In experimental physics, the networks are used to model a physical phenomenon [10,11].

We believe that the blob concept adds to these three complementary approaches for the same object.

Mathematics. We introduced a class of non-deterministic evolution rules for networks of automata. Viewing states as colors, we focused on the evolution of color blobs under specific rules called blob rules. We first described the blob concept as a predicate based on topological properties. We have considered two other types of predicates to preserve adjacency between two blobs and encapsulation of blobs. We introduced blob rules as evolution rules satisfying the first predicate locally. We present a combination of blob rules satisfying the adjacency and encapsulation predicate locally. For the three predicates, we proved that local satisfaction implies global satisfaction.

Computation. Blobs can be used to represent a data structure, in a distributed "hardware-free" manner, over a fixed hard network provided with an arbitrary architecture. We give two examples. Adjacency allows representation of a given "soft graph" over the hard network and encapsulation allows representation of hierarchical multiset structures.

Physics. Hardware freedom means that blobs can move through the network while preserving the relationships coding the represented object. This freedom of movement can then be exploited to optimize the shape and the placement of the blobs. We will need, for example, to minimize the diameter of blobs, since any communications cost diameters. Also, blobs will need to be enlarged before division. The natural way to manage blob shape is to discretize physical law.

This work is also relevant to the community working with complex systems consisting of elementary agents interacting using simple *local rules*. Emergent computation [3] is the obtention of a *global behavior*. It is what we obtain in this paper.

2 Non-deterministic Rule for Network of Automata

2.1 Non-deterministic Evolution

The architecture of a network of automata $\mathcal{A}$ is a non-oriented graph $(V_{\mathcal{A}}, E_{\mathcal{A}})$. We will consider an arbitrary architecture.

We define the *nearest neighborhood* $K(a)$ of an automaton a as the sub-graph of $(V_{\mathcal{A}}, E_{\mathcal{A}})$ obtained by taking a, all the automata connected to a by one edge at most and all the edges between those automata. A centered neighborhood $H(a)$ is a couple $(K(a), a)$ where the center a is recorded and noted center(H). $N(a)$ will denote the set of nodes of $K(a)$.

A *configuration* of $\mathcal{A}$ is a labeling of the nodes by a finite set of states Σ called *colors*. We will denote by $l(a,t) \in \Sigma$ the color of the automaton a at the

time t. A *context* of $\mathcal{A}$ is a labeled centered neighborhood. We will denote $\mathcal{H}_{\mathcal{A},\Sigma}$ the set of contexts on $\mathcal{A}$ labeled by Σ. In fact, we will consider the infinite set $\mathcal{H}_\Sigma$ of all the possible contexts for any possible architecture $\mathcal{A}$.

A *non-deterministic evolution rule* is an application $\mathcal{R} : \mathcal{H}_\Sigma \longrightarrow \mathcal{P}(\Sigma) \setminus \{\emptyset\}$. It is thus defined independently of a particular architecture. An *evolution* is a sequence of configurations $c_0, c_1, ..., c_t$. A rule $\mathcal{R}$ is used to constrain the possible evolution in the following way. Any automata a that changes its color between time t and $t+1$, must check that the new color is included in the colors allowed by the rule: $\mathcal{R}(H(a,t))$, where $H(a,t)$ is simply the context associated to a at time t.

We will consider evolution where only one node can change its color at each time step. This correspond to the dynamic called *chaotic dynamic* by F. Robert [9], where at each time step a unique node is randomly selected and its state is modified. The chaotic dynamic implies one node transition per time step and thus simplifies the proofs. However, all results of this paper are also valid for a parallel asynchronous update of the nodes, provided that two neighboring nodes do not make a transition simultaneously (local mutual exclusion). This claim arises from the fact that our evolution rule uses only labels of immediate neighbors. In such a case, it is straightforward to simulate an evolution satisfying local mutual exclusion with a longer sequential evolution. We just have to modify the automata of a given stage in sequence.

For the remainder of this paper, we consider a fixed interval T of time steps, starting at $t = 0$, which can be finite or infinite. An evolution will then be a coloring $l : V_\mathcal{A} \times T \longrightarrow \Sigma$ constrained by a given rule $\mathcal{R}$. We will study rules that ensure an organization of $V_\mathcal{A} \times T$ into color blobs.

2.2 Method of Progressive Refinement

The rules are devised by the intersection of families of *atomic rules*, where each of them is associated to a specific property that should be preserved. In this way, the intersection of all of these atomic rules preserves all the various properties that have to be maintained from the initial configuration. For this purpose, we define a partial order on non-deterministic evolution rules by:

$$\mathcal{R} \leq \mathcal{R}' \text{ if and only if } \mathcal{R}(H) \subset \mathcal{R}'(H), \text{ for every } H \in \mathcal{H}_\Sigma.$$

In such a case, $\mathcal{R}$ is called a sub-rule of $\mathcal{R}'$. Let $\mathcal{I}d$ be the rule that leaves the color unchanged. A rule $\mathcal{R}$ is called *conservative* if it contains $\mathcal{I}d$ as a sub-rule. We will consider conservative rules. The intersection $\mathcal{R} \wedge \mathcal{R}'$ of two conservative rules $\mathcal{R}$ and $\mathcal{R}'$ is always defined by:

$$(\mathcal{R} \wedge \mathcal{R}')(H) = \mathcal{R}(H) \cap \mathcal{R}'(H), \text{ for every } H \in \mathcal{H}_\Sigma.$$

In the case of $\mathcal{R} \leq \mathcal{R}'$, the properties provided by $\mathcal{R}'$ are transferred to $\mathcal{R}$, since the evolution complying with $\mathcal{R}$ can also comply with $\mathcal{R}'$. In this way, the partial order allows us to transfer properties from a rule to its sub-rules.

A simple way to build a complex rule is to progressively obtain it from intersecting successive atomic rules adding the desired list of properties.

3 Blob Structure

3.1 Definition

The blob concept tries to formalize a concept of "greasy indelible patches of color" moving in the space delimited by the network. Blobs are defined as topological objects of an evolution, i.e. a coloring $l : V_{\mathcal{A}} \times T \longrightarrow \Sigma$.

Let E be a set of colors, an E-blob represents a maximal set of automata on a network whose colors stay in E at each time of the evolution, which persists through time.

Definition 1. *Let $l : V_{\mathcal{A}} \times T \longrightarrow \Sigma$ be a coloring and E be a subset of Σ. An E-blob is a subset B of $V_{\mathcal{A}} \times T$ such that each projection $B_t = V_{\mathcal{A}} \times \{t\} \cap B$ satisfies the following four conditions:*

i) the sub-graph induced by B_t on $(V_{\mathcal{A}}, E_{\mathcal{A}})$ is non-empty and connected,
ii) for every $(a,t) \in B_t$, $l(a,t) \in E$,
*iii) B_t is maximal with respect to **i)** and **ii)**,*
iv) $B_t \subset N(B_{t+1})$ and $B_{t+1} \subset N(B_t)$,

where $N(B_t)$ denotes the cover $\bigcup_{a \in B_t} N(a)$ of B_t.

The assertion ***iv)*** means that in one time step the only automata admissible for a change of color are the automata next to the border of the blob.
The previous consideration presents a blob as connected components of $V_{\mathcal{A}} \times T$. Indeed, E-blobs are connected components of $V_{\mathcal{A}} \times T$ endowed with the product topology:

$$(a,t) \text{ is a neighbor of } (a',t') \text{ if } a \text{ is a neighbor of } a' \text{ and } |t - t'| \leq 1.$$

In the sequel, we will assume that $V_{\mathcal{A}} \times T$ is always equipped with this topology and will call *E-component*, a maximal connected component in $V_{\mathcal{A}} \times T$ with the automata colored in E.

3.2 Continuing Blobs: From Local to Global

We are interested in evolution, i.e. colored $V_{\mathcal{A}} \times T$, where the colors define a set of blobs as well as relationships between the blobs: set relationships such as inclusion, intersection or union and topological relationships as separation or adjacency. In order to attain this goal, we follow a systematic method. Let $V \subset V_{\mathcal{A}}$ and $[t_0, t_1] \subset T$.

i) Express the targeted goal as a predicate $\mathsf{P}(V \times [t_0, t_1])$ applied to $V = V_{\mathcal{A}}$ and $[t_0, t_1] = T$.
ii) Consider the biggest rule noted $\mathcal{R}_{\mathsf{P}}$ that preserves the predicate locally on $N(a) \times [t, t+1]$.
iii) Prove that P is then true globally on $V_{\mathcal{A}} \times T$.

Thus the key remark is that it is enough to check the predicate locally, that is, check the predicate on the subsets $N(a,t) \times [t, t+1]$ of $V_{\mathcal{A}} \times T$. As will be seen later, this scheme will not always be necessary, in that particular set properties can be demonstrated directly.

The rule $\mathcal{R}_{\mathsf{P}}$ defined in ***ii)*** exists because the set $\mathcal{S}_{\mathsf{P}}$ of rules preserving P locally verifies:

- it is non empty, it contains the identity $\mathcal{I}d$,
- if $\mathcal{R}, \mathcal{R}' \in \mathcal{S}_{\mathsf{P}}$ then $\mathcal{R} \vee \mathcal{R}' \in \mathcal{S}_{\mathsf{P}}$.

Thus we can take $\mathcal{R}_{\mathsf{P}} = \bigvee_{\mathcal{R} \in \mathcal{S}_{\mathsf{P}}} \mathcal{R}$, where $\mathcal{R} \vee \mathcal{R}'$ is defined as $\mathcal{R} \wedge \mathcal{R}'$ in Sect.2.2.

Our first predicate will implement blob. Theorem 1 proves the passage from local to global for this predicate. We will then use two other predicates to implement adjacency and encapsulation. These two next predicates can be expressed using the blob predicate, thus the proof of the passage from local to global for these two predicates appear just as a corollary of theorem 1.

Definition 2. *Let l be a labeling of $V_{\mathcal{A}} \times T$, $V \subset V_{\mathcal{A}}$, $[t_0, t_1] \subset T$ and E be a set of colors. The predicate $\mathsf{Blob}(E)(V \times [t_0, t_1])$ is defined by:*

all the E-components of $V \times [t_0, t_1]$ are E-blobs.

We note that the rule $\mathcal{R}_{\mathsf{Blob}(E)}$ is conservative.

In order to prove that $\mathcal{R}_{\mathsf{Blob}(E)}$ preserves the predicate globally, we present a characterization of $\mathcal{R}_{\mathsf{Blob}(E)}$. Let H be a context, we consider the graph G obtained from H by deleting the center. We define $C_E(H)$ as the number of E-component of G. We will prove that $\mathcal{R}_{\mathsf{Blob}(E)}$ depends only on this number.

We define the non-deterministic rule $\mathcal{R}_{\mathsf{cyl}(E)}$ which trivially checks the predicate locally, and thus define E-cylinders in $V_{\mathcal{A}} \times T$ with the base constituted by the connected components colored by E at time $t = 0$. Formally, $\mathcal{R}_{\mathsf{cyl}(E)}$ is characterized by $\mathcal{R}_{\mathsf{cyl}(E)}(H) = E$ if $\text{center}(H) \in E$ and $\mathcal{R}_{\mathsf{cyl}(E)}(H) = \Sigma \setminus E$ otherwise.

Lemma 1. ***(Characterization)*** *The rule $\mathcal{R}_{Blob(E)}$ is defined by:*

$$\mathcal{R}_{Blob(E)}(H) = \begin{cases} \Sigma & \text{if } C_E(H) = 1, \\ \mathcal{R}_{cyl(E)}(H) & \text{otherwise.} \end{cases}$$

Proof. We consider an evolution of $\mathcal{A}$ such that the predicate $\mathsf{Blob}(E)$ is true locally. Thus, for every $(a,t) \in V_{\mathcal{A}} \times T$, the E-component of $N(a,t) \times [t, t+1]$ are E-blobs, and in particular the number of these component at the time t is maintained at time $t+1$. Let $C_E(a,t) := C_E(H(a,t))$, the evolution of the color of a depends on $C_E(a,t)$. In the case of $C_E(a,t) = 1$, the coloring of a is not constrained and $\mathcal{R}_{\mathsf{Blob}(E)}(H(a,t) \subset \Sigma$. In the case of $C_E(a,t) = 0$, there is no E-component in $H(a,t)$ except possibly (a,t), (a,t) is then contained in an E-cylinder and $\mathcal{R}_{\mathsf{Blob}(E)}(H(a,t)) = \mathcal{R}_{\mathsf{cyl}(E)}(H(a,t))$. Finally, if $C_E(a,t) > 1$, coloring cannot be applied to a since the number of E-connected component cannot otherwise be maintained.

Conversely, we consider an evolution of $\mathcal{A}$ complying with an evolution rule with the previous definition. Then for every $(a,t) \in V_{\mathcal{A}} \times T$, the E-connected components of $N(a) \times \{t\}$ are maintained, thus so are E-blobs. It follows that $\mathsf{Blob}(E)$ is true locally. □

Theorem 1. ***(Continuance)*** *An evolution complying with $\mathcal{R}_{\mathsf{Blob}(E)}$ checks the predicate $\mathsf{Blob}(E)$ globally on $V_{\mathcal{A}} \times T$.*

Proof. To pass from local to global, it suffices to prove that each E-connected component of $V_{\mathcal{A}} \times T$, for the product topology, is an E-blob. This is true if each E-connected component in $V_{\mathcal{A}} \times \{t\}$ is contained in an E-blob. This result is the outcome of the following lemma. □

Lemma 2. *Let an evolution of $\mathcal{A}$ complying with $\mathcal{R}_{\mathsf{Blob}(E)}$. Then, for each E-component B_t in $V_{\mathcal{A}} \times \{t\}$, there exists a unique E-component B_{t+1} in $V_{\mathcal{A}} \times \{t+1\}$ such that:*

i) $B_{t+1} \subset N(B_t)$,
ii) $B_t \cap B_{t+1} \neq \emptyset$.

Proof. Let B_t be a E-component at time t, let us prove that there exists an E-component B_{t+1} at the time $t+1$ verifying ***i)*** and ***ii)***. Let $N(B_t) = \bigcup_{a' \in B_t} N(a')$ and $a \in N(B_t)$ be the automaton at which the rule is applied at the time t. We distinguish two case, either $a \in N(B_t) \setminus B_t$ or $a \in B_t$.

If $a \notin B_t$, then $l(a,t)$ is not in E. Then there are two cases: either $l(a,t+1)$ is not in E and we choose $B_{t+1} = B_t$, or $l(a,t+1) \in E$. In this last case, in accordance with lemma 1, $C_E(a,t) = 1$, therefore the nearest neighborhood of a without a at time t formed by automata with their color in E, denoted $H^*_E(a,t)$ is connected. In addition we have, $(H(a,t) \setminus \{a\}) \cap B_t \neq \emptyset$ since $a \in N(B_t)$ and consequently, as B_t is an E-connected component, we have $H^*_E(a,t) \cap B_t \neq \emptyset$. By connectivity, we then deduce that $H^*_E(a,t) \subset B_t$, and we choose $B_{t+1} = B_t \cup \{a\}$.

If $a \in B_t$, then $l(a,t) \in E$. Either $l(a,t+1) \in E$, and we choose $B_{t+1} = B_t$, or $l(a,t+1)$ is not in E then we choose $B_{t+1} = B_t \setminus \{a\}$.
In all this cases, B_{t+1} is E-connected.

The uniqueness results immediately from the connectivity and the construction of B_{t+1}. The rule $\mathcal{R}_{\mathsf{Blob}(E)}$ is symmetric, that is we can formally reverse the evolution, consequently the continuance property may also be expressed by inverting the role of t and $t+1$. It follows that under hypothesis of lemma 2 we also have $B_t \subset N(B_{t+1})$. □

3.3 Set Relationships

In this subsection we study conditions on sets of colors defining blobs, under which set relationships can be established between blobs. The proofs of results can be found in the research report [7]. All the proofs are focused on the continuance theorem allowing the change from local to global. So, an initial set of relationships between blobs at time $t = 0$ can be transferred through time.

Recall that blobs refer to subsets of $V_{\mathcal{A}} \times T$, so all set operations can be applied to them, such as $B \subseteq B'$, $B \cup B'$ or $B \cap B'$, are defined as subset of $V_{\mathcal{A}} \times T$.

Proposition 1. *Let E and E' be subsets of Σ such that $E \subset E'$ and an evolution of $\mathcal{A}$ complying with $\mathcal{R}_{Blob(E)} \wedge \mathcal{R}_{Blob(E')}$. If B is an E-blob and B' an E'-blob such that B_0 is a subset of B'_0, then B is a sub-blob of B'.*

Proposition 2. *Let E_i, $i \in I$, be a family of subset of Σ, and an evolution of $\mathcal{A}$ complying with $\bigwedge_{i \in I} \mathcal{R}_{Blob(E_i)}$. If for every $i \in I$, B^i is an E_i-blob, for all E-blob B the following assertions hold:*

i) (union) If $E = \bigcup_{i \in I} E_i$ and $B_0 \subseteq \bigcup_{i \in I} B_0^i$ then $B = \bigcup_{i \in I} B^i$.

ii) (intersection) If $E = \bigcap_{i \in I} E_i$ and $\bigcap_{i \in I} B_0^i \subseteq B_0$ then $\bigcap_{i \in I} B^i \subseteq B$.

3.4 Topological Properties

The topological predicates adjacency and hole can also be defined using the product topology on $V_{\mathcal{A}} \times T$.

Predicate adjacent. Let E and E' be subsets of Σ. We consider B an E-blob and B' an E'-blob and denote $d(B, B')$ the distance between B and B'. For $V \subset V_{\mathcal{A}}$ and $[t_0, t_1] \subset T$ the predicate $\mathsf{adjacent}_{(E,E')}(V \times [t_0, t_1])$ is true if the following conditions are satisfied:
- $\mathsf{Blob}(E)$ and $\mathsf{Blob}(E')$ are true,
- for any E-blob B and E'-blob B', either $d(B, B') \geq 1$ or for all $t \in [t_0, t_1]$, $d(B_t, B'_t) = 0$.

Proposition 3. $adjacent_{(E,E')} = Blob(E \cup E') \wedge Blob(E) \wedge Blob(E')$.

By application of theorem 1, the predicate is verified globally with rule $\mathcal{R}_{\mathsf{Blob}(E \cup E')} \wedge \mathcal{R}_{\mathsf{Blob}(E)} \wedge \mathcal{R}_{\mathsf{Blob}(E')}$. In particular, if B_0 and B'_0 are adjacent in $V_{\mathcal{A}} \times \{0\}$, then every pair B_t and B'_t remain adjacent.

Predicate hole. For encapsulation of blobs, we will use a predicate called hole. Let E be a subset of Σ. For $V \subset V_{\mathcal{A}}$ and $[t_0, t_1] \subset T$ the predicate $\mathsf{hole}_E(V \times [t_0, t_1])$ is true if the following conditions are satisfied:
- $\mathsf{Blob}(E)$ is true,
- for any E-blob B, the connected components of $(V \times [t_0, t_1]) \setminus B$ are colorless blobs, that is subsets of $V \times [t_0, t_1]$ that satisfy properties ***i)***, ***iii)*** and ***iv)*** of the definition 1.

Proposition 4. $hole_E = Blob(\Sigma \setminus E) \wedge Blob(E)$.

By application of theorem 1, the predicate hole_E is verified globally with rule $\mathcal{R}_{\mathsf{Blob}(\Sigma \setminus E)} \wedge \mathcal{R}_{\mathsf{Blob}(E)}$. In particular, if B is an E-blob and B_0 separates $V_{\mathcal{A}} \times \{0\}$ in two disjoint connected components then B separates $\mathcal{A} \times T$ in two disjoint connected components called interior and exterior and can be used as a membrane. Any other blob outside (resp. inside) B will remain outside (resp. inside).

4 Data Structure as Network of Blobs

Using graph of blobs on top of a network of automata allows representation of various data structures. Complex representations combine elementary representations such as graphs or sets. In order to illustrate the method of representing data structure, we consider two basic examples of data structures: graphs and hierarchical multisets. We choose these two, because they are used in [7] to achieve a complete representation of the cell graph used in cellular encoding.

A *representation* is given by the following data:

(1) Σ a set of colors,
(2) $\mathsf{Pred} \subset \mathcal{P}(\Sigma)$ a set of sets of colors, in relation to predicates to maintain,
(3) LAB a set of constraints on the initial configuration on the network of automata.

We then guarantee that the non-deterministic rule defined by the intersection: $\mathcal{R} = \bigwedge_{E \in \mathsf{Pred}} \mathcal{R}_{\mathsf{Blob}(E)}$ will preserve the representation.

4.1 Finite Graph

Graphs can be represented by first-order relational structures, and their property can be written as blob relations. The rule $\mathcal{R}_{\text{GRAPH}}$ which implements the minimal data structure of graph, is defined by the following data:

(1) $\Sigma_{\mathcal{R}_{\text{GRAPH}}} = \{\mathbf{space}, \mathbf{edg}, \mathbf{vert}\}$,
(2) $\mathsf{Pred}_{\mathcal{R}_{\text{GRAPH}}} = \{\{\mathbf{edg}\}, \{\mathbf{vert}\}, \{\mathbf{edg}, \mathbf{vert}\}\}$,
(3) $\text{LAB}_{\mathcal{R}_{\text{GRAPH}}}$: each $\{\mathbf{edg}\}$-blob is adjacent to exactly two distinct $\{\mathbf{vert}\}$-blobs.

Consider a network of automata $\mathcal{A}$ whose initial configuration complies with $\text{LAB}_{\mathcal{R}_{\text{GRAPH}}}$ and whose evolution complies with:

$$\mathcal{R}_{\text{GRAPH}} = \mathcal{R}_{\mathsf{Blob}(\{\mathbf{edg}\})} \wedge \mathcal{R}_{\mathsf{Blob}(\{\mathbf{vert}\})} \wedge \mathcal{R}_{\mathsf{Blob}(\{\mathbf{edg}\} \cup \{\mathbf{vert}\})}.$$

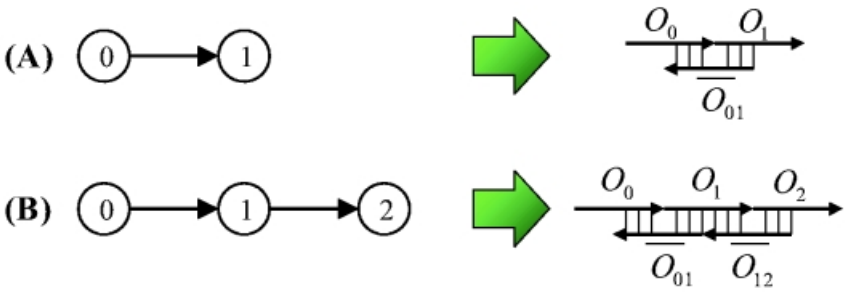
Then the network of blobs on the top of $\mathcal{A}$ represents a graph $\mathcal{G}$ defined by:

- The set of vertices $V_{\mathcal{G}}$ is constituted by the set of $\{\mathbf{vert}\}$-blobs,
- The set of edges $E_{\mathcal{G}}$ is the set of pairs (B_i, B_j) such that there exists a $\{\mathbf{edg}\}$-blob adjacent to a $\{\mathbf{vert}\}$-blob B_i and to a $\{\mathbf{vert}\}$-blob B_j.

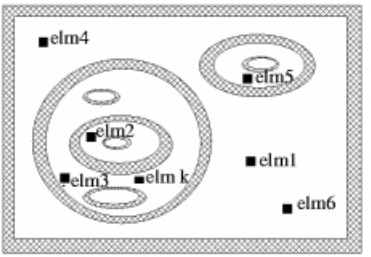
If the evolution of $\mathcal{A}$ is constrained by $\mathcal{R}_{\mathsf{Blob}(\{\mathbf{edg}\} \cup \{\mathbf{vert}\})}$, proposition 3 ensures that the predicate $\mathsf{adjacent}_{(\{\mathbf{edg}\},\{\mathbf{vert}\})}$ is true through the evolution. Therefore the configuration $\text{LAB}_{\mathcal{R}_{\text{GRAPH}}}$ is maintained and the network represents the same graph through time.

4.2 Hierarchical Multiset

The second elementary data structure that we present in this paper is hierarchical multisets span by a finite set, see Fig.1.b. The set Hier$(1..k)$ of hierarchical multisets on $\{1, 2, ..., k\}$ is defined by induction as follows:



(a) Graph



(b) Hierarchical Multiset

Fig. 1. Data structures graph and hierarchical multiset

– multisets on $\{1, 2, ..., k\}$ are elements of Hier(1..k),
– if $H_1, ..., H_l \in$ Hier(1..k), a multiset on $H_1, ..., H_l$ is an element of Hier(1..k).

In this implementation blobs are used to define the sets and elements of the hierarchical multiset. We introduce a particular blob ● to define a partial order on the blob network. The rule $\mathcal{R}_{\text{SET}}$, which implements this data structure, is defined by the following data :

(1) $\Sigma_{\mathcal{R}_{\text{SET}}} = \{\textbf{space}, \textbf{set}, \text{elm}_1, ..., \text{elm}_k\}$,
(2) $\text{Pred}_{\mathcal{R}_{\text{SET}}} = \{\{\textbf{set}\}, \{\textbf{space}\}, \{\textbf{space}, \text{elm}_1, ..., \text{elm}_k\}\}$,
(3) $\text{LAB}_{\mathcal{R}_{\text{SET}}}$: **(a)** there exists a single {**set**}-blob, denoted ● , such that the complement $V_{\mathcal{A}} \setminus$ ● is connected,
(b) each other {**set**}-blob splits the network in two connected components, one containing the blob ● , called exterior, and the other not, called interior.

Let $<$ the order on blobs defined by $X < Y$ if X is in the interior of Y. The hierarchical multiset $\mathcal{M}$ represented by the network of blobs is defined by:

- The sets of $\mathcal{M}$ are $\{\mathbf{set}\}$-blobs and $\{\mathrm{elm}_i\}$-blobs,
- Let X and Y be two sets of $\mathcal{M}$, $X \in Y$ if X precedes Y for the order $<$ and $i \in X$ if $\{\mathrm{elm}_i\}$ precedes X for $<$.

If the evolution of $\mathcal{A}$ is constrained by: $\mathcal{R}_{\mathrm{SET}} = \mathcal{R}_{\mathsf{Blob}(\{\mathbf{set}\})} \wedge \mathcal{R}_{\mathsf{Blob}(\{\mathbf{space}\})} \wedge \mathcal{R}_{\mathsf{Blob}(\{\mathbf{space}, \mathrm{elm}_1, \ldots, \mathrm{elm}_k\})}$, proposition 4 asserts that the initial configuration $\mathrm{LAB}_{\mathcal{R}_{\mathrm{SET}}}$ is maintained by the rule $\mathcal{R}_{\mathrm{SET}}$. The hierarchical multiset represented is globally invariant through time.

5 Future Directions

In summary, the blob is an initial basic topological concept introduced here. It is one step towards the design of the blob machine, outlined in [6] of the Blob Computing Project. We plan to use it in a wider context by labeling both edges and nodes, and using P-blobs instead of E-blob, where P is a locally computable predicate. This is indeed a generalization; E-blobs are also P-blobs, where P checks if the label of the node being rewritten is in E. This generalization leads to a much simplified representation of Membrane-Filament Data Structure.

The major problem remaining to be solved is in finding a local rule to optimize the shape of the blobs. Rules managing the shape will act by computing a probability distribution on the set of transitions allowed by the blob rules. They will discretize simplified models of physical laws, such as for pressure and elasticity. For example, a balloon has a spherical shape due to pressure; this is precisely the shape that minimizes the diameter of a given volume. Elastic force will keep blobs close to one another that need to be adjacent. In fact, the shape rule forms the essence of the blob machine because it constitutes a systematic method for resolving the mapping problem at the hardware level. Optimizing the shape of a blob, and where it should be located, would be the equivalent of solving the mapping problem: which hardware resource should be devoted to each task of a given program.

Acknowledgements. The authors thank Jean Sallantin and Philippe Reitz who encouraged us to find the elementary primitive of cellular encoding, that is the blob.

References

1. M. Cosnard and D. Trystram. Parallel Algorithms and Architectures, International Thomson Computer Press, (1995).
2. M. Delorme and J. Mazoyer, editors. Cellular Automata : A Parallel Model, volume 460 of Mathematics and Its Applications, Kluwer Academic Publishers, (1999).
3. S. Forrest, editor. Emergent Computation, special issue of Physica D, MIT/North-Holland, Cambridge, MA, (1991).
4. F. Gruau. Automatic definition of modular neural networks, Adaptative Behavior, MIT Press, 3 (2), 151-183, (1995).
5. F. Gruau, J.-Y. Ratajszczak and G. Wiber. A neural compiler, Theoretical Computer Science, 141 (1-2), 1-52 (1995).

6. F. Gruau and P. Reitz. Spécifications d'une machine cellulaire, LIRMM, Montpellier, RR-01-064, (2001).
7. F. Gruau and P. Malbos. Une règle d'évolution sur réseau d'automates satisfaisant le cahier des charges d'une partie de la machine cellulaire, LIRMM, Montpellier, RR-01-264, (2001).
8. M. Cosnard, Y. Robert, P. Quinton and M. Tchuente, editors. Parallel Algorithms and Architectures, North-Holland, (1986).
9. F. Robert. Les systèmes dynamiques discrets, In Collection *Mathematiques et Applications de la SMAI*, Springer-Verlag, vol 19, 1-300, (1995).
10. T. Toffoli and N. Margolus. Cellular Automata Machines: A new environment for modeling, MIT Press, Cambridge MA, (1987).
11. S. Wolfram. Cellular Automata and Complexity, Addison Wesley, Reading, Massachusetts, (1994).

Embedding a Logically Universal Model and a Self-Reproducing Model into Number-Conserving Cellular Automata

Katsunobu Imai[1], Kenji Fujita[2], Chuzo Iwamoto[1], and Kenichi Morita[1]

[1] Graduate School of Engineering, Hiroshima University, Higashi-Hiroshima, Japan,
{imai,iwamoto,morita}@iec.hiroshima-u.ac.jp
[2] Anritsu Corp., Tokyo, Japan,
Fujita.Kenji@hh.anritsu.co.jp

Abstract. A number-conserving cellular automaton (NCCA) is a cellular automaton (CA) such that all states of cells are represented by integers and the total number of its configuration is conserved throughout its computing process. It can be thought as a kind of modelization of the physical conservation law of mass or energy. Although NCCAs with simple rules are studied widely, it is quite difficult to design NCCAs with complex transition rules. We show a condition for two-dimensional von Neumann neighbor NCCAs with special symmetric rules and we construct a logically universal NCCA and a self-reproducing NCCA by employing this condition.

1 Introduction

A number-conserving cellular automaton (NCCA) is a cellular automaton such that all states of cells are represented by integers and the total number of its configuration is conserved throughout its computing process. It can be thought as a kind of modelization of the physical conservation law of mass or energy. So NCCAs are used for modeling physical phenomena, for example, for modeling fluid dynamics [4] and highway traffic flow [10].

Boccara et al. [1] studied number conservation of one-dimensional CAs on circular configurations based on a general theorem on additive conserved quantities by Hattori et al. [5]. Durand et al. [3] studied the two-dimensional case and the relation between several boundary conditions. Although their theorems are useful for deciding given CAs are number-conserving or not, it is quite difficult to design NCCAs with complex transition rules.

Another approach for designing a kind of NCCAs are developed by Morita et al. [9]. They use a framework of Partitioned CA (PCA) [8] and this approach is a natural extension of the notion of bit-conservation in BBMCA [7]. They construct a one-dimensional number-conserving (and reversible) PCA that simulates reversible two-counter machine. In the two-dimensional case, Washio [12] shows that a self-reproducing CA can be embedded into a number-conserving (and reversible) PCA. Although in a number-conserving PCA, the total number

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 164–175, 2002.

of its configuration is conserved throughout its computing process, each cell has multiple parts and it cannot be regarded as a usual type of CA.

So in this paper, we show a number-conserving condition for two-dimensional von Neumann neighbor CAs with a special symmetric rule, and employing this condition, we construct a logically universal NCCA by embedding a Wireworld [2] like model. We also construct a self-reproducing NCCA by embedding a Langton's self-reproducing Loop [6].

2 Two-Dimensional Cellular Automata and Their Number-Conserving Conditions

Definition 1. A *deterministic two-dimensional von Neumann neighbor cellular automaton* is a system defined by

$$A = (\mathbf{Z^2}, Q, f, q),$$

where $\mathbf{Z}$ is the set of all integers, Q is a non-empty finite set of internal states of each cell, $f : Q^5 \to Q$ is a mapping called a *local function* and $q \in Q$ is a *quiescent state* that satisfies $f(q, q, q, q, q) = q$.

A *configuration* over Q is a mapping $\alpha : \mathbf{Z}^2 \to Q$ and the set of all configurations over Q is denoted by $\mathrm{Conf}(Q)$, i.e., $\mathrm{Conf}(Q) = \{\alpha | \alpha : \mathbf{Z}^2 \to Q\}$. In this paper, we only consider CAs with finite configurations, i.e., the number of cells which states are not quiescent is finite.

The function $F : \mathrm{Conf}(Q) \to \mathrm{Conf}(Q)$ defined as follows is called the *global function* of A.

$$\forall (x, y) \in Z^2,$$
$$F(\alpha)(x, y) = f(\alpha(x, y), \alpha(x, y + 1), \alpha(x + 1, y), \alpha(x, y - 1), \alpha(x - 1, y)).$$

Let $Q = N_{[l,m]} \equiv \{l, l+1, \ldots, m-1, m\}, (l, m \in Z, l \leq m)$, $q = k \in N_{[l,m]}$. A is said to be *number-conserving* when it satisfies the following condition for all finite configurations α.

$$\sum_{(x,y)\in\mathbf{Z}^2} \{F(\alpha)(x, y) - \alpha(x, y)\} = 0 \tag{1}$$

Next we define some symmetry conditions.

Definition 2. CA A is said to be *rotation-symmetric* if its local function f satisfies the following condition.

$$\forall c, u, r, d, l \in Q, f(c, u, r, d, l) = f(c, r, d, l, u).$$

A is said to be ± 45-*degree reflection-symmetric* if its local function f satisfies the following condition.

$$\forall c, u, r, d, l \in Q, f(c, u, r, d, l) = f(c, l, d, r, u) = f(c, r, u, l, d).$$

A is said to be *permutation-symmetric* if A is rotation-symmetric and its local function also satisfies the following condition.

$$\begin{aligned}\forall c,u,r,d,l \in Q,\\ f(c,u,r,d,l) &= f(c,u,r,l,d) = f(c,u,d,r,l) = f(c,u,d,l,r)\\ &= f(c,u,l,d,r) = f(c,u,l,r,d).\end{aligned}$$

We show a necessary and sufficient condition for a two-dimensional von Neumann NCCA with ±45-degree reflection-symmetric rules.

Theorem 1. A deterministic two-dimensional ±45-degree reflection-symmetric von Neumann neighbor CA $A = (\mathbf{Z^2}, Q, f, q)$ is number-conserving iff f satisfies

$$\begin{aligned}\exists g : Q^2 \to Q, \quad \forall c,u,r,d,l \in Q,\\ f(c,u,r,d,l) = c + g(c,u) + g(c,r) + g(c,d) + g(c,l),\\ g(c,u) = -g(u,c).\end{aligned}$$

Proof. Let's consider the configuration of Fig. 1. Blank cells are all quiescent (k). Only shadowed thirteen cells change their states and the next condition is necessary from (1).

$$\begin{aligned}&\forall (c,u,r,d,l) \in Q^5,\ c+u+r+d+l+8k\\ &= f(c,u,r,d,l) + f(u,k,k,c,k) + f(r,k,k,k,c) + f(d,c,k,k,k) + f(l,k,c,k,k)\\ &+ f(k,k,k,r,u) + f(k,r,k,k,d) + f(k,l,d,k,k) + f(k,k,u,l,k)\\ &+ f(k,k,k,u,k) + f(k,k,k,k,r) + f(k,d,k,k,k) + f(k,k,l,k,k) \qquad (2)\end{aligned}$$

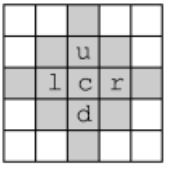


Fig. 1. A configuration for the proof of Theorem 1.

Subtracting (2) from (2)$|c = k$ yields the following.

$$\begin{aligned}k + f(c,u,r,d,l) &= c + f(k,u,r,d,l)\\ &+ \{f(u,k,k,k,k) - f(u,k,k,c,k)\} + \{f(r,k,k,k,k) - f(r,k,k,k,c)\}\\ &+ \{f(d,k,k,k,k) - f(d,c,k,k,k)\} + \{f(l,k,k,k,k) - f(l,k,c,k,k)\}. \qquad (3)\end{aligned}$$

Next, let's consider the special case of Fig. 1, i.e., the case $c = d = l = k$. In this case, only eight cells change their states and using ± 45-degree reflection-symmetric property, we obtain

$$\begin{aligned} u + r + 6k = {} & f(u,k,k,k,k) + f(r,k,k,k,k) + 2f(k,k,k,r,u) \\ & + 2f(k,k,k,u,k) + 2f(k,k,k,k,r). \end{aligned} \tag{4}$$

In the same way, we also obtain the following equations.

$$\begin{aligned} r + d + 6k = {} & f(r,k,k,k,k) + f(d,k,k,k,k) + 2f(k,r,k,k,d) \\ & + 2f(k,k,k,k,r) + 2f(k,d,k,k,k), \end{aligned} \tag{5}$$

$$\begin{aligned} d + l + 6k = {} & f(d,k,k,k,k) + f(l,k,k,k,k) + 2f(k,l,d,k,k) \\ & + 2f(k,d,k,k,k) + 2f(k,k,l,k,k), \end{aligned} \tag{6}$$

$$\begin{aligned} l + u + 6k = {} & f(l,k,k,k,k) + f(u,k,k,k,k) + 2f(k,k,u,l,k) \\ & + 2f(k,k,l,k,k) + 2f(k,k,k,u,k). \end{aligned} \tag{7}$$

Summing up (2)$|c = k$, (4), (5), (6), and (7), we obtain

$$f(k,u,r,d,l) = f(k,k,k,u,k) + f(k,k,k,k,r) + f(k,d,k,k,k) + f(k,k,l,k,k) - 3k.$$

Replacing this term in (3), we obtain

$$\begin{aligned} f(c,u,r,d,l) = c & + \{f(u,k,k,k,k) + f(k,k,k,u,k) - f(u,k,k,c,k) - k\} \\ & + \{f(r,k,k,k,k) + f(k,k,k,k,r) - f(r,k,k,k,c) - k\} \\ & + \{f(d,k,k,k,k) + f(k,d,k,k,k) - f(d,c,k,k,k) - k\} \\ & + \{f(l,k,k,k,k) + f(k,k,l,k,k) - f(l,k,c,k,k) - k\}. \end{aligned}$$

Although f is a five-ary function, the second (third, fourth and fifth, respectively) term of the right hand side only depends on two neighbors and it can be regarded as a two-ary function $f_U(c,u)$ ($f_R(c,r)$, $f_D(c,d)$ and $f_L(c,l)$, respectively).

On the other hand, ± 45-degree reflection symmetry implies that for any $a, b \in Q$,

$$f(b,a,k,k,k) = f(b,k,a,k,k) = f(b,k,k,a,k) = f(b,k,k,k,a)$$

hence also

$$f(k,a,k,k,k) = f(b,k,a,k,k) = f(b,k,k,a,k) = f(b,k,k,k,a).$$

By these conditions, $f_U(a,b)$, $f_R(a,b)$, $f_D(a,b)$, and $f_L(a,b)$ are known to be an equal function, i.e.,

$$\begin{aligned} g(a,b) \equiv f_U(a,b) = f_R(a,b) = f_D(a,b) = f_L(a,b) = \\ f(b,k,k,k,k) + f(k,k,k,b,k) - f(b,k,k,a,k) - k. \end{aligned} \tag{8}$$

Thus

$$f(c,u,r,d,l) = c + g(c,u) + g(c,r) + g(c,d) + g(c,l) \tag{9}$$

is necessary.

Next, by considering the special case, $c = b$, $u = r = d = l = k$, in Fig. 1 and the ±45-degree symmetry, we obtain

$$4k + b = f(b,k,k,k,k) + 4f(k,b,k,k,k). \tag{10}$$

By (10) and (8), $g(a,b)$ can be represented as follows.

$$g(a,b) = 3k + b - 3f(k,b,k,k,k) - f(b,a,k,k,k).$$

On the other hand, by (10) and (9)$|(c = b, u = a, r = d = l = k)$, we obtain

$$\begin{aligned} f(b,a,k,k,k) &= b + g(b,a) + 3g(b,k) \\ &= b + g(b,a) + 3\{k - f(k,b,k,k,k)\}. \end{aligned}$$

Thus

$$g(b,a) = -3k - b + 3f(k,b,k,k,k) + f(b,a,k,k,k) = -g(a,b).$$

Conversely,

$$\begin{aligned} &\sum_{x,y}(F(\alpha)(x,y) - \alpha(x,y)) \\ &= \sum_{x,y}\{g(\alpha(x,y),\alpha(x,y+1)) + g(\alpha(x,y),\alpha(x+1,y)) \\ &\quad + g(\alpha(x,y),\alpha(x,y-1)) + g(\alpha(x,y),\alpha(x-1,y))\}. \end{aligned}$$

Because $g(\alpha(x,y),\alpha(x,y+1)) = -g(\alpha(x,y+1),\alpha(x,y))$ and $g(\alpha(x,y),\alpha(x+1,y)) = -g(\alpha(x+1,y),\alpha(x,y))$, all terms in the summation are canceled to 0. □

By Theorem 1, to construct NCCA, it is sufficient to design a two-ary function g (say *flow function*) in (2). We have the following proposition.

Proposition 1. A deterministic 2-dimensional ±45-degree reflection-symmetric von Neumann neighbor NCCA A defined in Theorem 1 is permutation-symmetric.

Proof.

$$\begin{aligned} f(c,u,r,d,l) &= c + g(c,u) + g(c,r) + g(c,d) + g(c,l), \\ f(c,u,r,l,d) &= c + g(c,u) + g(c,r) + g(c,l) + g(c,d), \end{aligned}$$

thus $f(c,u,r,d,l) = f(c,u,r,l,d)$ and all the other conditions can be proved in the same way. □

Although the permutation-symmetric condition should be a strong constraint and an NCCA with a single flow function seems to have little ability, we show two examples which realize logical universality and self-reproducing ability in the next section.

3 Examples of NCCA

3.1 A Logically Universal NCCA

The Wireworld model [2] is a logically universal CA, i.e., you can embed any logic circuits into the space as a configuration and its transition process can simulates the circuit. In this section, we show that such Wireworld like model can be embedded into NCCA with a flow function. The original Wireworld model is a Moore neighbor CA, and we employ a modified model with von Neumann neighbor by Tojima [11].

$A_{NC19} = (\mathbf{Z}^2, \mathbf{N}_{[0,19]}, f_{NC19}, 0)$. The flow function f of f_{NC19} has the following values and satisfies $f(a,b) = -f(b,a)$ for all $a, b \in \mathbf{N}_{[0,19]}$. Values not defined by these conditions are 0.

$$
\begin{array}{llll}
f(1,10)=1, & f(3,8)=3, & f(3,12)=3, & f(3,13)=3, \\
f(3,15)=1, & f(4,11)=2, & f(6,7)=1, & f(6,10)=1, \\
f(6,11)=4, & f(6,12)=1, & f(8,7)=1, & f(8,10)=1, \\
f(8,11)=1, & f(8,13)=1, & f(8,18)=1, & f(8,19)=1, \\
f(9,7)=3, & f(9,10)=1, & f(9,12)=1, & f(9,15)=1, \\
f(9,19)=1, & f(11,10)=1, & f(16,10)=1, & f(17,10)=1.
\end{array}
$$

Tojima's model has configurations of a signal and wires. Wires are allowed to have branches (side roads) and cross points. A branching point and a cross point have functions depicted in Fig. 2 and combining these functions, it is possible to realize AND, OR, NOT gates and signal crossing elements [11].

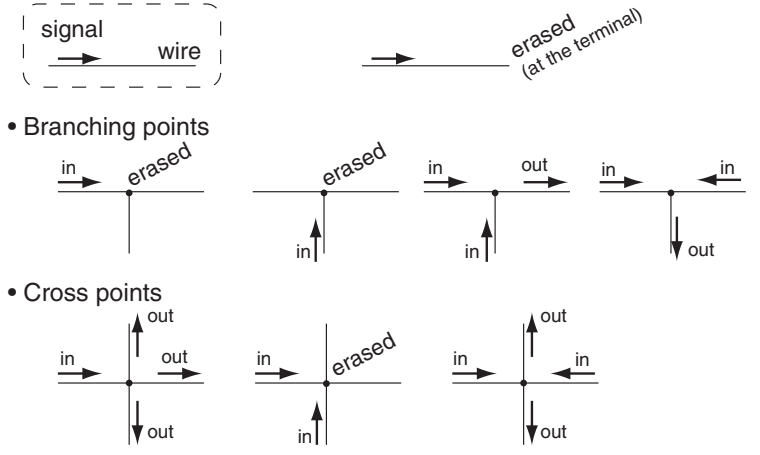


Fig. 2. Functions of a branching point and a cross point.

A wire is a connected cells of width 1 whose state is 9 and it can have turning points and a signal is encoded by two cells whose states are 8 and 10 (Fig. 3). A signal flows along a wire with speed 1, and at the end of the wire (i.e., the signal faced to a three quiescent cells), the signal is terminated. Only two values of the flow function ($f(8,10) = 1$ and $f(9,10) = 1$) realize the signal.

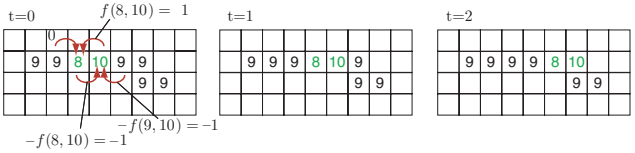


Fig. 3. A configuration of a wire and a signal.

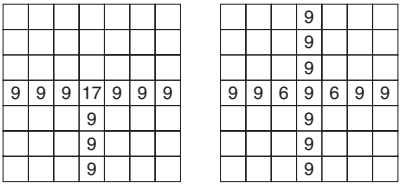


Fig. 4. A configuration of a branching point and a cross point.

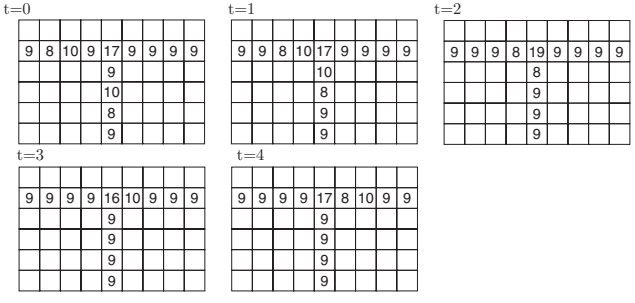


Fig. 5. An example of a branch (the case of two input signals and one output signal).

Fig. 4 shows a branching point and a cross point. A branching cell has the state 17 and a cross point has two neighboring cells whose states are 6. Fig. 5 shows an example of the case of two-input signals.

A branch point can be regarded as an AND gate. Configurations of an OR gate, a NOT gate, and a crossing element are shown in Fig. 6, Fig. 7, and Fig. 8 respectively.

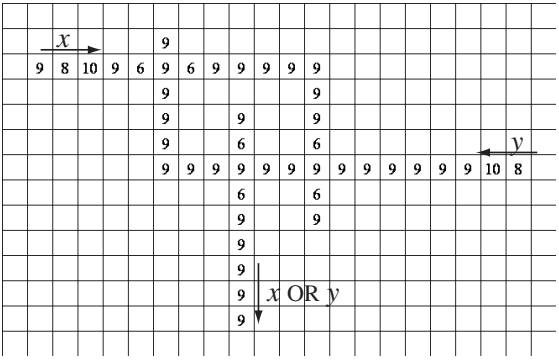


Fig. 6. A configuration of an OR gate.

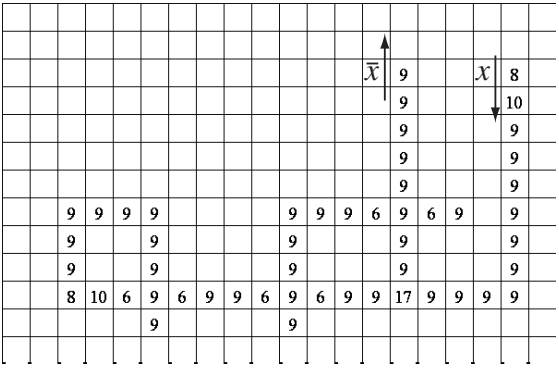


Fig. 7. A configuration of a NOT gate.

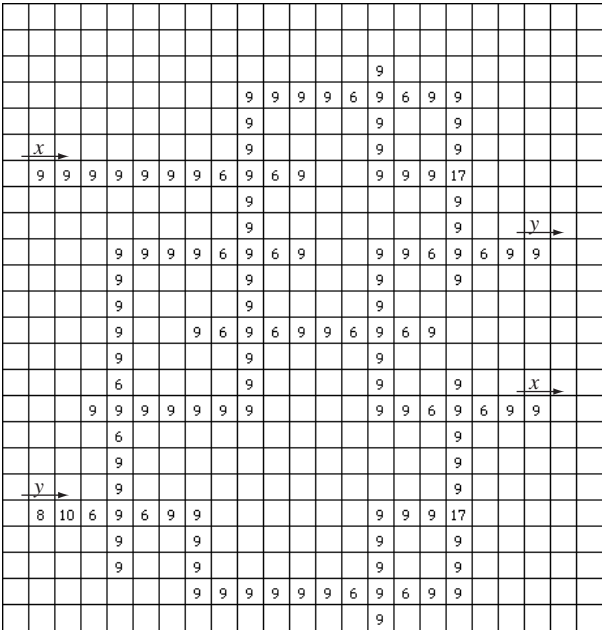


Fig. 8. A configuration of a crossing element.

3.2 A Self-Reproducing NCCA

The next example is an embedding of the Langton's self-reproducing Loop [6]. The Langton's Loop is an 8-state von Neumann neighbor CA and an initial mother Loop self-reproduce herself (Fig 9).

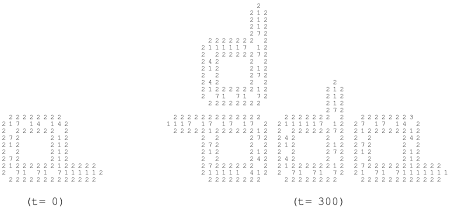


Fig. 9. The Langton's self-reproducing Loop.

We constructed a CA A_{NCSR} which embeds Langton's Loop.
$A_{NCSR} = (\mathbf{Z}^2, \mathbf{N}_{[0,395]}, f_{NCSR}, 235)$.

The flow function f' of f_{NCSR} has the following values and satisfies $f(a,b) = -f(b,a)$ for all $a, b \in \mathbf{N}_{[0,395]}$. Values not defined by these conditions are 0.

$f'(0,207) = -60$, $f'(151,195) = -10$, $f'(185,205) = 10$, $f'(200,288) = 35$,
$f'(0,215) = 20$, $f'(151,289) = 54$, $f'(189,199) = -40$, $f'(200,289) = 35$,
$f'(0,267) = 40$, $f'(154,235) = 161$, $f'(190,210) = 18$, $f'(202,208) = 37$,
$f'(0,355) = 80$, $f'(154,345) = 30$, $f'(191,196) = -17$, $f'(202,285) = 37$,
$f'(13,275) = 172$, $f'(158,200) = 37$, $f'(191,198) = 9$, $f'(202,315) = -47$,
$f'(40,267) = 275$, $f'(158,267) = 37$, $f'(192,195) = -1$, $f'(203,207) = -8$,
$f'(64,195) = 35$, $f'(158,275) = 37$, $f'(193,195) = -6$, $f'(203,345) = -1$,
$f'(64,235) = 35$, $f'(160,195) = -12$, $f'(193,197) = -28$, $f'(203,355) = -40$,
$f'(64,355) = 120$, $f'(160,289) = 35$, $f'(193,315) = 76$, $f'(208,214) = -55$,
$f'(74,235) = 27$, $f'(163,200) = 32$, $f'(195,201) = -44$, $f'(209,375) = 30$,
$f'(74,345) = 15$, $f'(163,267) = 32$, $f'(195,202) = 11$, $f'(214,285) = 55$,
$f'(139,211) = -24$, $f'(163,275) = 32$, $f'(195,205) = 10$, $f'(214,315) = -74$,
$f'(139,315) = 120$, $f'(170,208) = -27$, $f'(195,209) = 44$, $f'(219,235) = 35$,
$f'(141,206) = 3$, $f'(170,285) = -27$, $f'(195,254) = 10$, $f'(254,275) = -50$,
$f'(141,235) = 171$, $f'(170,315) = 121$, $f'(195,267) = 59$, $f'(267,355) = 302$,
$f'(141,355) = 40$, $f'(171,144) = -171$, $f'(197,205) = 10$, $f'(275,288) = 40$,
$f'(143,189) = -9$, $f'(173,200) = 22$, $f'(198,254) = -3$, $f'(275,289) = 35$,
$f'(143,213) = 10$, $f'(180,205) = 10$, $f'(198,345) = 16$, $f'(275,315) = 40$,
$f'(143,235) = 171$, $f'(181,275) = -40$, $f'(198,355) = -40$, $f'(275,355) = 40$,
$f'(143,355) = 40$, $f'(183,235) = 16$, $f'(199,213) = 40$, $f'(275,395) = 40$,
$f'(144,235) = 171$, $f'(184,275) = -40$, $f'(199,219) = 36$, $f'(285,315) = 30$,
$f'(144,355) = 40$, $f'(184,285) = -30$, $f'(199,275) = -40$, $f'(285,345) = 30$,
$f'(149,200) = 46$, $f'(185,190) = 10$, $f'(200,253) = 57$, $f'(285,375) = 30$,
$f'(151,185) = -10$.

Fig. 10 is a initial mother Loop of A_{NCSR}. Blank cell has a state 235. Fig. 11 is a configuration at $t = 154$.

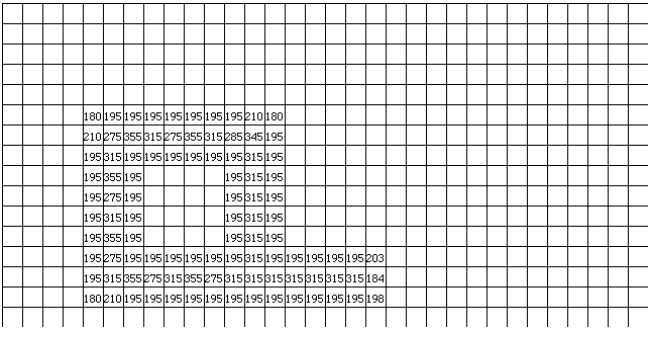


Fig. 10. An initial configuration of A_{NCSR}.

Several examples of transition processes of A_{NC19} and A_{NCSR} can be seen as QuickTime movies at the following WWW site.

`http://www.iec.hiroshima-u.ac.jp/projects/ncca/pm45/`

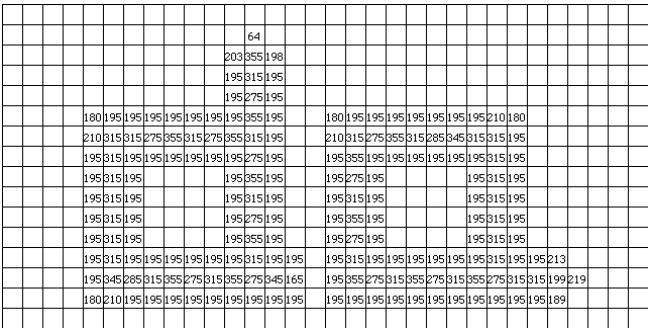


Fig. 11. A configuration of A_{NCSR} at $t = 154$.

4 Conclusion

In this paper, we show a necessary and sufficient condition for a two-dimensional von Neumann neighbor NCCA with a special symmetric rule and, employing the condition, we construct a logically universal NCCA and a self-reproducing NCCA.

It might be possible to reduce their number of states. It might also be possible to find a logically universal NCCA under stronger constraints, for example, a flow function $f(x, y)$ is only depends on the difference of x and y, and so on.

Acknowledgements. The authors thank Ferdinand Peper, Jia Lee, and Susumu Adachi at Communication Research Laboratory for their useful comments. The authors also thank anonymous reviewers whose comments pointed out many improvements particulary the form of Theorem 1.

References

1. Boccara, N. and Fukś, H.: Number-conserving cellular automaton rules, *Fundamenta Informaticae* (to Appear).
2. Dewdney, A. K.: Computer recreations: the cellular automata programs that create wireworld, rugworld and other diversions, *Scientific American* Jan. (1990) 136–139.

3. Durand, B., Formenti, E. and Roka, Z.: Number conserving cellular automata: from decidability to dynamics, nlin.CG/0102035, (2001) (http://arxiv.org/list/nlin.CG/0102).
4. Frisch, U., Hasslacher, B. and Pomeau, Y.: Lattice-Gas Automata for the Navier-Stokes equation, *Physical Review Letters* **56** (1986) 1505–1508.
5. Hattori, T. and Takesue, S.: Additive conserved quantities in discrete-time lattice dynamical systems, *Physica* **49D** (1991) 295-322.
6. Langton, C. G.: Self-reproduction in cellular automata, *Physica***10D**(1984) 135-144.
7. Margolus, N.: Physics-like models of computation, *Physica*, **10D** (1984) 81–95.
8. Morita, K. and Harao, M.: Computation universality of one-dimensional reversible (injective) cellular automata, *Trans. IEICE Japan* **E72** (1989) 758-762.
9. Morita, K. and Imai, K.: Number-conserving reversible cellular automata and their computation-universality, *Proceedings of MFCS'98 Workshop on Cellular Automata*, Brno (1998) 51-68.
10. Nagel, K. and Schreckenberg, M.: A cellular automaton model for freeway traffic, *Journal of Physics* I, 2 (1992) 2221–2229.
11. Tojima, Y.: Logic circuit composition on a two-dimensional three-state cellular automaton, Bachelor thesis, Hiroshima University (1997) (in Japanese).
12. Washio, T.: On self-reproduction in number-conserving reversible cellular automata, Bachelor thesis, Hiroshima University (1999) (in Japanese).

Generation of Diophantine Sets by Computing P Systems with External Output

Álvaro Romero Jiménez and Mario J. Pérez Jiménez

Dpto. de Ciencias de la Computacion e Inteligencia Artificial
Universidad de Sevilla, Spain
{Alvaro.Romero,Mario.Perez}@cs.us.es

Abstract. In this paper a variant of P systems with external output designed to compute functions on natural numbers is presented. These P systems are stable under composition and iteration of functions. We prove that every diophantine set can be generated by such P systems; then, the universality of this model can be deduced from the theorem by Matiyasevich, Robinson, Davis and Putnam in which they establish that every recursively enumerable set is a diophantine set.

1 Introduction

In 1998 G. Păun initiated a new branch of the field of *Natural Computing* by introducing a new model of molecular computation, based on the structure and functioning of the living cell: *transition P systems* (see [3]). The framework within which computation are performed in this model is the membrane structure, which recreates the cell-like one. Multisets of symbol-objects are processed along the computations, making them to evolve and distributing them among the membranes. The result of a halting computation is the number of objects collected in a specified output membrane.

Since the introduction of this model of computation many variants of it have been proposed. One of them, presented in [5] by G. Păun, G. Rozenberg and A. Salomaa, is the model of *transition P systems with external output*. In this model, the result of a halting computation is not collected in a fixed membrane of the membrane structure, but in the external environment associated with it. In this way, the output of a computation can be thought as a set of strings, instead of as a natural number, as occurred in the basic model.

P systems are usually considered as devices which generate numbers. Nevertheless, besides generating devices, they can also be thought as recognizing devices and as computing devices. These kind of P systems have been studied in [6].

In this paper we work with computing P systems, but instead of the basic transition ones we consider those with external output. Thanks to the special functioning of these apparatus, we have been able to define, in a comfortable manner, several operations between computing P systems with external output; more specifically, we have defined composition and iteration, what have allowed us to prove the universality of these devices through the generation of all the diophantine sets.

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 176–190, 2002.

2 Multisets. Membrane Structures. Evolution Rules

A *multiset* over a set, A, is an application $m : A \to \mathbb{N}$. A multiset is said to be empty (resp. finite) if its support, $supp(m) = \{a \in A : m(a) > 0\}$, is empty (resp. finite). If m is a finite multiset over A, we will denote it $m = \{\{a_1, \ldots, a_k\}\}$, where the elements a_i are possibly repeated. We write $M(A)$ for the set of all the multisets over A.

The set of *membrane structures*, MS, is defined by recursion as follows: 1. $[\,] \in MS$; 2. If $\mu_1, \ldots, \mu_n \in MS$, then $[\mu_1 \ldots \mu_n] \in MS$.

A membrane structure, μ, can also be seen as a rooted tree, $\big(V(\mu), E(\mu)\big)$. Then, the nodes of this tree are called *membranes*, the root node the *skin membrane* and the leaves *elementary membranes* of the membrane structure. The *degree* of a membrane structure is the number of membranes in it.

The concepts of *depth* of a membrane structure and *depth* of its membranes are easily defined from those of a tree and its nodes. We will also need the notion of *level* of a membrane within a membrane structure, which is defined as the difference between the depth of the second and the depth of the first.

The *membrane structure with external environment* associated with a membrane structure, μ, is $\mu^E = [_E \mu]_E$. If we consider the latter as a rooted tree, the root node is called the *external environment* of μ.

Given an alphabet, Γ, we associate with every membrane of a membrane structure a finite multiset of elements of Γ, which are called the *objects* of the membrane.

We also associate with every one of these membranes a finite set of *evolution rules*. A evolution rule over Γ is a pair (u, v), usually written $u \to v$, where u is a string over Γ and $v = v'$ or $v = v'\delta$, where v' is a string over

$$\Gamma \times \big(\{here, out\} \cup \{in_l : l \in V(\mu)\}\big)$$

The idea behind a rule is that the objects in u "evolve" into the objects in v', moving or not to another membrane and possibly dissolving the original one.

3 Computing P Systems with External Output

We are now prepared to introduce our new model of computation.

Definition 3.1. *A computing P system with external output of order* (m, n) *and degree* p *is a tuple*

$$\Pi = \big(\Sigma, \Lambda, \Gamma, \#, \mu_\Pi, \iota, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p)\big)$$

where

- Σ *is an ordered alphabet of size* m*, the input alphabet.*
- Λ *is an ordered alphabet of size* n*, the output alphabet.*
- Γ *is an alphabet such that* $\Sigma \cup \Lambda \subset \Gamma$*, the working alphabet.*
- $\#$ *is a distinguished element in* $\Gamma \setminus \big(\Sigma \cup \Lambda\big)$*, the halting element.*

- μ_Π *is a membrane structure of degree* p*, whose membranes we suppose labeled from 1 to* p*.*
- *The input membrane of* Π *is labeled by* $\iota \in \{1, \ldots, p\}$*.*
- $\mathcal{M}_i$ *is a multiset over* $\Gamma \setminus \Sigma$ *associated with the membrane labeled by* i*, for every* $i = 1, \ldots, p$*.*
- R_i *is a finite set of evolution rules over* Γ *associated with the membrane labeled by* i*, and* ρ_i *is a strict partial order over it, for every* $i = 1, \ldots, p$*.*

To formalize the semantics of this model we define first what a configuration of a P system is, from what follows the notion of computation.

Definition 3.2. *Let* Π *be a computing P system with external output.*

- *A configuration of* Π *is a pair* (μ^E, M)*, where* μ *is a membrane structure such that* $V(\mu) \subseteq V(\mu_\Pi)$ *and has the same root than* μ_Π*, and* M *is an application from* $V(\mu^E)$ *into* $M(\Gamma)$*. For every node* $nd \in V(\mu^E)$ *we denote* $M_{nd} = M(nd)$*.*
- *Suppose that* Π *is of order* (m, n) *and* $\Sigma = (a_1, \ldots, a_m)$*. Then, any* m*-tuple of natural numbers can be codified by a multiset over* Σ *and given as input to the P system. Thus, the initial configuration of* Π *for a tuple* $(k_1, \ldots, k_m) \in \mathbb{N}^m$ *is the pair* (μ^E, M)*, where* $\mu = \mu_\Pi$*,* $M_E = \emptyset$*,* $M_\iota = \mathcal{M}_\iota \cup \{\{a_1^{k_1} \ldots a_m^{k_m}\}\}$ *and* $M_i = \mathcal{M}_i$*, for every* $i \neq \iota$*.*

We can pass, in a non-deterministic manner, from one configuration of Π to another by applying to its multisets the evolution rules associated with their corresponding membranes. This is done as follows: given a rule $u \to v$ of a membrane i, the objects in u are removed from M_i; then, for every $(ob, out) \in v$ an object ob is put into the multiset associated with the parent membrane (or the external environment if i is the skin membrane); for every $(ob, here) \in v$ an object ob is added to M_i; for every $(ob, in_j) \in v$ an object ob is added to M_j (if j is not a children membrane of i, the rule cannot be applied). Finally, if $\delta \in v$, then the membrane i is dissolved, that is, it is removed from the membrane structure (the objects associated with this membranes are collected by the parent membrane, and the rules are lost. The skin membrane cannot dissolve). Moreover, the *priority relation* among the rules forbids the application of a rule if another one of higher priority is applied.

Given two configurations, C and C', of Π, we say that C' is obtained from C in one transition step, and we write $C \Rightarrow C'$, if we can pass from the first to the second by using the evolutions rules appearing in the membrane structure of C in a parallel and maximal way, and for all the membranes at the same time.

Definition 3.3. *Given a computing P system with external output of order* (m, n)*,* Π*, a computation of* Π *with input* $(k_1, \ldots, k_m) \in \mathbb{N}^m$ *is a sequence, possibly infinite, of configurations of* Π*,* $C_0 \Rightarrow C_1 \Rightarrow \ldots \Rightarrow C_q$*,* $q \geq 0$*, such that*

- C_0 *is the initial configuration of* Π *for* $(k_1, \ldots, k_m)$*.*
- *Each* C_i *is obtained from the previous configuration by one transition step.*

We say that a computation, $\mathcal{C}$*, is a halting computation of* Π*, if* $q \in \mathbb{N}$ *and there is no rule applicable to the objects present in its last configuration.*

Then, the output of a halting computation and of a computing P system with external output can be defined in a natural way.

Definition 3.4. *Let Π be a computing P system with external output of order (m,n) and suppose that $\Lambda = (b_1,\ldots,b_n)$. Let $\mathcal{C}$ be a halting computation of Π with input $(k_1,\ldots,k_m) \in \mathbb{N}^m$ and (μ^E, M) its last configuration. Then, the output of that computation is given by*

$$Output(\mathcal{C}) = \big(M_E(b_1),\ldots,M_E(b_n)\big).$$

Definition 3.5. *Let Π be a computing P system with external output of order (m,n). The output of Π with input $(k_1,\ldots,k_m) \in \mathbb{N}^m$ is given by*

$$Output(\Pi; k_1,\ldots,k_m) = \{Output(\mathcal{C}) : \mathcal{C} \text{ is a halting computation of } \Pi \text{ with input } (k_1,\ldots,k_m)\}.$$

The idea behind P systems with external output is that we cannot know what is happening inside the membrane structure, but we can only collect the information thrown from it to the external environment. In accordance with it, it seems natural that the halting computations of these P systems report to the outside when they have reached their final configurations.

Furthermore, the idea behind computing P systems is to use them as computing models of functions between natural numbers. These considerations lead us to the following notions:

Definition 3.6. *A computing P system with external output of order (m,n), Π, is said to be valid when the following is verified:*

- *If $\mathcal{C}$ is a halting computation of Π, then a rule of the form $u \to v(\#, out)$ must have been applied in the skin membrane of μ_Π, and only in the last step of the computation.*
- *If $\mathcal{C}$ is not a halting computation of Π, then no rule of the previous form is applied in the skin membrane in any step of the computation.*
- *For every $(k_1,\ldots,k_m) \in \mathbb{N}^m$ and for every two halting computations, $\mathcal{C}_1$ and $\mathcal{C}_2$, of Π with input $(k_1,\ldots,k_m)$, $Output(\mathcal{C}_1) = Output(\mathcal{C}_2)$.*

Definition 3.7. *A computing P system with external output of order (m,n), Π, computes a partial function, $f : \mathbb{N}^m \dashrightarrow \mathbb{N}^n$, if*

- *Π is a valid P system.*
- *For every $(k_1,\ldots,k_m) \in \mathbb{N}^m$*
  - *f is defined over $(k_1,\ldots,k_m)$ if and only if there exists a halting computation of Π with input $(k_1,\ldots,k_m)$.*
  - *If $\mathcal{C}$ is a halting computation of Π with input $(k_1,\ldots,k_m)$, then $Output(\mathcal{C}) = f(k_1,\ldots,k_m)$.*

We denote $CEP_p^{m,n}(\alpha,\beta,\gamma)$, where $m,n \in \mathbb{N}, p \geq 1$, $\alpha \in \{Pri, nPri\}$, $\beta \in \{Coo, Cat, nCoo\}$ and $\gamma \in \{\delta, n\delta\}$, the family of functions computed by computing P systems with external output of order (m,n), of degree at most p, and with or without priority, with cooperation, only catalysts or without cooperation (see [3]), and with or without dissolution, respectively. The union, for all $p \geq 1$, of the families of one of these types is denoted $CEP^{m,n}(\alpha,\beta,\gamma)$.

4 Composition of Computing P Systems with External Output

We introduce now the operation of composition between computing P systems with external output.

Definition 4.1. *Let* $f : \mathbb{N}^m - \to \mathbb{N}^n$ *and* $g_1 : \mathbb{N}^r - \to \mathbb{N}^{s_1}, \ldots, g_t : \mathbb{N}^r - \to \mathbb{N}^{s_t}$ *such that* $s_1 + \cdots + s_t = m$*. Then, the composition of* f *with* g_1 *to* g_t*, denoted* $C(f; g_1, \ldots, g_t)$*, is a partial function from* $\mathbb{N}^r$ *to* $\mathbb{N}^n$ *defined as follows*

$$C(f; g_1, \ldots, g_t)(k_1, \ldots, k_r) = f(g_1(k_1, \ldots, k_r), \ldots, g_t(k_1, \ldots, k_r))$$

Theorem 4.2. *Let* $f \in CEP^{m,n}(\alpha, \beta, \gamma), g_1 \in CEP^{r,s_1}(\alpha, \beta, \gamma), \ldots, g_t \in CEP^{r,s_t}(\alpha, \beta, \gamma)$*, with* $\alpha \in \{Pri, nPri\}$*,* $\beta \in \{Coo, Cat, nCoo\}$ *and* $\gamma \in \{\delta, n\delta\}$*. Then,* $C(f; g_1, \ldots, g_t) \in CEP^{r,n}(Pri, Coo, \gamma)$*.*

Proof. Let

$$\begin{aligned}
\Pi_f &= \left(\Sigma_f, \Lambda_f, \Gamma_f, \#_f, \mu_{\Pi_f}, \iota_f, \mathcal{M}_1^f, \ldots, \mathcal{M}_{p_f}^f, (R_1^f, \rho_1^f), \ldots, (R_{p_f}^f, \rho_{p_f}^f)\right)\\
\Pi_{g_1} &= \left(\Sigma_{g_1}, \Lambda_{g_1}, \Gamma_{g_1}, \#_{g_1}, \mu_{\Pi_{g_1}}, \iota_{g_1}, \mathcal{M}_1^{g_1}, \ldots, \mathcal{M}_{p_{g_1}}^{g_1}, (R_1^{g_1}, \rho_1^{g_1}), \ldots, (R_{p_{g_1}}^{g_1}, \rho_{p_{g_1}}^{g_1})\right)\\
&\vdots\\
\Pi_{g_t} &= \left(\Sigma_{g_t}, \Lambda_{g_t}, \Gamma_{g_t}, \#_{g_t}, \mu_{\Pi_{g_t}}, \iota_{g_t}, \mathcal{M}_1^{g_t}, \ldots, \mathcal{M}_{p_{g_t}}^{g_t}, (R_1^{g_t}, \rho_1^{g_t}), \ldots, (R_{p_{g_t}}^{g_t}, \rho_{p_{g_t}}^{g_t})\right)
\end{aligned}$$

be computing P systems with external output that compute, respectively, the function f and the functions g_1 to g_t.

By means of a renaming of the elements of the alphabets (and, therefore, also of the rules), we can suppose that

- $\Sigma_{g_1} = \cdots = \Sigma_{g_t} = (a_1, \ldots, a_r)$.
- $\Lambda_{g_1} = (b_1, \ldots, b_{s_1}), \ldots, \Lambda_{g_t} = (b_{s_1+\cdots+s_{t-1}+1}, \ldots, b_m)$.
- $\Sigma_f = (c_1, \ldots, c_m)$.
- $\Lambda_f = (d_1, \ldots, d_n)$.
- $\left(\Lambda_{g_1} \cup \cdots \cup \Lambda_{g_t}\right) \cap \Gamma_f = \emptyset$.
- $\#_{g_i} \neq \#_{g_j}$, for every $i \neq j$.

Let us consider the computing P system with external output

$$\Pi = \left(\Sigma, \Lambda, \Gamma, \#, \mu_\Pi, \iota, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p)\right)$$

given by

- $\Sigma = (e_1, \ldots, e_r)$. (We suppose that $\Sigma \cap \bigcup_{i=1}^t \Gamma_{g_i} = \emptyset$).
- There exist distinguished elements $\oplus, \ominus, \odot \in \Gamma \setminus (\Gamma_f \cup \bigcup_{i=1}^t \Gamma_{g_i})$.
- $\Lambda = (d_1, \ldots, d_n)$.
- $\# \neq \#_{g_i}$, for every $i = 1, \ldots, t$, and $\# \neq \#_f$.

- $\mu_{\Pi} = [_1 \mu_{\Pi_{g_1}} \dots \mu_{\Pi_{g_t}} \mu_{\Pi_f}]_1$, where the membranes from $\mu_{\Pi_{g_1}}, \dots, \mu_{\Pi_{g_t}}, \mu_{\Pi_f}$ have been adequately renamed (and therefore, also the rules of the corresponding P systems have been adapted). We denote $\sigma_{g_1}, \dots, \sigma_{g_t}, \sigma_f$ the skin membranes of the latter. Also, we consider that $\iota_{g_1}, \dots, \iota_{g_t}, \iota_f$ reflect the new labeling of the input membranes of $\Pi_{g_1}, \dots, \Pi_{g_t}, \Pi_f$, respectively.
- $\iota = 1$.
- $p = p_{g_1} + \dots + p_{g_t} + p_f + 1$.
- $\mathcal{M}_1 = \{\{\#, \ominus\}\}$. The remaining multisets are all empty.
- The evolution rules are the following:
  - Evolution rules for membrane 1:

$$e_i \to (e_i, in_{\sigma_{g_1}}) \dots (e_i, in_{\sigma_{g_t}}) \quad (i = 1, \dots, r)$$
$$\ominus \to (\ominus, in_{\sigma_{g_1}}) \dots (\ominus, in_{\sigma_{g_t}})$$
$$\#_{g_1} \dots \#_{g_t} \# \to (\ominus, in_{\sigma_f}) > \# \to \# > b_i \to (b_i, in_{\sigma_f}) \quad (i = 1, \dots, m)$$
$$d_i \to (d_i, out) \quad (i = 1, \dots, n)$$
$$\#_f \to (\#, out)$$

  - For every function $fun = g_1, \dots, g_t, f$ and for every membrane j of $\mu_{\Pi_{fun}}$, the following rules are included:

$$\ominus \to \oplus(\ominus, in_{j_1}) \dots (\ominus, in_{j_k})$$
$$\odot^u \oplus \to \mathcal{M}_j^{fun} > \oplus \to \oplus \odot$$
evolution rules associated with the membrane in Π_{fun}

    where j_1 to j_k are the children membranes of membrane j and u is its level within $\mu_{\Pi_{fun}}$. Moreover, if j is ι_{fun}, then the rule $\oplus \to \oplus\odot$ has higher priority than the original rules of Π_{fun} for this membrane.
  - Let fun be as above and let $j_1, \dots, j_q$ be the membrane path from σ_{fun} to ι_{fun}. Then, for $k = 1, \dots, q-1$ the following rules are included in membrane j_k:

$$e_i \to (e_i, in_{j_{k+1}}) \quad (i = 1, \dots, r) \qquad \text{for } fun = g_1, \dots, g_t$$
$$b_i \to (b_i, in_{j_{k+1}}) \quad (i = 1, \dots, m) \qquad \text{for } fun = f$$

    Also, the following rules are included in membrane $j_q = \iota_{fun}$.

$$e_i \to a_i \quad (i = 1, \dots, r) \qquad \text{for } fun = g_1, \dots, g_t$$
$$b_i \to c_i \quad (i = 1, \dots, m) \qquad \text{for } fun = f$$

The P system constructed in this way, denoted $C(\Pi_f; \Pi_{g_1}, \dots, \Pi_{g_t})$, is a valid computing P system with external output which computes the composition of f with g_1 to g_t. Furthermore, it preserves the use or not of dissolution from the P systems which compute the functions.

Indeed, the system works as follows:

– Phase 1: *Computation of the functions g_1 to g_t over the input data*
To perform this stage, we need to carry out two operations: the first one consists of taking the input arguments from membrane 1, which recall is the input membrane of Π, to all the input membranes of the P systems Π_{g_1} to Π_{g_t}. This is easily done by displacing the objects representing the arguments through all the necessary membranes.
The second operation is a little bit more complicated: in order for a specific P system Π_{g_j} to compute correctly the value of the function g_j over the input data, we need that all the membranes of this P system start to apply their original rules *at the same time* (that is, we have to synchronize locally the membranes of each Π_{g_j}). We achieve this by using counters for every one of these membranes. First, we use the object $\ominus$ to activate the counters, represented by objects $\oplus$, in all the membranes. These latter objects use objects $\odot$ to count and, when a certain quantity is reached, the corresponding membrane is allowed to use the rules of Π_{g_j}. Because of the way we have implemented this, these quantities turn out to be the levels of the membranes in the structure $\mu_{\Pi_{g_j}}$.
It is also important that when the P system Π_{g_j} starts to compute the value, the objects representing the input data have reached its input membrane. However, as we perform the two operations above simultaneously, we get it for free.
Finally, we have to wait until all the values from Π_{g_1} to Π_{g_t} have been computed, before allowing the P system Π_f to be used (that is, there must be a global synchronization in the skin of Π).
Let us see with greater detail the rules involved in this phase:

1. At the first step of a computation of Π with input $(k_1, \ldots, k_r)$, we have in membrane 1 the multiset $\{\{e_1^{k_1}, \ldots, e_r^{k_r}, \#, \ominus\}\}$ and the other membranes are empty. Therefore, only the rules which send the objects e_i and the object $\ominus$ into the skins of $\mu_{\Pi_{g_1}}$ to $\mu_{\Pi_{g_t}}$ and the rule $\# \to \#$ in membrane 1 can be applied.
2. Now, membrane 1 waits for the values of g_1 to g_t over $(k_1, \ldots, k_r)$ by means of the rule $\# \to \#$. With regard to membrane structures $\mu_{\Pi_{g_1}}$ to $\mu_{\Pi_{g_t}}$, the rule $\ominus \to \oplus(\ominus, in_{j_1}) \ldots (\ominus, in_{j_k})$ makes the object $\ominus$ to spread to all their membranes, because when it reaches a particular membrane, it is immediately transformed into a counter object $\oplus$ and also sent to the children membranes. Thus, from a step of the computation to the next one, $\ominus$ reaches the membranes one depth greater. Meanwhile, the rule $\oplus \to \oplus\odot$ makes the object $\oplus$ to generate objects $\odot$. A close look to the situation created shows that the activating object $\ominus$ have reached all the membranes exactly when the counter objects $\oplus$ have generated in each membrane a number of objects $\odot$ equal to their levels in $\mu_{\Pi_{fun}}$ $(fun = g_1, \ldots, g_t)$. At that moment, the rule $\odot^u \oplus \to \mathcal{M}_j^{fun}$ introduces in membrane j the objects associated with it in Π_{fun}, and this is done for all the membranes of each Π_{fun} at the same time. From now on, the values of g_1 to g_t over $(k_1, \ldots, k_r)$ are computed exactly in the same way than the P systems Π_{g_1} to Π_{g_t} would do it.

3. Simultaneously, the objects e_i cover the path from the skin membrane of each $\mu_{\Pi_{g_j}}$ to the input membrane of Π_{g_j}, by means of the rules $e_i \to (e_i, in_{j_{k+1}})$, and are changed there into the corresponding objects a_i, by means of the rules $e_i \to a_i$. Note that the objects e_i and the object $\ominus$ reach the input membrane at the same time. So, when Π_{g_j} starts its original functioning, as stated above, the input data is in its place.

– Phase 2: *Computation of the function f*

Phase 1 ends when membrane 1 has collected at least one object of each $\#_{g_1}$ to $\#_{g_t}$. It is then when the values computed have to be sent as input data to the P system Π_f. To synchronize the end of phase 1 with the beginning of phase 2, membrane 1 apply once and again the rule $\# \to \#$ until the rule $\#_{g_1} \dots \#_{g_t} \# \to (\ominus, in_{\sigma_f})$ can be used.

This latter rule sends an object $\ominus$ into the skin of μ_{Π_f}, in order to initiate its membranes' counters so that they start to apply their original rules at the same time (local synchronization within Π_f). This is done just as before. Also, in the next step of the computation the objects b_i, which represent the values obtained in phase 1, are put into the skin of μ_{Π_f} and, subsequently, moved, by means of the rules $b_i \to (b_i, in_{j_{k+1}})$, through all the membranes from this one to the input membrane of Π_f. Next, the rules $b_i \to c_i$ change them into the corresponding input objects of Π_f.

It is easy to see that, although there is a gap of one step of computation between when $\ominus$ gets into a membrane and when the b_is do so, this is not at all a problem.

Now, the value of the function f over the arguments represented by the objects c_i is computed, and along this computation objects d_i representing the result are thrown out of μ_{Π_f}. These objects are collected in membrane 1, and immediately expelled from μ_Π. The calculation finishes when some objects $\#_f$ are collected in membrane 1 and they are expelled from μ_Π as objects $\#$.

□

5 Iteration of Computing P Systems with External Output

We introduce now the operation of iterating a computing P system with external output.

Definition 5.1. *Let $f : \mathbb{N}^m - \to \mathbb{N}^m$. Then, the iteration function of f, denoted $It(f)$, is a partial function from $\mathbb{N}^{m+1}$ to $\mathbb{N}^m$ defined as follows:*

$$It(f)(\boldsymbol{x}, 0) = \boldsymbol{x}$$
$$It(f)(\boldsymbol{x}, n+1) = It(f)(f(\boldsymbol{x}), n)$$

Theorem 5.2. *Let $f \in CEP^{m,m}(\alpha, \beta, n\delta)$, with $\alpha \in \{Pri, nPri\}$ and $\beta \in \{Coo, Cat, nCoo\}$. Then $It(f) \in CEP^{m+1,m}(Pri, Coo, n\delta)$.*

Proof. Let

$$\Pi_f = \left(\Sigma_f, \Lambda_f, \Gamma_f, \#_f, \mu_{\Pi_f}, \iota_f, \mathcal{M}_1^f, \ldots, \mathcal{M}_{p_f}^f, (R_1^f, \rho_1^f), \ldots, (R_{p_f}^f, \rho_{p_f}^f)\right)$$

be a computing P system with external output such that computes f.

By means of a renaming of the elements of the alphabets (and, therefore, also of the rules), we can suppose that

- $\Sigma_f = (a_1, \ldots, a_m)$.
- $\Lambda_f = (b_1, \ldots, b_m)$.

Let us consider the computing P system with external output

$$\Pi = \left(\Sigma, \Lambda, \Gamma, \#, \mu_\Pi, \iota, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p)\right)$$

verifying the following

- $\Sigma = (c_1, \ldots, c_{m+1})$, and is such that $\Sigma \cap \Gamma_f = \emptyset$.
- There exist distinguished elements $\oplus, \ominus, \odot, \otimes, \oslash \in \Gamma \setminus \Gamma_f$.
- $\Lambda = (c_1, \ldots, c_m)$.
- $\# \neq \#_f$.
- $\mu_\Pi = [_1 \mu_{\Pi_f}]_1$, where the membranes from μ_{Π_f} have been adequately renamed (and therefore, also the rules of Π_f have been adapted). We denote σ_f the skin membrane of the latter. Also, we consider that ι_f reflects the new labeling of the input membrane of Π_f.
- $\iota = 1$.
- $p = p_f + 1$.
- $\mathcal{M}_1 = \{\{\#\}\}$. The remaining membranes are all empty.
- The evolution rules are the following:
  - Evolution rules for membrane 1:

$$\begin{aligned} \#c_{m+1} \to (\ominus, in_{\sigma_f}) > \#c_i \to \#(c_i, out) \quad (i = 1, \ldots, m) > \\ > \# \to (\#, out) > \#_f \# \to \#_f > \#_f \to (\oslash, in_{\sigma_f}) > \\ > \otimes^u b_i \to \otimes^u c_i \quad (i = 1, \ldots, m) > \otimes^u \to \# > \\ > c_i \to (c_i, in_{\sigma_f}) \quad (i = 1, \ldots, m) \end{aligned}$$

    where u is the degree of μ_{Π_f}.
  - For every membrane j distinct from membrane 1 the following rules are included:

$$\ominus \to \oplus(\ominus, in_{j_1}) \ldots (\ominus, in_{j_k})$$
$$\odot^u \oplus \to \mathcal{M}_j^f > \oplus \to \oplus \odot$$

    evolution rules associated with the membrane in Π_f

$$\oslash \to \otimes(\oslash, in_{j_1}) \ldots (\oslash, in_{j_k})$$
$$ob \otimes \to \otimes \quad (ob \in \Gamma_f) > \otimes \to (\otimes, out)$$

    where j_1 to j_k are the children membranes of membrane j and u is its level within μ_{Π_f}. Moreover, if j is ι_f, then the rule $\oplus \to \oplus\odot$ has higher priority than the original rules of this membrane in Π_f.

- Let $j_1, \ldots, j_q$ be the membrane path from σ_f to ι_f. Then, for $k = 1, \ldots, q-1$, the following rules are included in membrane j_k:

$$c_i \to (c_i, in_{j_{k+1}}) \quad (i = 1, \ldots, m)$$

  Also, the following rules are included in membrane j_q:

$$c_i \to a_i \quad (i = 1, \ldots, m)$$

The P system constructed in this way, denoted by $It(\Pi_f)$, is a valid computing P system with external output which computes the iteration of f.

Indeed, the system works as follows:

The number of iterations of f to make is given by the $(m+1)$th argument provided to $It(f)$. What we do then is to reduce this argument by one and, next, carry out a two phases process: first, computing one iteration of f; second "reseting" the P system Π_f to its initial state. We repeat this process until the $(m+1)$th argument gets to zero.

The test to decide if one iteration has to be done is performed in membrane 1 looking how many objects c_{m+1}, which represents the $(m+1)$th argument, there are. If such an object is present, the rule $\#c_{m+1} \to (\ominus, in_{\sigma_f})$ (followed by the rules $c_i \to (c_i, in_{\sigma_f})$) is applied, starting the calculation of a new iteration of f, which is done in two phases.

- Phase 1: *Computing one iteration of f*
  This phase starts when an object $\ominus$ gets into the skin of μ_{Π_f}. This object initiate counters in the membranes of μ_{Π_f}, in the same way than we did for the composition, in order to assure that they start to apply their original rules at the same time (local synchronization within Π_f). Also, with a gap of one step of computation that does not matter, the input data, represented by objects c_i, is taken from the skin of μ_{Π_f} to the input membrane of Π_f. Although through the execution of this phase the result of the iteration is being sent out into membrane 1, they do not activate any rule in it.
- Phase 2: *Reseting the P system Π_f*
  Phase 1 ends when some objects $\#_f$ get into membrane 1. Before we could compute another iteration of f, we need to erase all the objects left in the membranes of μ_{Π_f}. This is what we do in this stage, which start by reducing the objects $\#_f$ in membrane 1 to only one. Then the rule $\#_f \to (\oslash, in_{\sigma_f})$ in membrane 1 introduce an object $\oslash$ into the skin of μ_{Π_f}.
  This object spreads to all the membranes just as $\ominus$ do in the previous phase, and leave one object $\otimes$ in each of them. This latter objects act as erasers, removing all the objects from the membranes by means of the rules $ob\otimes \to \otimes$. When a membrane has been emptied, the object $\otimes$ is expelled from it.
  Therefore, this phase finishes when membrane 1 collects as many objects $\otimes$ as the degree of μ_{Π_f}. Only then the rules $\otimes^u b_i \to \otimes^u c_i$ can be applied, which transform the result of the iteration of f into input data for Π. Finally, the rule $\otimes^u \to \#$ is applied to start the process again.

When no object c_{m+1} is present in membrane 1, no more iteration has to be done. What is left is to send the objects c_1 to c_m of this membrane to the external environment, followed by the object #.

Note that during the evaluation of the test, no rule can be applied in another membrane other than membrane 1, because they are empty. □

6 Diophantine Sets

We introduce in this section the notion of diophantine set, which will help us to prove the universality of the model of computing P system with external output.

Definition 6.1. *A set of natural tuples, $A \subseteq \mathbb{N}^m$, is a diophantine set if there exists a polynomial $P(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{Z}[\boldsymbol{a}, \boldsymbol{x}]$ such that*

$$A = \{\boldsymbol{a} \in \mathbb{N}^m : \exists \boldsymbol{x} \in \mathbb{N}^n \big(P(\boldsymbol{a}, \boldsymbol{x}) = 0\big)\}$$

The following property is relatively easy to prove.

Proposition 6.2. *Every diophantine set is a recursively enumerable set.*

The main result about diophantine sets was obtained by Y. Matiyasevich from works by J. Robinson, M. Davis and H. Putnam, providing a negative solution for Hilbert's Tenth Problem.

Theorem 6.3 (MRDP [1]). *Every recursively enumerable set is a diophantine set.*

7 Generation of Diophantine Sets

First we need to introduce the concept of generation of a set by a P system.

Definition 7.1. *Let Π be a computing P system with external output of order (m, n).*

- *A set $A \subseteq \mathbb{N}^m$ is said to be partially generated by Π if this P system computes its partial characteristic function; that is, the function*

$$\mathcal{C}_A^*(k_1, \ldots, k_m) = \begin{cases} 1, & \text{if } (k_1, \ldots, k_m) \in A \\ \textit{undefined}, & \textit{otherwise} \end{cases}$$

- *A set $A \subseteq \mathbb{N}^m$ is said to be totally generated by Π if this P system computes its characteristic function.*

The main result of this paper is the following.

Theorem 7.2. *Every diophantine set is partially generated by a computing P system with external output.*

Before beginning with the proof, let us consider the following computing P systems with external output:

– P systems Π_n^{Id}, with $n \geq 1$:

$$\Sigma = \Lambda = (a_1, \ldots, a_n), \quad \mu_{\Pi_n^{Id}} = [_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{\#\}\}$$
$$R_1 = \{\# \to (\#, out)\} \cup \{a_i \to (a_i, out) : i = 1, \ldots, n\}, \quad \rho_1 = \emptyset$$

These P systems compute the identity functions, $Id^n : \mathbb{N}^n \to \mathbb{N}^n$, defined as $Id^n(k_1, \ldots, k_n) = (k_1, \ldots, k_n)$.

– P systems $\Pi_{n,j}^{proj}$, with $n \geq 1$ and $1 \leq j \leq n$:

$$\Sigma = (a_1, \ldots, a_n), \quad \Lambda = (a_j), \quad \mu_{\Pi_{n,j}^{proj}} = [_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{\#\}\}$$
$$R_1 = \{\# \to (\#, out), a_j \to (a_j, out)\}, \quad \rho_1 = \emptyset$$

These P systems compute the projection functions, $\Pi_j^n : \mathbb{N}^n \to \mathbb{N}$, defined as $\Pi_j^n(k_1, \ldots, k_n) = k_j$.

– P systems $\Pi_{n,c}^{const}$, with $n \geq 1$ and $c \in \mathbb{N}$:

$$\Sigma = (a_1, \ldots, a_n), \quad \Lambda = (b), \quad \mu_{\Pi_{n,c}^{const}} = [_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{b^c, \#\}\}$$
$$R_1 = \{\# \to (\#, out), b \to (b, out)\}, \quad \rho_1 = \emptyset$$

These P systems compute the constant functions, $\mathcal{C}_c^n : \mathbb{N}^n \to \mathbb{N}$, defined as $\mathcal{C}_c^n(k_1, \ldots, k_n) = c$.

– P systems $\Pi_2^{sum'}$, $\Pi_2^{prod'}$ and $\Pi_2^{expt'}$:
The first of these P systems is given by

$$\Sigma = \Lambda = (a_1, a_2), \quad \mu_{\Pi_2^{sum'}} = [_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{\#\}\}$$
$$R_1 = \{\# \to (\#, out), a_1 \to (a_1, out), a_2 \to (a_1, out)(a_2, out)\}, \quad \rho_1 = \emptyset$$

This P system computes the function $+' : \mathbb{N}^2 \to \mathbb{N}^2$, defined as $+'(k_1, k_2) = (k_1 + k_2, k_2)$.
The iteration of the P system $\Pi_2^{sum'}$ is a P system which computes the function $It(+') : \mathbb{N}^3 \to \mathbb{N}^2$ given by $It(+')(k_1, k_2, k_3) = (k_1 + k_2 k_3, k_2)$. Then, the P system $\Pi_2^{prod'} = C(It(\Pi_2^{sum'}); \Pi_{2,0}^{const}, \Pi_{2,2}^{proj}, \Pi_{2,1}^{proj})$ computes the function $*' : \mathbb{N}^2 \to \mathbb{N}^2$, defined as $*'(k_1, k_2) = (k_1 k_2, k_2)$.
The iteration of the P system $\Pi_2^{prod'}$ is a P system which computes the function $It(*') : \mathbb{N}^3 \to \mathbb{N}^2$ given by $It(*')(k_1, k_2, k_3) = (k_1 k_2^{k_3}, k_2)$. Then, the P system $\Pi_2^{expt'} = C(It(\Pi_2^{prod'}); \Pi_{2,1}^{const}, \Pi_2^{Id})$ computes the function $expt' : \mathbb{N}^2 \to \mathbb{N}^2$, defined as $expt'(k_1, k_2) = (k_1^{k_2}, k_1)$.

– P systems Π_n^{sum}, Π_n^{prod} and Π_2^{expt}:
We defined these P systems by recursion over $n \geq 2$.
$(n = 2)$

$$\Pi_2^{sum} = C(\Pi_{2,1}^{proj}; \Pi_2^{sum'})$$
$$\Pi_2^{prod} = C(\Pi_{2,1}^{proj}; \Pi_2^{prod'})$$
$$\Pi_2^{expt} = C(\Pi_{2,1}^{proj}; \Pi_2^{expt'})$$

$(n > 2)$

$$\Pi_n^{sum} = C(\Pi_{n-1}^{sum}; C(\Pi_2^{sum}; \Pi_{n,1}^{proj}, \Pi_{n,2}^{proj}), \Pi_{n,3}^{proj}, \ldots, \Pi_{n,n}^{proj})$$
$$\Pi_n^{prod} = C(\Pi_{n-1}^{prod}; C(\Pi_2^{prod}; \Pi_{n,1}^{proj}, \Pi_{n,2}^{proj}), \Pi_{n,3}^{proj}, \ldots, \Pi_{n,n}^{proj})$$

They compute, respectively, the n-ary sum function, the n-ary product function and the exponential function.

– P system Π_2^{dif}:

$$\Sigma = (a_1, a_2), \quad \Lambda = (b^+, b^-), \quad \mu_{\Pi_2^{dif}} = [\,_1]_1, \quad \iota = 1, \quad \mathcal{M}_1 = \{\{\#\}\}$$
$$(R_1, \rho_1) \equiv \{a_1 a_2 \to \lambda > \big(\# \to (\#, out), a_1 \to (b^+, out), a_2 \to (b^-, out)\big)\}$$

This P system computes the function $dif : \mathbb{N}^2 \to \mathbb{N}^2$ defined as $dif(k_1, k_2) = (\max(k_1 - k_2, 0), |\min(k_1 - k_2, 0)|)$.

Proof (of theorem 7.2). Given a polynomial $P(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{Z}[\boldsymbol{a}, \boldsymbol{x}]$, we construct a computing P system with external output which partially generates the diophantine set determined by this polynomial, in several steps:

1. Computing a monomial:
   Let us denote $\Pi_{j,k}^{expt_i} = C(\Pi_2^{expt}; \Pi_{j,k}^{proj}, \Pi_{j,i}^{const})$, which computes the function $expt_i^{j,k} : \mathbb{N}^j \to \mathbb{N}$ given by $expt_i^{j,k}(a_1, \ldots, a_j) = a_k^i$.
   Let $m(\boldsymbol{a}, \boldsymbol{x}) = c\, a_1^{i_1} \ldots a_m^{i_m} x_1^{j_1} \ldots x_n^{j_n}$, with $c > 0$, be a monomial of $P(\boldsymbol{a}, \boldsymbol{x})$. Then, the following computing P system with external output

$$\Pi_{c,i_1,\ldots,i_m,j_1,\ldots,j_n}^{mon} = C(\Pi_{m+n+1}^{prod}; \Pi_{m+n,c}^{const}, \Pi_{m+n,1}^{expt_{i_1}}, \ldots, \Pi_{m+n,m}^{expt_{i_m}}, \Pi_{m+n,m+1}^{expt_{j_1}}, \ldots, \Pi_{m+n,m+n}^{expt_{j_n}})$$

   computes $m(\boldsymbol{a}, \boldsymbol{x})$, considered as a function from $\mathbb{N}^{m+n}$ to $\mathbb{N}$.
2. Computing the polynomial:
   Suppose that

$$P(\boldsymbol{a}, \boldsymbol{x}) = \sum_{k=1}^{r_1} c_k a_1^{i_1^k} \cdots a_m^{i_m^k} x_1^{j_1^k} \cdots x_n^{j_n^k} - \sum_{k=r_1+1}^{r_1+r_2} c_k a_1^{i_1^k} \cdots a_m^{i_m^k} x_1^{j_1^k} \cdots x_n^{j_n^k}$$

   with $c_k > 0$ for every $k = 1, \ldots, r_1 + r_2$. Then, the following computing P system with external output

$$\Pi_{P(\boldsymbol{a},\boldsymbol{x})}^{pol} = C\big(\Pi_2^{dif}; \Pi_{P(\boldsymbol{a},\boldsymbol{x})}^{pol+}, \Pi_{P(\boldsymbol{a},\boldsymbol{x})}^{pol-}\big)$$

   where

$$\Pi_{P(\boldsymbol{a},\boldsymbol{x})}^{pol+} = C\big(\Pi_{r_1}^{sum}; \Pi_{c_1,\boldsymbol{i}^1,\boldsymbol{j}^1}^{mon}, \ldots, \Pi_{c_{r_1},\boldsymbol{i}^{r_1},\boldsymbol{j}^{r_1}}^{mon}\big)$$
$$\Pi_{P(\boldsymbol{a},\boldsymbol{x})}^{pol-} = C\big(\Pi_{r_2}^{sum}; \Pi_{c_{r_1+1},\boldsymbol{i}^{r_1+1},\boldsymbol{j}^{r_1+1}}^{mon}, \ldots, \Pi_{c_{r_1+r_2},\boldsymbol{i}^{r_1+r_2},\boldsymbol{j}^{r_1+r_2}}^{mon}\big)$$

   computes $P(\boldsymbol{a}, \boldsymbol{x})$, considered as a function from $\mathbb{N}^{m+n}$ to $\mathbb{N}$.

3. Computing the diophantine set:
Considering that $\Sigma^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} = (d_1, \ldots, d_m, e_1, \ldots, e_n)$ and $\Lambda^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} = (b^+, b^-)$, let us define the computing P system with external output

$$\Pi = (\Sigma, \Lambda, \Gamma, \#, \mu_\Pi, \iota, \mathcal{M}_1, \ldots, \mathcal{M}_p, (R_1, \rho_1), \ldots, (R_p, \rho_p))$$

as follows:
- $\Sigma = (d_1, \ldots, d_m)$.
- $\Lambda = (b)$.
- There exist distinct objects $\#', \#_1, \ldots, \#_n \in \Gamma \setminus \{\#, \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}\}$.
- $\mu_\Pi = [_1[_2]_2 \ldots [_{n+1}]_{n+1}\mu_{\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}}]_1$, where the membranes from $\mu_{\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}}$ have been adequately renamed (and therefore, also the rules of $\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}$ have been adapted). We denote σ_{pol} the skin membrane of the latter.
- $\iota = 1$.
- $p = p^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} + n + 1$.
- $\mathcal{M}_1 = \{\{\#\}\}, \mathcal{M}_2 = \{\{\#_1\}\}, \ldots, \mathcal{M}_{n+1} = \{\{\#_n\}\}$. All the remaining multisets are the empty one.
- Evolution rules:
  - Rules for membrane 1:

$$\#_1 \ldots \#_n \# \to \#' > \# \to \# > \\ > \begin{cases} d_i \to (d_i, in_{\sigma_{pol}}) & (i = 1, \ldots, m) \\ e_j \to (e_j, in_{\sigma_{pol}}) & (j = 1, \ldots, n) \\ \#' \to (\#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}, in_{\sigma_{pol}})(\ominus, in_{\sigma_{pol}}) \end{cases}$$

$$\left.\begin{matrix} b^+ \to b^+ \\ b^- \to b^- \end{matrix}\right\} > \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}\#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} \to \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} > \#^{pol}_{P(\boldsymbol{a},\boldsymbol{x})} \to (b, out)(\#, out)$$

  - Rules for membrane i, $2 \leq i \leq n+1$:

$$\#_{i-1} \to (\#_{i-1}, out)$$
$$\#_{i-1} \to \#_{i-1}(e_{i-1}, out)$$

  - The rules for the remaining membranes are the same than in $\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}$.

Then Π is a valid computing P system with external output which computes the partial characteristic function of the diophantine set represented by $P(\boldsymbol{a}, \boldsymbol{x})$.
Indeed, the P system works as follows:

a) First, a tuple $\boldsymbol{x}$ is nondeterministically generated from membranes 2 to $n+1$ into membrane 1.
b) Second, the input objects d_i and the objects e_i which represent the previous tuple in membrane 1 are sent into the skin membrane of $\mu_{\Pi^{pol}_{P(\boldsymbol{a},\boldsymbol{x})}}$ and the computation of P over the input $\boldsymbol{a}$ and the tuple $\boldsymbol{x}$ starts.
c) Finally, if a non-zero result is obtained, then the computation enters an infinite loop: the rule $b^+ \to b^+$ or the rule $b^- \to b^-$ is applied once and again. If a zero result is obtained, then these rules cannot be applied, and an object b and an object $\#$ are sent out of the membrane structure.

□

8 Conclusions

We have studied in this paper the computing P systems with external output. This is a variant of the model of computation introduced in [5], which in turn is a variant of the basic model of transition P system introduced by G. Păun in [3]. The idea behind this new model is to be able to compute functions without worrying about the content of the membrane structure used to do it, but only considering the objects collected in its external environment.

We have defined two operations between computing P systems with external output: composition and iteration. These operations have allowed us to prove the universality of this model, by means of the MRDP theorem about diophantine sets.

References

[1] Y. Matiyasevich. *Hilbert's Tenth Problem*. The MIT Press, 1993.
[2] G. Păun. Computing with membranes. An introduction. *Bull. European Assoc. Theoret. Comput. Sci.*, 67:139–152, 1999.
[3] G. Păun. Computing with membranes. *J. Comput. System Sci.*, 61(1):108–143, 2000.
[4] G. Păun and G. Rozenberg. A guide to membrane computing. *Theoret. Comput. Sci.*, to appear.
[5] G. Păun, G. Rozenberg, and A. Salomaa. Membrane computing with external output. *Fund. Inform.*, 41(3):313–340, 2000.
[6] F. Sancho Caparrini. *Verificación de programas en modelos de computación no convencionales*. PhD thesis, Universidad de Sevilla, Departamento de Ciencias de la Computación e Inteligencia Artificial, 2002.
[7] The P Systems Web Page, (`http://dna.bio.disco.unimib.it/psystems/`)

An Analysis of Computational Efficiency of DNA Computing

Atsushi Kameda[1], Nobuo Matsuura[3], Masahito Yamamoto[2], and Azuma Ohuchi[3]

[1] Japan Science and Technology Cooperation (JST)
Honmachi 4-1-8, Kawaguchi 332-0012, Japan
kameda@dna-comp.org
http://ses3.complex.eng.hokudai.ac.jp/index.html
[2] PRESTO, Japan Science and Technology Cooperation (JST)
and Graduate School of Engineering, Hokkaido University
North 13, West 8, Kita-ku, Sapporo 060-8628, Japan
masahito@dna-comp.org
[3] Graduate School of Engineering, Hokkaido University
North 13, West 8, Kita-ku, Sapporo 060-8628, Japan
ohuchi@dna-comp.org

Abstract. In this paper, we investigate the relation between an experimental condition and computational (reaction) efficiency of a simple model in DNA computing. A DNA computing algorithm needs many chemical and enzyme reactions in its solution process. An experimental condition for DNA computing is generally based on a molecular biology protocol. However, the purpose of an experiment is different in DNA computing and molecular biology. Therefore, we verified the relation between an experimental condition, reaction time, and reaction efficiency specifically from the viewpoint of DNA computing.

1 Introduction

Since Adleman's experiment in 1994, some DNA computing experiments for solving various problems have been presented [1]. To solve many experimental procedures were needed, such as kination reaction, ligation reaction, hybridization (annealing), polymerase chain reaction (PCR), and gel electrophoresis, etc. These experimental procedures were mostly developed in molecular biology; therefore experimental conditions in DNA computing have generally been based on molecular biology protocols.

However, in molecular biology and DNA computing, the purpose of an experiment is generally different. DNA computing requires greater accuracy and speed compared with molecular biology. An experimental condition in molecular biology is optimized as an end in itself, therefore it seems that such experimental conditions might not be suitable for DNA computing.

In this paper, we focus on the relationship between an experimental condition and a reaction efficiency in a common experimental protocol based on molecular

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 191–198, 2002.

biology. We select the construction of DNA paths in a simple line graph as the experimental sample. This experimental procedure involves a kination reaction, hybridization and a ligation reaction.

2 Motivation

In order to analyze the relation between the experimental conditions and the computational (reaction) efficiency of DNA computing, we used a very simple model, i.e., a process to construct DNA paths in a directed graph, based on the Adleman-Lipton model [2]. The construction of DNA path in the mixture that contains some oligonucleotides needs a lot of chemical reactions such as kination, hybridization, and ligation. In Fig. 1, the typical experimental procedure to construct DNA paths for a very simple line graph is shown. To visualize the constructed DNA paths, PCR and gel electrophoresis would also have to be performed. Therefore, the computational results are derived from many experimental factors in a complicated process.

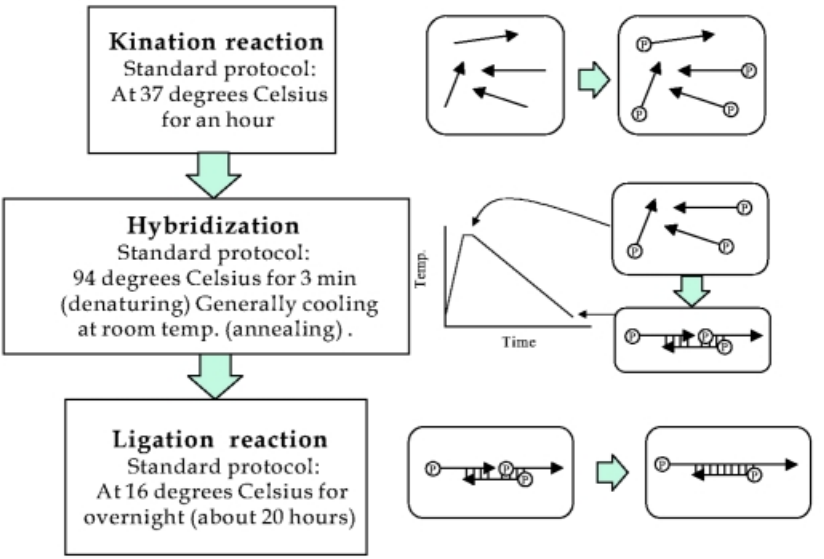


Fig. 1. Experimental procedure of the construction of DNA paths for a simple line graph. This procedure is composed of three steps. First, the mixing and kination process in which the DNA oligomer that constitute the simple line graph were mixed and the 5' end was phosphorylated. Next, the hybridization process in which DNA oligomers were annealed by general cooling after denaturing. Last, the ligation process that bridges the nicks in the generated DNA paths.

In this paper, we focus on the ligation reaction as a means to reduce the total computational time in DNA computing, i.e., we analyze the relation between ligation time and efficiency. Aoi et al. reported an analysis of the ligation process from the viewpoint of DNA computing; however, they investigated ligation errors for cases where there were some mismatches in the sequences [3]. We investigate ligation efficiency itself in this paper. In order to isolate only the influence of the ligation process, some reaction parameters in the kination reaction and hybridization are fixed. Moreover, by using SYBR Gold (Molecular Probes, Inc.) staining, the PCR reaction can be omitted. Therefore, we can investigate the ligation efficiency by simply measuring the concentration of the constructed DNA paths. To reducing the total computational time, it would be helpful to know the relation between the reaction time and efficiency.

3 Materials and Methods

3.1 Experiments for Simple Line Graph

We applied a simple line graph with two or three vertices, as shown in Fig. 2, in order to confirm the reaction efficiency of each step in Fig. 1.

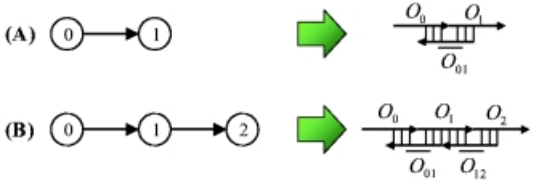


Fig. 2. Simple line graph with (A) two vertices and one edge or (B) three vertices and two edges. Each DNA sequence is described in Table 1.

3.2 DNA Sequences and Paths

DNA sequences of vertices and edges (DNA vertices and edges) were designed by using simulated annealing method, which is the encoding method based on the binding degree between a pair of DNA oligonucleotides [4]. The obtained sequence designs are shown in Table 1. DNA sequences were synthesized by Hokkaido System Science Co. (Hokkaido, Japan). Two sets of oligonucleotides were used as simple line graphs, and are shown in Table 2.

3.3 Standard Protocol

STEP 1: Mixing and Kination of DNA vertices and edges. DNA paths were generated in the hybridization process. We mixed DNA vertices and edges in one

Table 1. Designed sequences of DNA oligomers for simple line graphs.

Name	Sequences (5' → 3')
O_0	TAGCCTAGAGCACGAATACC
O_1	CGTCTATGGACCGTTCAAGA
O_2	AGCCTATCCATAACCATGCC
$\overline{O_{01}}$	TCCATAGACGGGTATTCGTG
$\overline{O_{12}}$	TGGATAGGCTTCTTGAACGG

Table 2. DNA oligomer sets for simple line graphs.

DNA set 1	DNA set 2
two vertices	three vertices
O_0	O_0
O_1	O_1
O_{01}	O_2
	$\overline{O_{01}}$
	$\overline{O_{12}}$

tube. The 5' end of the DNA vertices and edges was phosphorylated by using 5 units of T4 Polynucleotide Kinase (Takara Co., Japan) in 100μl of reaction mixture containing Kinase Buffer provided by the supplier (Takara) and 1 mM ATP (Reaction A in Fig. 1). The kination reaction was performed at 37°C for one hour.

STEP 2: Generation of DNA paths. The reaction mixture was heated to 94°C for denaturing and gradually cooled to make annealed oligonucleotide. The thermal schedule for denaturing and annealing was 94°C for 3 min and gradual cooling to 16°C (-0.02°C/sec). After this process, DNA paths were generated from DNA vertices and edges.

STEP 3: Ligation nicks of DNA paths. In order to bridge the nicks of the DNA paths, 175 units of T4 DNA Ligase (Takara) and 0.5 mM ATP and 10 mM DTT were added to the mixture. The ligation reaction was performed at 16°C overnight (about 20 hours).

STEP 4: Quantification of generated DNA paths. We performed 5% polyacrylamide gel electrophoresis (PAGE). A 5% PAGE enables separation of DNA paths according to their length. DNA paths subjected to 5% PAGE were visualized by SYBR Gold (Molecular Probes, Inc.) staining. We captured the image of the gel with the imaging system, Molecular Imager FX (BIO-RAD Co., USA), and quantified the intensity of the DNA bands by using the image analyzing software, Quantity One (BIO-RAD Co.).

3.4 Ligation Test

In the standard protocol, ligation time is overnight (about 20 hours). This ligation time was changed to 5, 30, 60, 300, 1,200, 3,600, 10,800 and 72,000 sec. DNA sets, 1 and 2 were used (Table 2).

4 Results

We confirmed the construction of DNA paths using DNA set 1 (two vertices) by 5% PAGE, shown in Fig. 3(A). 40 bp bands were complete DNA paths and 20 bp bands were of DNA oligomer which had not constructed DNA paths. Fig. 3(B) shows the efficiency of the ligation reaction in the construction of DNA paths using DNA set 1 (two vertices). About 60% of the DNA oligomer was used for constructing complete DNA paths at 5 sec. At 20 min, about 80% of the DNA oligomer had been used. At 20 hours, about 90% of the DNA oligomer had been used and about 10% of the DNA oligomer remained without constructing DNA paths.

DNA path construction using DNA set 2 (three vertices) was also confirmed, as shown in Fig. 4(A). Some 60 bp bands were complete DNA paths. 50 bp and 40 bp bands were incomplete DNA paths lacking one or two DNA oligomers, and 20 bp bands were of DNA oligomer which had not constructed DNA paths.

Figure 4 shows the efficiency of the ligation reaction in the construction of DNA paths using DNA set 2 (three vertices). The rate of constructing complete DNA paths is lower than that of the two vertices simple line graph. The final rate of constructing complete DNA paths was about 60% at 20 hours. From these results, it can be seen that the more DNA vertices and edges, the lower the construction rate of complete DNA paths.

5 Discussion

Since the experimental result from 1 min to 60 min kination time was almost the same, it seems that 60 min kination in the standard protocol is longer than the time which needs for enough phosphorylating (data not shown). The kination process is preparation of the ligation reaction. This process can be omitted by using the 5' end phosphorylated DNA oligomers, and the total time for the construction of DNA paths can thus be shortened.

Ligation time in the standard protocol is about 20 hours. The efficiency of the ligation reaction was about 90% at 20 hours in Fig. 3. The main reasons for this efficiency may be (i) existence of non-phosphorylated DNA oligomer caused by incomplete kination process, (ii) lowering of enzyme activity of T4 DNA ligase, and (iii) mishybridization in the hybridization process. Reason (i) can be verified by using the 5' end phosphorylated DNA oligomers. In the reaction mixture, enzyme activity of T4 DNA ligase was excessive (over 10^6 fold), so it was hard to believe that this efficiency was caused by (ii). In Fig. 4, the ratio of the efficiency of constructing complete DNA paths was lower than that of Fig. 3.

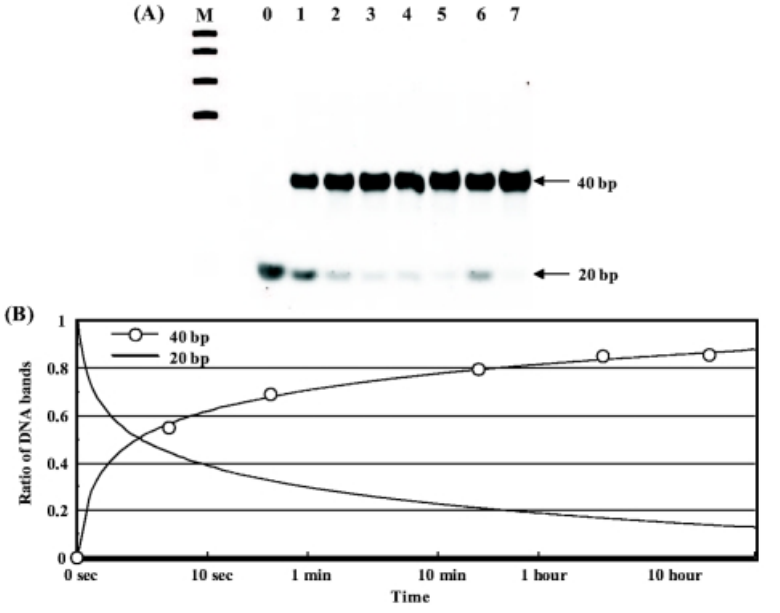


Fig. 3. (A) Confirmation of the formation of DNA paths from DNA set 1 with 5% PAGE captured by Molecular Imager FX. Lane M, DNA size marker (100 bp ladder); Ligation times were as follows, lane 0: 0 min, lane 1: 5 sec, lane 2: 30 sec, lane 3: 1 min, lane 4: 5 min, lane 5: 20 min, lane 6: 3 hour, lane 7: 20 hour. (B) Quantification of generated DNA paths from DNA set 1. Open circle denotes the ratio of 40 bp DNA band that contains complete DNA path. Close circle denotes the ratio of 20 bp DNA band that contains DNA oligomer.

The differences between these two conditions lay in the number of ligation points (nicks) needed for constructing complete DNA paths. Whenever the number of ligation points needed for constructing completely DNA paths increases, the efficiency of the construction of complete DNA paths decreases exponentially. In the case of Fig. 4(B), three ligation points need ligasing for construction of a complete DNA path (Fig. 2(B)). So, it is expected that the ratio of complete DNA paths in Fig. 4(B) should be equal to the ratio of complete DNA paths in Fig. 3(B) cubed. This expectation was almost exactly from the results of Fig. 3 and 4. These results suggest the following. As a DNA path becomes large, it becomes more difficult to make a perfect DNA path. Although the possibility of (iii) was also taken into consideration, there is still much that is unknown about hybridization process.

We are investigating hybridization process at present. So far we have tested various cooling schedule parameters from $-0.02°C$/sec to $-2.00°C$/sec. Annealing times have also been varied from about 70 min ($-0.02°C$/sec) to about 4

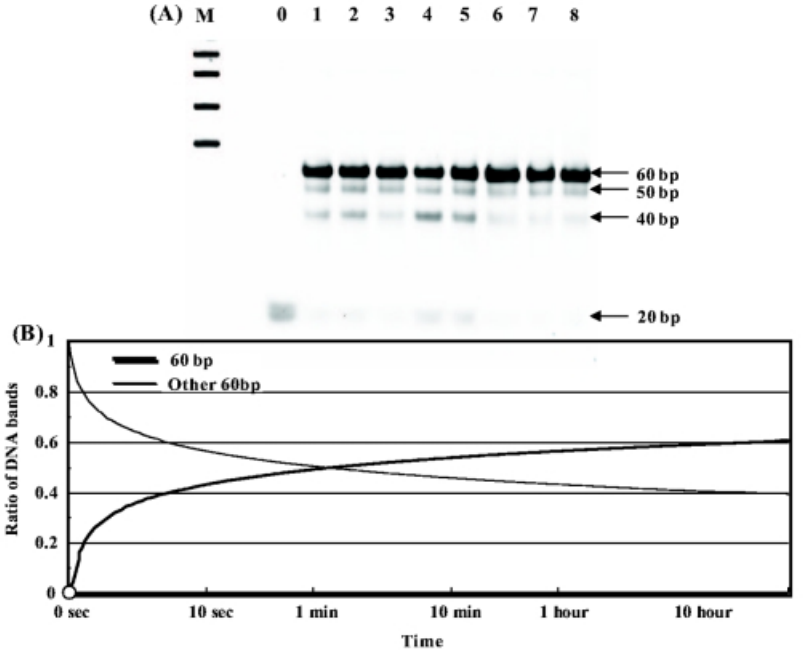


Fig. 4. (A) Confirmation of the formation of DNA paths from DNA set 2 (Table 2) with 5% PAGE captured by Molecular Imager FX. Lane M, DNA size marker (100 bp ladder); Ligation times were as follows, lane 0: 0 min, lane 1: 5 sec, lane 2: 30 sec, lane 3: 1 min, lane 4: 5 min, lane 5: 20 min, lane 6: 1 hour, lane 7: 3 hour, lane 8: 20 hours. (B) Quantification of generated DNA paths from DNA set 2. Open circle denotes the ratio of 60 bp DNA band that contains complete DNA paths. Close circle denotes the ratio of the other DNA bands (20, 40 and 50 bp) that contains DNA oligomer and incomplete DNA paths.

min (−2.00°C/sec). If a hard cooling schedule can anneal complementary DNA path without mishybridization, total reaction time can be further shortened. The hybridization process seemed to be influenced by the cooling schedule and DNA concentration according to our unpublished experimental data. If optimization of experimental conditions progresses, it will be possible to perform that DNA computing efficiently with a shorter reaction time.

References

1. L. Adleman, "Molecular Computation of Solutions to Combinatorial Problems", *Science* 266, pp. 1021-1024 (1994).
2. R. J. Lipton, "DNA Solution of Hard Computational Problems", *Science* 268, pp. 542-545 (1995).

3. Y. Aoi, T. Yoshinobu, K. Tanizawa, K. Kinoshita and H. Iwasaki, "Ligation Errors in DNA Computing", *BioSystems*, Vol. 52, No. 1-3, pp. 181-187 (1999).
4. M. Yamamoto, J. Yamashita, T. Shiba, T. Hirayama, S. Takiya, K. Suzuki, M. Munekata and A. Ohuchi, "A Study on the Hybridization Process in DNA Computing", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 54, pp. 101-110, 2000.

Communication and Computation by Quantum Games

Takeshi Kawakami

Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
cawakami@is.s.u-tokyo.ac.jp

Abstract. In the game called Prisoner's Dilemma, none of two prisoners, as players, is permitted to exchange information with the other and has to make his decision without knowing the decision of the opponent. Prisoners tell their jailer as their arbitrator their decisions and the jailer gives each of the prisoners an appropriate payoff according to the payoff function (the payoff table). That is, a certain communication between the two prisoners is performed at the moment the payoff is given to them by the jailer. Motivated by such a view of Prisoner's Dilemma game and many other games, we study communication and information carriers in quantum games. We also show that, quite surprisingly, communications in special quantum games can be used to solve problems that cannot be solved by using communications in classical games.

1 Introduction

Game theory is a field in which a process of decision making is analyzed. The game called "Prisoner's Dilemma" [4] can be used to discuss interesting points and difficulties of such analyses. In the standard game theory, equilibria of games are of a central interest. However, this will not be our case. We will look at quantum games from the new point of view and we will discuss special communication and computation aspects of games, not equilibria. In the Prisoner's Dilemma, two prisoner, Alice and Bob, are prohibited to communicate with each other, and forced to decide whether they will cooperate (C) or defect (D). After they tell their jailer, as their arbitrator, their decision, he gives a payoff to each of them, based on Table 1. In Table 1, each row stands for a decision of Alice and each column for a decision of Bob. The left side of each cell is the payoff of Alice and the right side is the payoff of Bob.

For example, if Alice decides C and gains the payoff 3, the Alice knows that the decision of Bob was C by Table 1. That is, the payoff given by the jailer contains an information, which determines one of two possibilities, about opponent's decision. We can also think Table 1 instead of Table 1 as a payoff table. In this case, an information which the payoff given by the jailer determines one of four possibilities. Later we will see a payoff table like Table 1 in a quantum game.

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 199–207, 2002.

Table 1. 2×2 and 4×4 payoff tables

	C	D
C	3 3	0 5
D	5 0	1 1

	C_1	C_2	D_1	D_2
C_1	3 3	1 1	0 5	5 0
C_2	1 1	3 3	5 0	0 5
D_1	5 0	0 5	1 1	3 3
D_2	0 5	5 0	3 3	1 1

In this paper, we first focus on such an aspect of quantum games, and we discuss information exchanged in it as well as its information carriers. In quantum games, there is the special carrier which does not exist in classical games, and it becomes possible to exchange two-bit information in them due to it. Furthermore, later we discuss computations that are possible in quantum games, but not in classical games, through communication and measurement.

2 Communication by Quantum Games

Let us start with a detailed analysis of the communication aspects of the quantized Prisoner's Dilemma game. In our approach, we quantize the decisions in the original Prisoner's Dilemma as follows:

$$\mathrm{C} \longrightarrow |0\rangle, \quad \mathrm{D} \longrightarrow |1\rangle.$$

That is, each prisoner has his private qubit and applies a unitary transformation to this. Their jailer gives a payoff to each of them based on a measured result of each qubit (Table 2).

Table 2. Payoff by measurement result

	$\|0\rangle$	$\|1\rangle$
$\|0\rangle$	3 3	0 5
$\|1\rangle$	5 0	1 1

Therefore, unitary transformations are strategies for prisoners and the key roles will be played by Pauli matrices.

2.1 Pauli Matrices

Any unitary matrix $\boldsymbol{U}$ of order 2 can be represented as (see [3])

$$\boldsymbol{U} = e^{i\gamma} \begin{pmatrix} e^{i\alpha} & 0 \\ 0 & e^{-i\alpha} \end{pmatrix} \begin{pmatrix} \cos\theta & i\sin\theta \\ i\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} e^{i\beta} & 0 \\ 0 & e^{-i\beta} \end{pmatrix}.$$

If $\gamma = 0$, $\boldsymbol{U}$ can be represented by the identity matrix $\boldsymbol{I}$ and Pauli matrices

$$\boldsymbol{\sigma}_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \boldsymbol{\sigma}_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \boldsymbol{\sigma}_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

as follows:

$$\boldsymbol{U} = \cos\phi\cos\theta \cdot \boldsymbol{I} + \sin\phi\cos\theta \cdot i\boldsymbol{\sigma}_z + \cos\psi\sin\theta \cdot i\boldsymbol{\sigma}_x + \sin\psi\sin\theta \cdot i\boldsymbol{\sigma}_y.$$

In the following we make use of the fact that Pauli matrices have the following properties:

$$\boldsymbol{\sigma}_j{}^2 = \boldsymbol{I} \quad \text{if} \quad j \in \{x, y, z\}, \qquad \boldsymbol{\sigma}_k\boldsymbol{\sigma}_l = -\boldsymbol{\sigma}_l\boldsymbol{\sigma}_k = i\boldsymbol{\sigma}_m$$

if (k, l, m) is a cyclic permutation of (x, y, z).

2.2 Strategies

In our approach, strategies of prisoners$\boldsymbol{U}$ (unitary transformations) are restricted to be products of the operators $\boldsymbol{I}$, $\boldsymbol{\sigma}_x$, $\boldsymbol{\sigma}_y'$ $(= i\boldsymbol{\sigma}_y)$, and $\boldsymbol{\sigma}_z$, and can be represented as follows:

$$\boldsymbol{U} = \boldsymbol{\sigma}_z{}^j\boldsymbol{\sigma}_x{}^k. \tag{2.1}$$

That is, each prisoner has to choose a j and a k. For example, if $j = 1$ and $k = 1$, $\boldsymbol{U}$ is $\boldsymbol{\sigma}_y{}'$. In this game, matrices $\boldsymbol{J}$ and $\boldsymbol{J}^*$, the operators which their jailer can only use, are

$$\boldsymbol{J} = \frac{1}{\sqrt{2}}\left(\boldsymbol{I} \otimes \boldsymbol{I} + i\boldsymbol{\sigma}_x \otimes \boldsymbol{\sigma}_x\right), \quad \boldsymbol{J}^* = \frac{1}{\sqrt{2}}\left(\boldsymbol{I} \otimes \boldsymbol{I} - i\boldsymbol{\sigma}_x \otimes \boldsymbol{\sigma}_x\right).$$

The strategy of Alice, $\boldsymbol{U}_A$, and the strategy of Bob, $\boldsymbol{U}_B$, are then represented as

$$\boldsymbol{U}_A = \boldsymbol{\sigma}_z{}^{j_A}\boldsymbol{\sigma}_x{}^{k_A}, \quad \boldsymbol{U}_B = \boldsymbol{\sigma}_z{}^{j_B}\boldsymbol{\sigma}_x{}^{k_B}.$$

The initial state $\boldsymbol{\varphi}_{in}$ and the final state $\boldsymbol{\varphi}_{fin}$ [1,2] which is the state on which measurements are performed are the following ones:

$$\boldsymbol{\varphi}_{in} = \boldsymbol{J}|00\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{i}{\sqrt{2}}|11\rangle, \qquad \boldsymbol{\varphi}_{fin} = \boldsymbol{J}^*\left(\boldsymbol{U}_A \otimes \boldsymbol{U}_B\right)\boldsymbol{J}|00\rangle.$$

In such a setting, we get

$$\begin{aligned}&\boldsymbol{J}^*\left(\boldsymbol{U}_A \otimes \boldsymbol{U}_B\right)\boldsymbol{J} \\ &\quad = \boldsymbol{\sigma}_z{}^{j_A}\boldsymbol{\sigma}_x{}^{k_A} \otimes \boldsymbol{\sigma}_z{}^{j_B}\boldsymbol{\sigma}_x{}^{k_B}\left(\delta_{f(j_A,j_B),0}\boldsymbol{I} \otimes \boldsymbol{I} + i\delta_{f(j_A,j_B),1}\boldsymbol{\sigma}_x \otimes \boldsymbol{\sigma}_x\right)\end{aligned}$$

where

$$f\left(j_A, j_B\right) = \left(j_A + j_B\right) \bmod 2 \quad \text{and} \quad \delta_{m,n} = \begin{cases} 0 & \text{if} \quad m \neq n \\ 1 & \text{if} \quad m = n \end{cases}.$$

Therefore, the final state of the game can be represented as

$$\boldsymbol{\varphi}_{fin} = {\boldsymbol{\sigma}_z}^{j_A}{\boldsymbol{\sigma}_x}^{k_A} \otimes {\boldsymbol{\sigma}_z}^{j_B}{\boldsymbol{\sigma}_x}^{k_B} \left(\delta_{f(j_A,j_B),0}|00\rangle + i\delta_{f(j_A,j_B),1}|11\rangle\right). \tag{2.2}$$

We can determine the payoff which is given to each prisoner on the basis of his strategy from formula (2.2) and Table 2 and it is given in Table 3. Moreover, we can see from formula (2.1) that each strategy can be distinguished by j and k. Therefore, each strategy is represented by a pair (j, k) in Table 3. Let Alice decide 00. In such a case, Alice gains the payoff 5 and knows that the decision of Bob was 11, by Table 3. In this case, information which is exchanged between them through their jailer is represented as 2 bits, to determine one of the four possibilities.

Table 3. Payoff by strategies

jk	00	10	01	11
00	3 3	1 1	0 5	5 0
10	1 1	3 3	5 0	0 5
01	5 0	0 5	1 1	3 3
11	0 5	5 0	3 3	1 1

2.3 Communiction Aspects of Games

A game can have a communication aspect. Indeed, in Prisoner's Dilemma game, there is a communication of two bits of information by two local one-qubit operations while in a classical game, there is a communication of one bit by one-bit operations. This is similar to superdense coding [3] and indicates that quantum communication has more computational ability than the classical one.

In our quantum game, each unitary matrix ${\boldsymbol{\sigma}_z}^{j}{\boldsymbol{\sigma}_x}^{k}$, a strategy which prisoners can use, can be seen as

$$\boldsymbol{I}: \quad |0\rangle \longrightarrow |0\rangle \qquad |1\rangle \longrightarrow |1\rangle \quad \text{if } j = 0 \text{ and } k = 0, \tag{2.3}$$

$$\boldsymbol{\sigma}_x: \quad |0\rangle \longrightarrow |1\rangle \qquad |1\rangle \longrightarrow |0\rangle \quad \text{if } j = 0 \text{ and } k = 1, \tag{2.4}$$

$$\boldsymbol{\sigma}_z: \quad |0\rangle \longrightarrow |0\rangle \qquad |1\rangle \longrightarrow -|1\rangle \quad \text{if } j = 1 \text{ and } k = 0, \tag{2.5}$$

$$\boldsymbol{\sigma}'_y: \quad |0\rangle \longrightarrow -|1\rangle \qquad |1\rangle \longrightarrow |0\rangle \quad \text{if } j = 1 \text{ and } k = 1. \tag{2.6}$$

That is,

$${\boldsymbol{\sigma}_z}^{j}{\boldsymbol{\sigma}_x}^{k}: \quad |0\rangle \longrightarrow (-1)^{jk}|k\rangle, \qquad |1\rangle \longrightarrow (-1)^{j\overline{k}}|\overline{k}\rangle.$$

It can be said that ${\boldsymbol{\sigma}_z}^{j}{\boldsymbol{\sigma}_x}^{k}$ changes not only the value of a qubit, but also its phase (amplitude). If there would be a way to distinguish the phase, a qubit could contain infinite amounts of information. But we cannot distinguish phases by the measurements. For example, in the case of the state

$$\boldsymbol{\varphi} = a|0\rangle \quad \left(|a|^2 = 1\right),$$

there are infinitely many complex numbers such that $|a|^2 = 1$, but we cannot distinguish them by measurement. Therefore, phase difference is of no importance for measurement and one qubit contains only one bit information that can be distinguished by measurement. However, in our quantum game, two-bit communication can be done by two local one-qubit operations. This means that the phase difference is distinguished somehow.

The initial state $\boldsymbol{\varphi}_{in}$ in our quatum game is

$$\boldsymbol{\varphi}_{in} = \boldsymbol{J}|00\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{i}{\sqrt{2}}|11\rangle.$$

That is, an entangled state [3] is created by the operator $\boldsymbol{J}$. Observe that one of the properties of the tensor product, for vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and constants a, b, is

$$(a\boldsymbol{x}) \otimes (b\boldsymbol{y}) = ab\,(\boldsymbol{x} \otimes \boldsymbol{y})\,. \tag{2.7}$$

formula (2.7) can be seen as the phase interaction that happens between two different qubits. That is, after each prisoner applies a unitary transformation to his qubit, the phases of two different qubits interact. Due to such a view, we can say that in formula (2.2), $f\,(j_A, j_B)$ is determined by this interaction. In other words, $\delta_{f(j_A,j_B),0}$ or $\delta_{f(j_A,j_B),1}$ represent interferences caused by $\boldsymbol{J}^{\boldsymbol{*}}$.

In our Prisoner's Dilemma game, an entangled state, in which each qubit is correlated with other one, is created by the operator $\boldsymbol{J}$, the phase interaction between qubits is caused by unitary transformations of prisoners, and the interference is due to the operator $\boldsymbol{J}^{\boldsymbol{*}}$. Therefore, by such a process, the phase differences in formulas (2.3)$\backsim$(2.6) are distinguished.

For example, let Alice decide to perform strategy $\boldsymbol{\sigma}_z\,(jk = 10)$ and let Bob decide to perform strategy $\boldsymbol{\sigma}'_y\,(jk = 11)$. Since the entangled state $\boldsymbol{\varphi}_{in}$ created by $\boldsymbol{J}$ is

$$\boldsymbol{\varphi}_{in} = \boldsymbol{J}|00\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{i}{\sqrt{2}}|11\rangle,$$

after Alice and Bob apply their unitary transformations to their qubits, the following phase interaction between qubits takes place

$$\begin{aligned}(\boldsymbol{U}_A \otimes \boldsymbol{U}_B)\,\boldsymbol{\varphi}_{in} &= \frac{1}{\sqrt{2}}|0\rangle \otimes (-|1\rangle) + \frac{i}{\sqrt{2}}\,(-|1\rangle) \otimes |0\rangle \\ &= -\frac{1}{\sqrt{2}}|01\rangle - \frac{i}{\sqrt{2}}|10\rangle.\end{aligned}$$

Finally, due to the interference caused by $\boldsymbol{J}^*$, we get

$$\begin{aligned}\boldsymbol{\varphi}_{fin} &= \boldsymbol{J}^* \left(-\frac{1}{\sqrt{2}}|01\rangle - \frac{i}{\sqrt{2}}|10\rangle\right) \\ &= -\frac{1}{2} \cdot 2 \cdot |01\rangle + \frac{i}{2} \cdot 0 \cdot |10\rangle \\ &= -|01\rangle.\end{aligned}$$

Therefore, in this case, the state $|01\rangle$ is measured by the jailer and Alice gains the payoff 0 and Bob gains the payoff 5 respectively, according to Table 2. From the payoff, Alice knows that Bob has changed his qubit as follows, $|0\rangle \longrightarrow -|1\rangle$ and $|1\rangle \longrightarrow |0\rangle$, and Bob knows that Alice has changed her qubit as follows, $|0\rangle \longrightarrow |0\rangle$ and $|1\rangle \longrightarrow -|1\rangle$. Therefore, it can be said that the payoff the jailer returns to each prisoner includes the phase information of each qubit and this way it becomes possible to distinguish a phase difference in a quantum game.

3 Computation by Quantum Games

A communication that is meaningless in a classical game can be meaningful in a quantum game. We show that the following problem can be solved by playing a quantum game n times.

Problem: There are M prisoners. Let the prisoner P_i have a number $b_i \in \{0, 1\}$ and n be any integer. Determine the remainder of the sum of b_i's divided by 2^n under the following conditions:

- No prisoner can communicate with other prisoners.
- A prisoner can send only his qubit to the jailer.
- Each prisoner knows the total number of prisoners.
- The jailer cannot measure the qubits sent from the prisoners.

That is, the jailer cannot know measured results of any qubits and each prisoner can know only a measured result of his qubit.

In this case, the operators and the states, $\boldsymbol{J}, \boldsymbol{J}^*, \boldsymbol{\varphi}_{in}, \boldsymbol{\varphi}_{fin}$ are defined as follows:

$$\boldsymbol{J} = \frac{1}{\sqrt{2}} \bigotimes_{j=1}^{M} \boldsymbol{I} + \frac{i}{\sqrt{2}} \bigotimes_{j=1}^{M} \boldsymbol{\sigma}_x, \quad \boldsymbol{J}^* = \frac{1}{\sqrt{2}} \bigotimes_{j=1}^{M} \boldsymbol{I} - \frac{i}{\sqrt{2}} \bigotimes_{j=1}^{M} \boldsymbol{\sigma}_x,$$

$$\boldsymbol{\varphi}_{in} = \boldsymbol{J} \bigotimes_{j=1}^{M} |0\rangle = \frac{1}{\sqrt{2}} \bigotimes_{j=1}^{M} |0\rangle + \frac{i}{\sqrt{2}} \bigotimes_{j=1}^{M} |1\rangle, \quad \boldsymbol{\varphi}_{fin} = \boldsymbol{J}^* \left(\bigotimes_{j=1}^{M} \boldsymbol{U}_j(t) \right) \boldsymbol{J} \bigotimes_{j=1}^{M} |0\rangle.$$

Let us now assume that a unitary transformation $\boldsymbol{U}_j(t)$, which jth prisoner can apply to his qubit, is the following one:

$$\boldsymbol{U}_j(t) = \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(-i\frac{2\pi}{2^t M} r_{t-1}^j\right) \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(i\frac{2\pi}{2^t} b_j\right) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(i\frac{2\pi}{2^t}\left(b_j - \frac{r_{t-1}^j}{M}\right)\right) \end{pmatrix}$$

where b_j is a number which the jth prisoner has, t is an iteration time. Later we prove that r_{t-1}^j is the remainder of 2^{t-1} of the sum of numbers prisoners have, that is updated every iteration by a recursive formula (3.1), and its initial value r_0^j is 0. Therefore,

$$\boldsymbol{U}_j(t): \quad |0\rangle \longrightarrow |0\rangle \qquad |1\rangle \longrightarrow \exp\left(i\frac{2\pi}{2^t}\left(b_j - \frac{r_{t-1}^j}{M}\right)\right)|1\rangle.$$

The final state $\boldsymbol{\varphi}_{fin}$ is therefore

$$\begin{aligned}\boldsymbol{\varphi}_{fin} &= \boldsymbol{J}^* \left(\bigotimes_{j=1}^{M} \boldsymbol{U}_j (t) \right) \boldsymbol{J} \bigotimes_{j=1}^{M} |0\rangle \\ &= \frac{1}{2} \left[1 + \exp \left(i\pi \frac{\sum_{j=1}^{M} b_j - r_{t-1}^j}{2^{t-1}} \right) \right] \bigotimes_{j=1}^{M} |0\rangle \\ &\qquad - \frac{i}{2} \left[1 - \exp \left(i\pi \frac{\sum_{j=1}^{M} b_j - r_{t-1}^j}{2^{t-1}} \right) \right] \bigotimes_{j=1}^{M} |1\rangle \\ &= \frac{1 + (-1)^{R(t)}}{2} \bigotimes_{j=1}^{M} |0\rangle - i \frac{1 - (-1)^{R(t)}}{2} \bigotimes_{j=1}^{M} |1\rangle,\end{aligned}$$

where

$$R_j (t) = \frac{\sum_{j=1}^{M} b_j - r_{t-1}^j}{2^{t-1}}.$$

Let us now introduce a function f_t defined as follows:

$$f_t (b_1, b_2, \ldots, b_n) = \left(\sum_{j=1}^{M} b_j \text{ quotient } 2^{t-1} \right) \bmod 2.$$

Let us assume that before starting $(t+1)$th game iteration $(t \geq 1)$, each prisoner P_j updates r_{t-1}^j to r_t^j by a unitary transformation,

$$r_t^j := r_{t-1}^j + q_t^j \cdot 2^{t-1} \quad \left(r_0^j = 0 \right) \tag{3.1}$$

where q_t^j is the result of the measurement of the qubit of P_j after tth game iteration which he can measure.

Proposition: r_t^j is the remainder of 2^t of the sum of numbers prisoners have, for $t \geq 0$.

Proof If $t = 0$, it is clear that the proposition holds.

If $t = k\,(\geq 1)$, then

$$R_j (k) = \frac{\sum_{j=1}^{M} b_j - r_{k-1}^j}{2^{k-1}}.$$

Let us next suppose that the proposition holds for $t = k - 1$ and we can represent in such a case, $R(k)$ as follows:

$$R_j (k) = \sum_{j=1}^{M} b_j \text{ quotient } 2^{k-1}. \tag{3.2}$$

Therefore, it holds

$$f_k\left(b_1, b_2, \ldots, b_n\right) = R_j\left(k\right) \bmod 2. \tag{3.3}$$

After a kth game iteration, the final state is

$$\varphi_{fin} = \delta_{f_k(b_1,\ldots,b_n),0} \bigotimes_{j=1}^{M} |0\rangle - i\delta_{f_k(b_1,\ldots,b_n),1} \bigotimes_{j=1}^{M} |1\rangle.$$

The prisoner P_j can therefore determine whether $f_k\left(b_1, \ldots, b_n\right)$ is 0 or 1, by measuring his qubit. Since it holds

$$\sum_{j=1}^{M} b_j = B \cdot 2^k + b \cdot 2^{k-1} + r_{k-1}^j = \left(B \cdot 2 + b\right) \cdot 2^{k-1} + r_{k-1}^j,$$

we get from formula (3.2),

$$R_j\left(k\right) = B \cdot 2 + b.$$

Using the relation from formula (3.3), we have

$$f_k\left(b_1, b_2, \ldots, b_n\right) = \left(B \cdot 2 + b\right) \bmod 2 = b \bmod 2.$$

Each prisoner P_j knows whether b, in the above formula is 0 or 1 by measuring his qubit. Therefore, when he updates r_{k-1}^j to r_k^j according to formula (3.1), r_k^j becomes the remainder of 2^k of the sum of numbers of prisoners. This way we have shown that the proposition holds for $t \geq 0$. □

On the basis of the above proposition, we can say that a computation problem can be solved by iterating a quantum game several times. That is, it can be meaningful in a quantum game that each party measures his qubit after each game iteration, which looks meaningless in the classical case.

4 Conclusion

There are two carriers in the quantum game. One of these is a measured result of a qubit, 0 or 1. If there would not be any other information carrier, one qubit could contain only one-bit information. However in our quantum game, two-bit information is exchanged between prisoners. This implies that there is another information carrier, which is the phase of each qubit. However, the phase difference cannot be distinguished by the simple measurement. Therefore, a certain device is necessary to do this and it is a series of unitary transformations, $\boldsymbol{J}^{*}\left(\boldsymbol{U}_A \otimes \boldsymbol{U}_B\right)\boldsymbol{J}$ in section 2, or $\boldsymbol{J}^{*}\left(\bigotimes_{j=1}^{M} \boldsymbol{U}_j\left(t\right)\right)\boldsymbol{J}$ in section 3.

In quantum games, the phase is one carrier of information and plays an important role in processing information. How we should use a series of unitary transformations as a device to extract information from the carrier, it is the most important to extract information from a qubit, and devising the series is equivalent to designing a quantum algorithm.

Acknowledgments. The author would like to thank Prof. Jozef Gruska and Prof. Hiroshi Imai for their useful advices.

References

[1] Jiangfeng Du, Hui Li, Xiaodong Xu, Mingjun Shi, Xianyi Zhou, and Rongdian Han. Multi-player and multi-choice quantum game. Los Alamos e-print archive quant-ph/0010092, 2001.
[2] Jens Eisert, Martin Wilkens, and Maciej Lewenstein. Quantum games and quantum strategies. Los Alamos e-print archive quant-ph/9806088, 1999.
[3] Jozef Gruska. *Quantum Computing*. McGraw-Hill, 1999.
[4] William Poundstone. *Prisoner's Dilemma/John Von Neumann, Game Theory and the Puzzle of the Bomb*. Doubleday, 1992.

On the Power of Tissue P Systems Working in the Minimal Mode*

Shankara Narayanan Krishna and Raghavan Rama

Department of Mathematics,
Indian Institute of Technology, Madras
Chennai-600 036, India
{krishiitm,ramiitm}@hotmail.com

Abstract. In this paper, we continue the study of tissue P systems (tP systems) recently introduced by C. MartinVide et.al in [5]. We consider here only tP systems working in the minimal mode. All the results in this paper are solutions to some of the open problems listed in [5] with respect to tP systems working in the minimal mode. We obtain two characterizations of recursively enumerable languages: tP systems having 2 cells and 3 states as well as tP systems having 4 cells and 2 states generate $PsRE$. We also show that 2 cells and 2 states are sufficient for generating $E0L$ languages, whereas 3 cells and 2 states are sufficient for generating $ET0L$ languages.

1 Introduction

The P systems are computing models inspired from the structure and the functioning of the living cells [1,6]. A P system consists of a *membrane structure* which is a three dimensional structure of vesicles, all of them placed in a main vesicle, delimited by a *skin* membrane. In the compartments defined by these membranes (vesicles), there are placed multisets of objects. The multisets of objects can be interpreted as chemical compounds swimming in the regions delimited by the membranes. There are evolution rules governing the modification of these objects in each membrane; the objects can also pass through the membranes or leave the system. In each time unit, all objects in a compartment which can evolve by the rules associated with that compartment have to evolve. In this way, we get *transitions* from a *configuration* to the next one. A sequence of transitions constitutes a *computation*; with each computation, we associate a *result*, the number of objects sent out of the system during the computation.

Thus, a P system is a computing device which abstracts from a single cell structure and functioning. But in most cases, since cells live together and are associated with tissues and organs, inter-cellular communication becomes an essential feature. This communication is done through the protein channels established among the membranes of the neighboring cells [4]. This has been the main motivation for the introduction of tP systems [5].

* This work was supported partially by a Project Sanction No. DST/MS/124/99, DST, India.

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 208–219, 2002.

tP systems are also motivated from the way neurons cooperate. A neuron has a body containing a nucleus; their membranes are prolonged by two classes of fibres: the *dendrites* which form a filamentary bush around the body of the neuron, and the *axon*, a unique, long filament which ends in a filamentous bush. Each of the filaments from the end of the axon is terminated with a small bulb.

Neurons process impulses in the complex net established by *synapses*. A synapse is a contact between an endbulb of a neuron and the dendrites of another neuron. The transmission of impulses from one neuron to another one is done in the following way: A neuron will be *fired* only if it gets sufficient excitation through its dendrites. After firing a neuron and having it excited, there is a small interval of time necessary to synthesize the impulse to be transmitted to the neighboring neurons through the axon; also there is a small interval of time necessary for the impulse to reach the endbulbs of the axon. The inputs to a neuron can also be from various sensory areas, like the eyes, the skin and the muscles, in addition to the impulses they get from other neurons. The neuron synthesizes an impulse which is transmitted to the neurons to which it is related by synapses; the synthesis of an impulse and its transmission to adjacent neurons are done according to certain *states* of the neuron.

These observations have been made use of while defining a tP system [5]. A tP system consists of several cells, related by protein channels. The term *synapses* is used for referring to these channels. Each cell has a state from a given finite set of states and can process multisets of objects represented by symbols from a given alphabet. The standard rules are of the form $sM \to s'M'$ where s, s' are states and M, M' are multisets of symbols. Some of the elements are marked with an indication "go" and this means they have to leave the cell immediately and pass to the cells to which we have direct links through synapses. This communication can be done in a replicative manner or non-replicative manner. In this paper, we use only the *minimal way* of applying rules $sM \to s'M'$: we can apply such a rule only to one occurrence of M. As in usual P systems, we start from an initial configuration, and allow the system to proceed until reaching a halting configuration, where no further rule can be applied. A particular cell is designated as the output cell, and in its rules $sM \to s'M'$, the indication "out" is allowed; the symbols marked with "out" are sent out of the system, contributing to the result of the computation.

2 Mathematical Prerequisites

Let $\mathbf{N}$ denote the set of natural numbers. A *multiset* over a set X is a mapping $M : X \longrightarrow \mathbf{N} \cup \{\infty\}$. For $a \in X, M(a)$ is called the multiplicity of a in the multiset M. The *support* of M is the set $supp(M) = \{a \in X \mid M(a) > 0\}$. We can write a multiset M of finite support, $supp(M) = \{a_1, \ldots, a_n\}$ in the form $\{(a_1, M(a_1)), \ldots, (a_n, M(a_n))\}$. We can also represent this multiset by the string $w(M) = a_1^{M(a_1)} \ldots a_n^{M(a_n)}$, as well as by any permutation of $w(M)$.

For two multisets M_1, M_2 over the same set X we say that M_1 is included in M_2, and we write $M_1 \subseteq M_2$ if $M_1(a) \leq M_2(a)$ for all $a \in X$. The union of

M_1, M_2 is the multiset $M_1 \cup M_2$ defined by $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$. We define here the difference $M_2 - M_1$ of two multisets only if $M_1 \subseteq M_2$ and this is the multiset defined by $(M_2 - M_1)(a) = M_2(a) - M_1(a)$.

An *alphabet* is a finite nonempty set of symbols. For an alphabet V, we denote by V^* the set of all strings of symbols in V. The empty string is denoted by λ. The length of a string $x \in V^*$ (the number of symbol occurrences in x)is denoted by $|x|$. The number of occurrences of a given symbol $a \in V$ in a string x is denoted by $|x|_a$. If $V = \{a_1, a_2, \ldots, a_k\}$, then the *Parikh mapping* of V is $\Psi_V : V^* \longrightarrow N^k$, defined by $\Psi_V(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_k})$. If L is a language, then its *Parikh set* is defined by $PsL = \{\Psi_V(w) \mid w \in L\}$.

3 Language Theory Prerequisites

In the proof from the next section, we need the notion of a *matrix grammar with appearance checking*. Such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \rightarrow x_1, \ldots, A_n \rightarrow x_n), n \geq 1$, of context-free rules over $N \cup T$ with $A_i \in N, x_i \in (N \cup T)^*$, and F is a set of occurrences of rules in M. We say that N is the non terminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices.

For $w, z \in (N \cup T)^*$, we write $w \Longrightarrow z$ if there is a matrix $(A_1 \rightarrow x_1, \ldots, A_n \rightarrow x_n)$ in M and strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leq i \leq n$, either (1) $w_i = w'_i A_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or (2) $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \rightarrow x_i$ appears in F. The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied - one says that these rules are applied in the *appearance checking* mode.

The language generated by G is $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use only grammars with $F = \emptyset$, the generated family is denoted by MAT.

We denote by CF, CS, RE the families of context-free, context-sensitive and recursively enumerable languages respectively. It is known that $CF \subset MAT \subset MAT_{ac} = RE$, all inclusions being proper.

A matrix grammar $G = (N, T, S, M, F)$ is said to be in the *binary normal form*, if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms:

1. $(S \rightarrow XA)$, with $X \in N_1, A \in N_2$
2. $(X \rightarrow Y, A \rightarrow x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$
3. $(X \rightarrow Y, A \rightarrow \#)$, with $X, Y \in N_1, A \in N_2$
4. $(X \rightarrow \lambda, A \rightarrow x)$, with $X \in N_1, A \in N_2$ and $x \in T^*$

Moreover, there is only one matrix of type one and F consists exactly of all rules $A \rightarrow \#$ appearing in matrices of type 3; $\#$ is called a trap symbol, because once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

According to Lemma 1.3.7 in [2], for every matrix grammar, there is an equivalent matrix grammar in binary normal form. For an arbitrary matrix grammar $G = (N, T, S, M, F)$, let us denote by $ac(G)$ the cardinality of the set $\{A \in N \mid A \to \alpha \in F\}$. In [3], it was proved that each recursively enumerable language can be generated by a matrix grammar G such that $ac(G) \leq 2$. We say that this is the strong binary normal from for matrix grammars.

Now we pass on to defining some basic types of L systems. Basically, a E0L system is a context-free pure grammar with parallel derivations : $G = (V, T, \omega, R)$, where V is an alphabet, $T \subseteq V$ is the terminal alphabet, $\omega \in V^*$ (axiom), and R is a finite set of rules of the form $a \to v$ with $a \in V, v \in V^*$, such that for each $a \in V$ there is at least one rule $a \to v$ in R (we say R is *complete*). For $w_1, w_2 \in V^*$ we write $w_1 \Longrightarrow w_2$ if $w_1 = a_1 \dots a_n, w_2 = v_1 \dots v_n$ for $a_i \to v_i \in R, 1 \leq i \leq n$. The generated language is $L(G) = \{x \in T^* \mid \omega \Longrightarrow^* x\}$. The family of languages generated by E0L systems is denoted by *E0L*. A *tabled* E0L system, abbreviated ET0L, is a system $G = (V, T, \omega, R_1, \dots, R_n)$, such that each triple (V, T, ω, R_i) is an E0L system; each R_i is called a *table*, $1 \leq i \leq n$. The generated language is defined by

$$L(G) = \{x \in T^* \mid \omega \Longrightarrow_{R_{j_1}} w_1 \Longrightarrow_{R_{j_2}} w_2 \dots \Longrightarrow_{R_{j_m}} w_m = x, \text{ where } m \geq 0, 1 \leq j_i \leq n, 1 \leq i \leq m\}.$$

(Each derivation step is performed by rules of the same table.) The family of languages generated by a ET0L system is denoted by *ET0L*.

It is known that $CF \subset E0L \subset ET0L \subset CS$. Moreover, *E0L* is incomparable with *MAT*.

In the sequel we will make use of the following normal form for ET0L systems. Each language $L \in ET0L$ can be generated by an ET0L system $G = (V, T, w, R_1, R_2)$ having only two tables. Moreover, from the proof of Theorem V.1.3 in [7], we see that any derivation with respect to G starts by several steps of R_1, then R_2 is used exactly once, and the process is iterated; the derivation ends by using R_2.

4 Tissue P Systems

In this section, we introduce the definition of tP systems, as given in [5].
A *tissue P system* of degree $m \geq 1$ (the degree of a system is the number of cells in the system) is a construct

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_{out}),$$

where

1. O is a finite non-empty alphabet;
2. $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$ (synapses among cells); if $(i, j) \in syn$, then j is a *successor* of i and i is an *ancestor* of j;
3. $i_{out} \in \{1, 2, \dots, m\}$ indicates the *output cell*;

4. $\sigma_1, \ldots, \sigma_m$ are *cells*, of the form

$$\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i), 1 \leq i \leq m,$$

where:
(a) Q_i is a finite set of *states*;
(b) $s_{i,0} \in Q_i$ is the *initial state*;
(c) $w_{i,0} \in O^*$ is the *initial multiset* of objects;
(d) P_i is a finite set of *rules* of the form $sw \to s'xy_{go}z_{out}$, where $s, s' \in Q_i$, $w, x \in O^*$, $y_{go} \in (O \times \{go\})^*$ and $z_{out} \in (O \times \{out\})^*$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, 2, \ldots, m\}$ different from i_{out}.

A tP system as above is said to be cooperative if it contains at least a rule $sw \to s'x$, such that $|w| > 1$, and non-cooperative in the opposite case.

Let $O_{go} = \{(a, go) \mid a \in O\}, O_{out} = \{(a, out) \mid a \in O\}$, and $O_{tot} = O \cup O_{go} \cup O_{out}$. For $s, s' \in Q_i$, $x \in O^*, y \in O^*_{tot}$, $sx \Longrightarrow_{min} s'y$ iff $sw \to s'w' \in P_i$, $w \subseteq x$, and $y = (x - w) \cup w'$. In the *min* mode, only one occurrence of the multiset from the left hand side of a rule is processed and replaced by the multiset from the righthand of the rule. The multiset w' from a rule $sw \to s'w'$ contains symbols from O, but also symbols of the form $(a, go), (a, out)$. Such symbols will be sent to the cells related by synapses to the cell σ_i where the rule $sw \to s'w'$ is applied, according to the three modes:

- *repl:* each symbol a, for (a, go) appearing in w' is sent to each of the cells σ_j such that $(i, j) \in syn$;
- *one*: all symbols a appearing in w' in the form (a, go) are sent to one of the cells σ_j such that $(i, j) \in syn$, nondeterministically chosen.
- *spread:* the symbols a appearing in w' in the form (a, go) are nondeterministically distributed among the cells σ_j such that $(i, j) \in syn$.

Any $m-$tuple of the form $(s_1w_1, \ldots, s_mw_m)$ with $s_i \in Q_i$ and $w_i \in O^*$ is called a configuration of Π; thus, $(s_{1,0}w_{1,0}, \ldots, s_{m,0}w_{m,0})$ is the initial configuration of Π. Using the rules from the sets P_i, we can define transitions among the configurations of the system. During any transition, some cells can do nothing: if no rule is applicable to the available multiset of objects in the current state, then the cell waits until new objects are sent to it from other cells. Each transition lasts one time unit, and the work of the net is synchronized, the same clock marks the time for all cells. A sequence of transitions among the configurations of a tP system is called a computation of Π. A computation which ends in a configuration where no rule in no cell can be used, is called a halting computation. Assume that the tP systems sends out through the cell $\sigma_{i_{out}}$, the multiset z. Then the vector $\Psi_O(z)$, representing the multiplicities of objects from z is said to be computed by Π.

We denote by $N_{min,\beta}(\Pi), \beta \in \{repl, one, spread\}$, the set of all vectors of natural numbers generated by a tP system Π, in the mode (min, β). The family of all sets $N_{min,\beta}(\Pi)$, generated by all non-cooperative tP systems with atmost $m \geq 1$ cells, each of them using at most $r \geq 1$ states, is denoted by $NtP_{m,r}(nCoo, min, \beta), \beta \in \{repl, one, spread\}$.

Note: We do not use rules of the form $s \to s'$ or $s \to s'w$, which are unrealistic from the biological point of view.

5 The Power of Tissue P Systems

In this section, we will discuss a few results regarding the generative power of tP systems that are working in the minimal mode. In all the proofs below, we shall use the short form $sa \to s'(x, go)(y, out)$ for a rule $sa \to s'(x_1, go)(x_2, go), \ldots, (x_n, go)(y_1, out)(y_2, out)\ldots(y_m, out)$, where $x = x_1x_2\ldots x_n$ and $y = y_1y_2\ldots y_m$. First, we recall some results from [5]:

Lemma 1. *1. For all tP systems Π where each cell has atmost one successor, we have $N_{min,repl}(\Pi) = N_{min,one}(\Pi) = N_{min,spread}(\Pi)$.*
2. $NtP_{m,r}(nCoo, min, one) = NtP_{m,r}(nCoo, min, spread), \forall\ m, r \geq 1$.

In [5], one proves that $PsRE = NtP_{2,5}(nCoo, min, \beta), \beta \in \{one, spread, repl\}$ and one asks whether or not these results can be improved. Here, we answer this question in the affirmative.

Theorem 1. $PsRE = NtP_{2,3}(nCoo, min, \beta), \beta \in \{one, spread, repl\}$.

Proof. We prove only the inclusion $\subseteq$, the opposite one can be obtained as a consequence of the Turing-Church thesis. Because of the equality $PsRE = PsMAT_{ac}$, we start with a matrix grammar $G = (N, T, S, M, F)$ in strong binary normal form. So, there are only two symbols $A^{(1)}, A^{(2)} \in N_2$ used in appearance checking rules. Assume we have k matrices in M.

We construct the tP system $\Pi = (O, \sigma_1, \sigma_2, (1, 2), (2, 1), 1)$, with the alphabet

$$O = N_1 \cup N_2 \cup \{Y_0, E, H, Z\} \cup \{X_{i,j}, X_{i,0} \mid X \in N_1, 1 \leq i \leq k, 0 \leq j \leq k\}$$
$$\cup \{A_{i,j}, i \mid A \in N_2, 1 \leq i \leq k, 0 \leq j \leq k\},$$

and the following cells (the axiom multiset consists of symbols X, A corresponding to the initial matrix $(S \to XA)$ of the grammar G. XA is placed in σ_1 if $A \neq A^{(2)}$; otherwise, X is placed in σ_1 and A is placed in σ_2. Without loss of generality, assume that for the given grammar G, the initial matrix $(S \to XA)$ is such that $A \neq A^{(2)}$.

$$\sigma_1 = (\{s, s', s_1\}, \{s\}, \{XA\},$$
$$\{sX \to s'(X_{i,i}, go), s'A \to s(A_{i,i}, go) \mid m_i : (X \to Y, A \to x) \in M, 1 \leq i \leq k\}$$
$$\cup \{sX \to s(Y_2, go) \mid m_i : (X \to Y, A^{(2)} \to \#) \in M, 1 \leq i \leq k\}$$
$$\cup \{sX \to s_1Y(H, go) \mid m_i : (X \to Y, A^{(1)} \to \#) \in M, 1 \leq i \leq k\}$$
$$\cup \{s_1A^{(1)} \to sZ, sZ \to sZ, s_1H \to s\} \cup \{sY_0 \to sY \mid Y \in N_1\}$$
$$\cup \{sX_{i,0} \to s'Y_0 \mid m_i : (X \to Y, A \to x) \in M, 1 \leq i \leq k\}$$
$$\cup \{s'A_{i,1} \to sz(y, out)(w^{(2)}, go) \mid m_i : (X \to Y \text{or} X \to \lambda, A \to xy) \in M,$$
$$1 \leq i \leq k, y \in T^*, z \in (N_2 \backslash A^{(2)})^*, w^{(2)} \in (A^{(2)})^*, \text{ and } x \text{ is obtained by}$$
$$\text{catenating symbols of } z, w^{(2)} \text{ in some order}\}$$

$$\cup \{sX_{i,0} \to s' \mid m_i : (X \to \lambda, A \to x) \in M, 1 \le i \le k\}$$
$$\cup \{si \to sA_{i,1} \mid m_i : (X \to Y, A \to x) \in M, 1 \le i \le k\}$$
$$\cup \{sA_{i,1} \to sZ, sE \to s(E, go)\}$$
$$\sigma_2 = \{\{s, s', s_2\}, \{s\}, \lambda,$$
$$\cup \{sX_{i,j} \to s'X_{i,j-1}, s'A_{i,k} \to sA_{i,k-1} \mid X \in N_1, A \in N_2, j > 1, k > 2\}$$
$$\cup \{s'A_{i,2} \to s(i, go), sX_{i,1} \to s(X_{i,0}, go) \mid A \in N_2, X \in N_1\}$$
$$\cup \{sA_{i,j} \to sZ \mid 1 \le i, j \le k, A \in N_2\} \cup \{s_2A^{(2)} \to s_2Z\}$$
$$\cup \{sY_2 \to s_2Y(E, go) \mid Y \in N_1\} \cup \{sY \to s(Y, go) \mid Y \in N_1\}$$
$$\cup \{sZ \to sZ, s_2Z \to s_2Z, s_2E \to s, sH \to s(H, go)\}$$

Assume that we apply the rules $sX \to s'(X_{i,i}, go)$ and $s'A \to s(A_{j,j}, go)$ in cell 1. The symbols $X_{i,i}$ and $A_{j,j}$ are sent to cell 2, where under the control of states s, s', one alternately decreases by one the second components of the subscripts. During this time, the first cell has no symbols to be processed and remains inactive. The computation continues in this way till one of the symbols from cell 2 reaches a subscript with the second component being equal to 2. We have three cases:

Case 1: If $i < j$, then in the second cell, we use the rule $sA_{i,j} \to sZ$ and the computation never ends.

Case 2: If $i > j$, then the rule $s'A_{i,2} \to s(i, go)$ will be applied in cell 2 before $X_{i,2}$ is obtained. In this case, in the next step, rules $si \to sA_{i,1}$ and $sX_{i,j} \to s'X_{i,j-1}, j > 2$ will be applied in cells 1 and 2. In the next step, there are no rules to be applied in cell 2 and the computation loops in cell 1 by the rule $sA_{i,1} \to sZ$.

Case 3: If $i = j$, then the rules $s'A_{i,2} \to s(i, go), sX_{i,1} \to s(X_{i,0}, go)$ are applied in order in cell 2. In cell 1 the rule $si \to sA_{i,1}$ is applied when the rule $sX_{i,1} \to s(X_{i,0}, go)$ is applied in cell 2. In the next step, we have both symbols $A_{i,1}$ and $X_{i,0}$ in cell 1. Now, the rules $sX_{i,0} \to s'Y_0$ and $s'A_{i,1} \to sz(y, out)(w^{(2)}, go)$ should be applied in order in cell 1 to get the correct result. If the rule $sA_{i,1} \to sZ$ is used before $sX_{i,0} \to s'Y_0$, the computation will loop. Note that the rule $s'A_{i,1} \to sz(y, out)(w^{(2)}, go)$ sends to cell 2 symbols of N_2 which are of the form $A^{(2)}$.

In this way, we correctly simulate the matrices of G without appearance checking rules.

Assume now that we start in cell 1 by using a rule $sX \to s(Y_2, go)$. The symbols $A^{(2)}$ if any, will be present only in cell 2. In cell 2, we apply rules $sY_2 \to s_2Y(E, go)$ and $s_2A^{(2)} \to s_2Z$ if $A^{(2)}$ is present. If $A^{(2)}$ is not present in cell 2, then cell 2 waits for the E to return back from cell 1. Then the rule $s_2E \to s$ is applied.

If we start in cell 1 by applying a rule $sX \to s_1Y(H, go)$ and if the corresponding symbol $A^{(1)}$ is present in cell 1, the computation loops. Otherwise, cell 1 waits for H to return from cell 2 and applies the rule $s_1H \to s$. In this way, one correctly simulates an appearance checking rule. Consequently, all derivations in G

can be simulated in Π and conversely, all halting computations in Π correspond to terminal derivations in G. One can easily see that $\Psi_T(L(G)) = N_{min,\beta}(\Pi)$, for all $\beta \in \{one, spread, repl\}$.

Theorem 2. $PsRE = NtP_{4,2}(nCoo, min, \beta), \beta \in \{one, spread\}$.

Proof. As above, we start from a matrix grammar $G = (N, T, S, M, F)$ in the strong binary normal form having k matrices. For $i = 1, 2$ let $A^{(i)}$ denote the symbols of N_2 which appear in matrices of type 3. Let $N_2' = N_2 \backslash (A^{(1)} \cup A^{(2)})$.

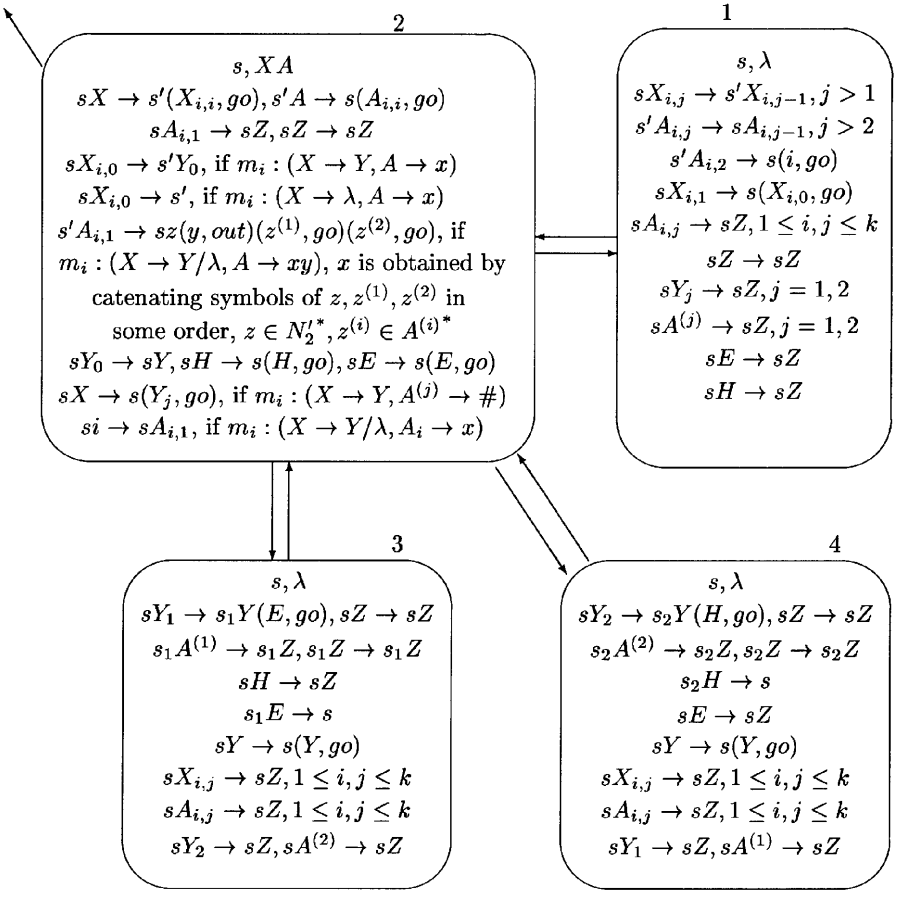


Fig. 1. The tP system from the proof of Theorem 2

The axiom multiset consists of symbols X, A corresponding to the initial matrix $(S \to XA)$ of the grammar G. XA is placed in σ_2 if $A \neq A^{(i)}$. If $A \neq A^{(1)}$, X is placed in σ_2 and A is placed in σ_3. Otherwise, X is placed in σ_2 and A is

placed in σ_4. Assume that for the given grammar G, the initial matrix $(S \to XA)$ is such that $A \neq A^{(i)}$, so that we have XA in σ_2.

We construct the tP system

$$\Pi = (O, \sigma_1, \sigma_2, \sigma_3, \sigma_4, (1,2), (2,1), (2,3), (3,2), (2,4), (4,2), 2),$$

with the alphabet

$$O = N_1 \cup N_2 \cup \{E, H, Z\} \cup \{X_{i,j}, X_{i,0} \mid X \in N_1, 1 \leq i \leq k, 0 \leq j \leq k\}$$
$$\cup \{A_{i,j} \mid A \in N_2, 1 \leq i \leq k, 0 \leq j \leq k\},$$

and cells as in Figure 1.

Since we have the axiom XA in cell 2, the symbol A corresponds to a type 2 or 4 matrix. The simulation of type 2 and 4 matrices are similar to Theorem 1.

To simulate appearance checking rules, we apply rules $sX \to s(Y_j, go)$ in cell 2. For the computation to proceed correctly, Y_1 has to be sent to cell 3 and Y_2 has to be sent to cell 4. In cell 3, the rule $sY_1 \to s_1 Y(E, go)$ is applied. If any $A^{(1)}$ is present, the rule $s_1 A^{(1)} \to s_1 Z$ is applied and the computation loops. Otherwise, the rules $s_1 E \to s, sY \to s(Y, go)$ are applied and the computation continues in cell 2.

Consequently, all derivations in G can be simulated in Π and conversely, all halting computations in Π correspond to terminal derivations in G. One can easily see that $\Psi_T(L(G)) = N_{min,\beta}(\Pi)$, for all $\beta \in \{one, spread, repl\}$.

Theorem 3. $PsE0L \subseteq NtP_{2,2}(nCoo, min, \beta), \beta \in \{one, spread, repl\}$.

Proof. Let $G = (V, T, w, R)$ be an E0L system. We construct the tP system $\Pi = (V \cup \{E, F, K, L, N, Z\}, \sigma_1, \sigma_2, (1,2), (2,1), 2)$, and cells as in Figure 2.

By using the rule $sa \to s(x, go)(N, go)$ in cell 1 associated with $a \to x \in R$, we simulate a derivation step in G; when all symbols are rewritten, we can use the rule $sa \to s'(x, go)(E, go)$, otherwise, the trap symbol Z is introduced and the computation will never stop.

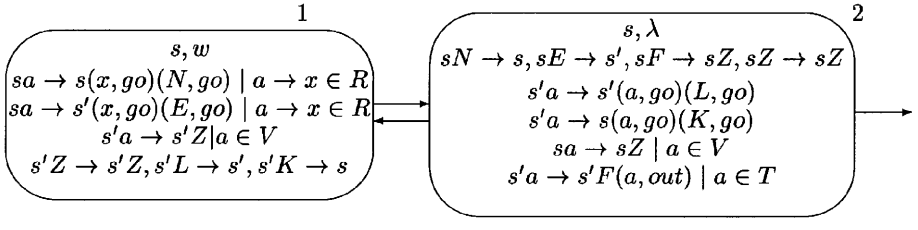


Fig. 2. The tP system from the proof of Theorem 3

While applying $sa \to s(x, go)(N, go)$ in cell 1, the rule $sN \to s$ should be applied in cell 2; otherwise, the rule $sa \to sZ$ will be applied and the computation will loop. Similarly, during the last step in cell 1, the symbol E is introduced and the rule $sE \to s'$ is applied in cell 2. Once the state s' is introduced in cell 2, the symbols $a \in V$ can either be sent out or sent back to cell 1 for

further computations. For this purpose, the rules $s'a \to s'F(a, out)$ or $s'a \to s'(a, go)(L, go)$ are applied in cell 2. The rule $s'a \to s(a, go)(K, go)$ is applied when all symbols $a \in V$ are sent back to cell 1; otherwise, the trap symbol Z is introduced and the computation never stops. For the simulation to proceed correctly, either the rules $s'a \to s'(a, go)(L, go), s'a \to s(a, go)(K, go)$ should be applied to the symbols $a \in V$ in cell 2, or the rule $s'a \to s'(a, out)F$ should be applied. If one uses both the rules $s'a \to s'(a, go)(L, go), s'a \to s'(a, out)F$ to a set of symbols $a \in V$, then the computation will loop. (If rules $s'a \to s'(a, go)(L, go), s'a \to s'(a, out)F$ alone are applied, the looping takes place due to the rule $s'a \to s'Z$ in cell 1; if all three rules $s'a \to s'(a, go)(L, go), s'a \to s(a, go)(K, go), s'a \to s'(a, out)F$ are applied, then the looping takes place due to the rule $sF \to sZ$ in cell 2.) Thus, we have the equality $\psi_T(L(G)) = N_{min,\beta}(\Pi)$, for all $\beta \in \{repl, one, spread\}$.

Corollary 1. $NtP_{2,2}(nCoo, min, \beta) - PsMAT \neq \emptyset, \beta \in \{one, spread, repl\}$.

Theorem 4. $PsET0L \subseteq NtP_{3,2}(nCoo, min, \beta), \beta \in \{one, spread\}$.

Proof.

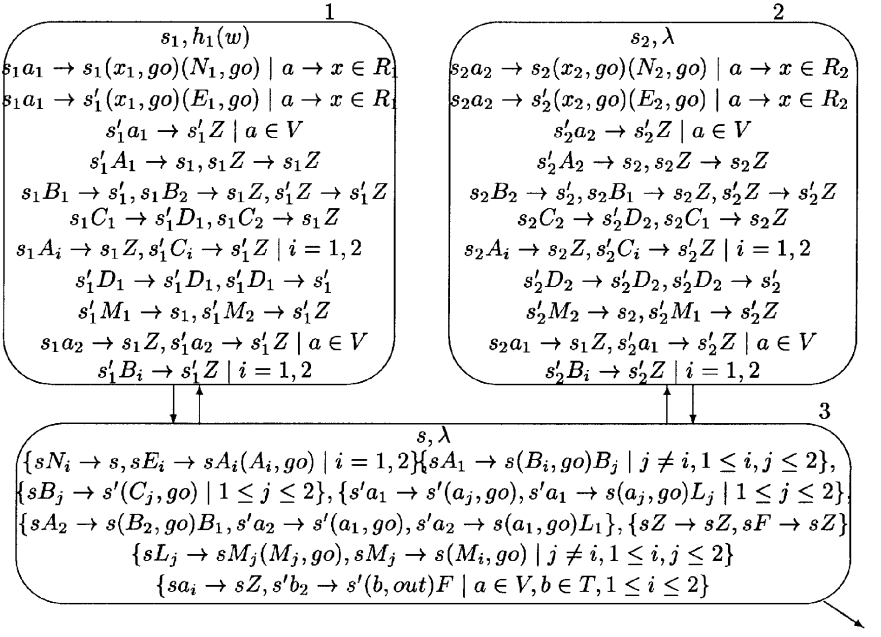


Fig. 3. The tP system from the proof of Theorem 4

Let $G = (V, T, w, R_1, R_2)$ be an ET0L system in the normal form. We construct the tP system $\Pi = (O, \sigma_1, \sigma_2, \sigma_3, (1, 3), (3, 1), (2, 3), (3, 2), 3)$ with the alphabet

$$O = V \cup \{a_1, a_2 \mid a \in V\} \cup \{A_i, B_i, C_i, D_i, E_i, L_i, M_i, N_i, F, Z \mid i = 1, 2\},$$

and the cells as in Figure 3 (h_1, h_2 are the morphisms which replace each $a \in V$ by a_1, a_2 respectively)

In the initial configuration, we have the string $h_1(w)$ in cell 1, where w is the axiom. In cells 1 and 2, we simulate the actions of tables 1 and 2 respectively. By using rules of the form $s_i a_i \to s_i(x_i, go)(N_i, go)$ in cells $i, i = 1, 2$, associated with $a \to x \in R_i$, we simulate a derivation step in table i; when all symbols are rewritten, we can use the rule $s_i a_i \to s_i'(x_i, go)(E_i, go)$, otherwise, the trap symbol Z is introduced and the computation will never stop. Corresponding to the above rules in cells $i, i = 1, 2$, rules $sN_i \to s$ and $sE_i \to sA_i(A_i, go)$ must be applied in cell 3; otherwise, the rule $sa_i \to sZ$ will be applied, leading to an endless computation. The symbol A_i must be sent to cell $i, i = 1, 2$ if table i is the most recently table; otherwise, the computation never halts.

Assume that A_i goes to cell i. In cell i, the rule $s_i' A_i \to s_i$ is applied. Now, the cells i, j are in states s_i, s_j respectively. If table 1 was the last table simulated, the symbol A_1 will be present in cell 3. The rule $sA_1 \to s(B_i, go)B_j$ can now be applied. The symbol B_i must be sent to the cell i, to avoid endless computation. The symbol B_i decides the next table to be simulated. If the above rule is applied, then table j is simulated next, otherwise, table i is simulated. In cell i, the rule $s_i B_i \to s_i'$ is applied, blocking the simulation of table i. The computation is continued in cell 3 by applying $sB_j \to s'(C_j, go)$. The symbol C_j if sent to cell i loops the computation; so C_j is sent to cell j where the rule $s_j C_j \to s_j' D_j$ is applied.

In the next few steps, rules $s_j' D_j \to s_j' D_j$ and $s' a_1 \to s'(a_j, go)$ must be applied in cells j and 3. Corresponding to the last symbol $a_1 \in h_1(V)$ in cell 3, the rules $s_j' D_j \to s_j' D_j$ and $s' a_1 \to s(a_j, go)L_j$ must be applied in cells j and 3, and in the next step, rules $s_j' D_j \to s_j'$ and $sL_j \to sM_j(M_j, go)$ should be applied. Otherwise, the computation will never halt since the rule $s_j' a_j \to s_j' Z$ will be applied in cell j.

Note here that while applying rules $s' a_1 \to s'(a_j, go)$ in cell 3, if the subscript j of the symbol a_j is not kept constant in all steps, i.e., if one applies rules $s' a_1 \to s'(a_1, go)$ and $s' b_1 \to s'(b_2, go)$ in two different steps, then the computation never halts. The subscripts of the symbols $a \in V$ in $s' a_1 \to s'(a_j, go)$ must be j if the rule $sA_1 \to s(B_i, go)B_j$ has been applied in an earlier step. Also, the symbols a_j must be sent to exactly one cell j; otherwise, the computation will loop. So, to obtain a correct result, the subscripts of the symbols must be a fixed number say, j in accordance to the rule $sA_1 \to s(B_i, go)B_j$ and, they must be sent only to cell j.

The symbol M_j restores the state of cell j to s_j and then, the simulation of table j can be started. To restore the state of cell i to s_i, the rules $sM_j \to s(M_i, go)$ and $s_i' M_i \to s_i$ are applied in cells 3 and i. This, way a computation can be iterated.

The computation can be stopped only after simulating table 2. The two choices that are left after simulating table 2 once are: (1) Simulate table 1 or

(2) Stop the computation. To stop the computation and send out all symbols, the rule $s'a_2 \to s'(a, out)F$ is applied to all $a \in T$. If some symbols $a \in V \backslash T$ remain, the computation will never halt. (If in cell 3, the rules $s'a_2 \to s'(a, out)F$ and $s'a_2 \to s'(a_1, go)$ are the only ones used with respect to state s', then the computation loops in cell 1; if rules $s'a_2 \to s'(a, out)F, s'a_2 \to s'(a_1, go)$ and $s'a_2 \to s(a_1, go)L_1$ are applied, then the computation loops in cell 3 due to the rule $sF \to sZ$. So, to avoid endless computation, either the rules $s'a_2 \to s'(a_1, go), s'a_2 \to s(a_1, go)L_1$ alone must be applied to symbols a_2 in cell 3 or, the rule $s'a_2 \to s'(a, out)F$ alone must be applied to all the a_2 when $a \in T$.)

This shows that $ET0L$ is contained in the family $NtP_{3,2}(nCoo, min, \beta), \beta \in \{one, spread\}$.

6 Conclusion

We have investigated the power of tissue P systems working in the minimal mode using a small number of (≤ 4) cells. We have solved the open problems listed in [5] with respect to tP systems working in the minimal mode. We do not know whether the results given in this paper are optimal; but we conjecture that a characterization of RE cannot be obtained in the minimal mode by using a system having m cells and r states where $m < 2, r < 3$ or $m < 4, r < 2$.

References

1. C. S. Calude and Gh. Păun, *Computing with Cells and Atoms : An Introduction to quantum, DNA and membrane computing*, Taylor & Francis, London, 2001.
2. J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
3. R. Freund, Gh. Păun, On the number of nonterminals in graph-controlled, programmed, and matrix grammars, *Lecture Notes in Computer Science*, 2055, Springer-Verlag, 2001,214–225.
4. W. R. Loewenstein, *The Touchstone of Life. Molecular Information,Cell Communication, and the Foundations of Life,* Oxford Univ. Press, New York, Oxford, 1999.
5. C. MartinVide, Gh. Păun, J. Pazos and A. Rodriguez Paton, Tissue P Systems, *Turku Center for Computer Science-TUCS Report* No **421**, www.tucs.fi.
6. Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
7. G. Rozenberg,A. Salomaa, Eds., *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.

Reversible Computation in Asynchronous Cellular Automata

Jia Lee[1], Ferdinand Peper[1], Susumu Adachi[1], Kenichi Morita[2], and Shinro Mashiko[1]

[1] Communications Research Laboratory, Nanotechnology Group, 588-2 Iwaoka, Iwaoka-cho, Nishi-ku, Kobe 651-2492, Japan
{lijia, peper, sadachi, mashiko}@crl.go.jp
[2] Hiroshima University, Higashi-Hiroshima 739-8527, Japan
morita@iec.hiroshima-u.ac.jp

Abstract. Reversible computation has attracted much attention over the years, not only for its promise for computers with radically reduced power consumption, but also for its importance for Quantum Computing. Though studied extensively in a great variety of synchronous computation models, it is virtually unexplored in an asynchronous framework. Particularly suitable asynchronous models for the study of reversibility are asynchronous cellular automata. Simple yet powerful, they update their cells at random times that are independent of each other. In this paper, we present the embedding of a universal reversible Turing machine (RTM) in a two-dimensional self-timed cellular automaton (STCA), a special type of asynchronous cellular automaton, of which each cell uses four bits to store its state, and a transition on a cell accesses only these four bits and one bit of each of the four neighboring cells. The embedding of a universal RTM on an STCA requires merely four rotation-symmetric transition rules, which are bit-conserving and locally reversible. We show that the model is globally reversible.

1 Introduction

In the quest to develop increasingly powerful and energy-efficient computers, reversibility has played a key role. As physical phenomena on the microscale are reversible, computers that test the limits of physics must conduct their computations in a reversible way, lest the microstate information about their history is transformed into some uncontrolled and inaccessible form, such as thermal entropy [5] of the system and its surroundings [1]. Reversibility is also a precondition for Quantum Computation. Due to this, the operations of a quantum computer can be expressed in terms of unitary transformations.

In all research up to now on reversible computing, it is implicitly assumed that the underlying system is synchronously timed, i.e., that there is a central clock in accordance to which all logic elements switch in lock-step. For example, a Fredkin gate [2] only works correctly if all its inputs arrive at roughly the same time. As asynchronous systems carry a high degree of nondeterminism due to the

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 220–229, 2002.

randomness by which events in them are timed, they are thought to be incompatible with the backward determinism accompanying reversible computing. Still, the fact is that most physical interactions on the microscale are asynchronous. As with reversibility, asynchronicity tends to reduce power consumption and heat dissipation, be it for different reasons: logic elements in an asynchronous system go into a sleeping state if they have no work to do, unlike in synchronous systems, where idle logic elements have to engage in dummy switching when receiving clock signals. In the context of Quantum Computing, the study of reversibility in asynchronous systems is also significant, as the absence of a clock signal allows a better isolation of a quantum system from its environment, thus improving its ability to maintain the superpositions of its states.

Synchronous cellular automata are a very popular model to conduct research on into reversibility [3,4,6,8,11]. For a cellular automaton to be reversible, the left-hand side (LHS) of its transition rules must consist of the same number of elements as on the right-hand side (RHS). This is achieved in the Billiard-Ball Model Cellular Automaton [6] by transitions rules that change the states of a block of 2×2 cells all at the same time. Alternatively, in [4,8] cells are divided in partitions, and transition rules are applied to the partitions of cells and their neighbors, such that the number of partitions in the LHS is the same as the number of partitions in the RHS. The resulting model is called a Partitioned Cellular Automaton (PCA).

In this paper we study reversible computing on asynchronous cellular automata. In such cellular automata transitions of cells are timed randomly and independently of each other. We use a special type of asynchronous cellular automaton, a so-called two-dimensional Self-Timed Cellular Automaton (STCA) [9,10], which is basically a PCA with four partitions per cell, each partition carrying one of two possible states. The proposed STCA employs four rotation-symmetric transition rules, each of which takes the four partitions of a cell and one partition of each of the four neighboring cells as arguments, and changes their states.

We simulate a universal reversible Turing machine (RTM) by constructing an infinite circuit based on a reversible logic module, a so-called Rotary Element (RE) [7], and embedding this circuit in an infinitely sized STCA. This construction [7] allows the circuit to operate in asynchronous mode, with a signal being subject to arbitrary delays, while at the same time the global transitions of the STCA are reversible. In other words, the STCA can carry out reversible computation.

This paper is structured as follows. Section 2 describes the Rotary Element and shows how a reversible Turing machine can be made in accordance with Morita's construction by wiring infinitely many Rotary Elements in an appropriate way. Section 3 describes the STCA model in more detail, followed by section 4, which shows how to embed a Rotary Element in an STCA. This shows how a reversible Turing machine can be simulated on an infinite STCA. This paper finishes with concluding remarks.

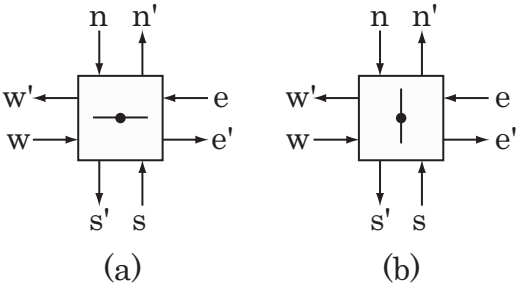


Fig. 1. (a) An RE in H-state, and (b) an RE in V-state

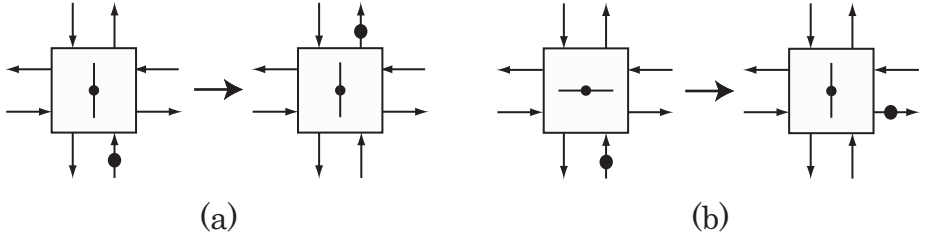


Fig. 2. RE operating on an input signal in (a) parallel case, and (b) vertical case. The signal is denoted by a token on a line

2 Rotary Elements

The *Rotary Element* (RE) introduced by Morita [7] is a reversible logic module. It has four input lines $\{n, e, s, w\}$, four output lines $\{n', e', s', w'\}$, and two states, the H-state and the V-state (see Fig. 1), which are displayed as horizontal and vertical bars respectively in the RE. When a signal, which is one-valued, arrives on one of the input lines of an RE, the RE operates on the signal, by possibly changing its state and transmitting the signal to an appropriate output line in the following way: If a signal comes from a direction parallel to the rotating bar of an RE, then it passes straight through to the opposite output line, without changing the direction of the bar (the state of the RE), as in Fig. 2(a); if a signal comes from a direction orthogonal to the rotating bar, then it turns right and rotates the bar by 90 degrees (Fig. 2(b)). Simultaneous signals on any pair of input lines of an RE are not allowed. Obviously, the functionality of an RE is reversible and conservative.

The RE is capable of universal computation, in the sense that any reversible Turing machine (RTM) can be realized by a network of an infinite number of RE modules, in which there is at most one signal moving around in the entire network at any time [7]. Thus, delays in modules or lines do not affect the correct operation of the entire circuit, i.e., this circuit can operate in an asynchronous mode, removing the need of a central clock signal [7]. An example of a circuit

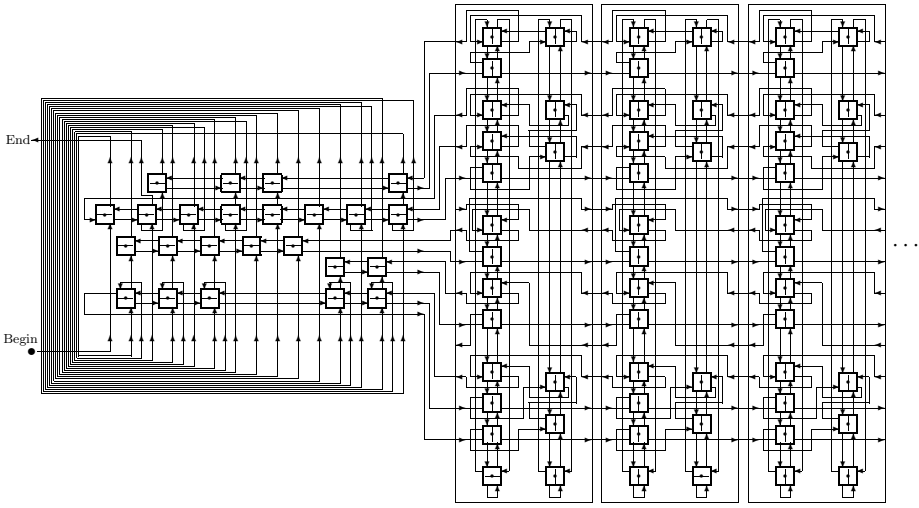


Fig. 3. The realization of a reversible Turing machine by REs. Note that each RE in the construction has its state set before the operation of the circuit starts, such as to ensure a proper initial configuration of the Turing machine. (Reprinted from [7] copyright©2001, Springer)

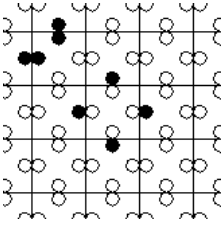


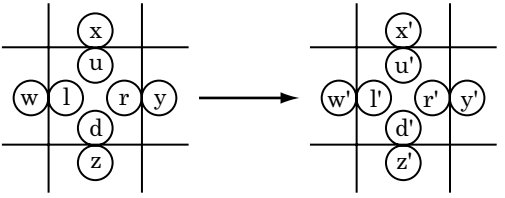
Fig. 4. Self-timed cellular space

composed of RE modules which simulates a particular RTM is given in Fig. 3, in which the partial network of RE modules on the left is used for finite-state control, and the infinite array of blocks on the right is in one-to-one correspondence with the infinite number of tape cells of the Turing machine. This circuit computes the function $f(n) = 2n$, starting with a signal arriving on input line "Begin".

3 Self-Timed Cellular Automata

A *Self-Timed Cellular Automaton* (STCA) [9] is a two-dimensional array of identical cells, each of which can be in one of 16 states. Each cell's state is partitioned into four bits in one-to-one correspondence with its neighboring cells (see Fig. 4), in which a filled circle denotes a 1-bit, and a blank circle denotes a 0-bit.

Each cell undergoes state transitions via transition rules that operate on the four bits of the cell and the nearest bit of each of its four neighbors. Thus a transition rule $f(u, r, d, l, x, y, z, w) = (u', r', d', l', x', y', z', w')$ can be depicted as follows,



where $u, u', r, r', d, d', l, l', x, x', y, y', z, z', w, w' \in \{0, 1\}$, and $(u, r, d, l, x, y, z, w) \neq (u', r', d', l', x', y', z', w')$. In STCA, transitions of cells are timed independently of each other and occur at arbitrary times. For this reason, they are asynchronous cellular automata.

Let $Conf(c)$ denote the set of all configurations over an STCA. For any $c_i, c_j \in Conf(c)$, there exists a global transition from c_i to c_j written by $c_i \to c_j$, iff c_j can be obtained by random selection and update of at least one cell in c_i at a time, and no neighboring cells in c_i can change a bit common to them to different values at the same time [10]. Moreover, global transitions in an STCA are reversible iff

$$\forall c_i, c_j, c_h \in Conf(c),\ c_i \to c_j \wedge c_h \to c_j \Longrightarrow c_i = c_h.$$

4 Embedding Rotary Elements in Self-Timed Cellular Automata

In this section, we introduce an STCA that can embed a network of Rotary Elements. The interactions between the cells are described by the transition rules in Fig. 5. Since the RHS of each rule differs from those of the other rules, the rules are locally reversible. Also, since the numbers of 1-bits in the LHS and RHS of each rule are the same, the rules are bit-conserving.

A line is implemented in our STCA by a straight path of continuous cells, all bits of which are 0. A signal is represented as a pattern of one 1-bit and seven 0-bits, as in Fig. 6(a). Applying transition rule 1 in Fig. 5 twice to this pattern moves the signal along the path of cells towards the east in Fig. 6(b). We adopt the convention that the rotation-symmetric equivalents of transition rules may also serve as transition rules. This allows, for example, the use of transition rule 1 for transmitting signals in directions towards the north, south, and west as well.

Fig. 7(a) shows a pattern that can be used as a left-turn element for signals (see Fig. 7(b)). A signal coming from the opposite direction just passes through the left-turn element as in Fig. 7(c). Using three left-turn elements, we can construct a right-turn element.

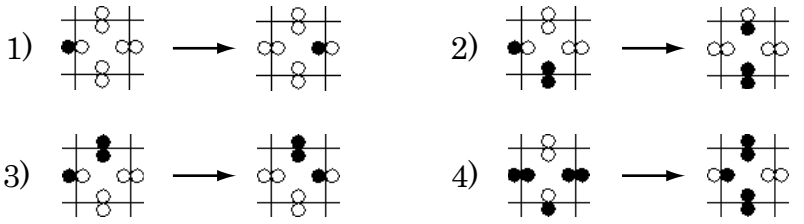


Fig. 5. Transition rules of an STCA

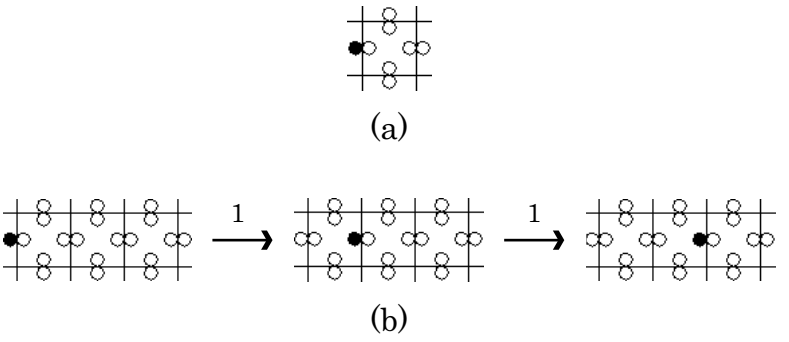


Fig. 6. (a) A signal. (b) Transmitting a signal to the east. The integer above each arrow denotes the transition rule (or its rotation-symmetric equivalents) in Fig. 5 used to update the states of the cells in the configurations

The pattern in Fig. 8(a) represents a rotation bar. A signal coming from a direction orthogonal to the bar will turn left and rotate the bar by 90 degrees as in Fig. 8(b).

A rotary element embedded in the STCA is a composition of the rotation bar and left-turn elements, as in Fig. 9. Fig. 10 illustrates the operations of an RE operating on a signal that arrives at one of its input lines at directions parallel with or orthogonal to the rotation bar. The correctness of the processing in Fig. 10 can be easily verified.

By implementing the pattern representing an RE in Fig. 9, along with the other elements mentioned above, we can lay out an infinite network of REs on our STCA and simulate an arbitrary reversible Turing machine, such as the RTM in Fig. 3. Since only one signal moves around in the entire circuit in Fig. 3, at most one cell in the entire configuration undergoes a transition at a time, and this cell is the one at which the signal arrives. In this case, the local reversibility of the transition rules results in the global transitions in our STCA to be reversible, even though transitions may be subject to arbitrary delays due to the asynchronicity of the STCA. Thus, our STCA is computationally universal, and it computes reversibly when a reversible Turing machine is embedded in its cellular space.

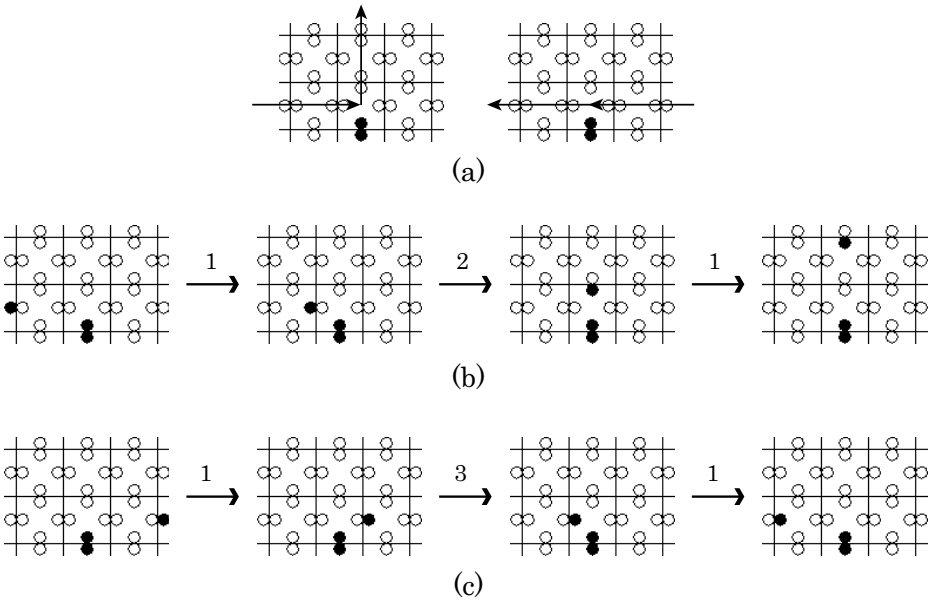


Fig. 7. (a) Patterns representing a left-turn element. The arrow indicates the direction in which a signal propagates. (b) A signal from the west turning left to the north. (c) A signal from the east passing to the west

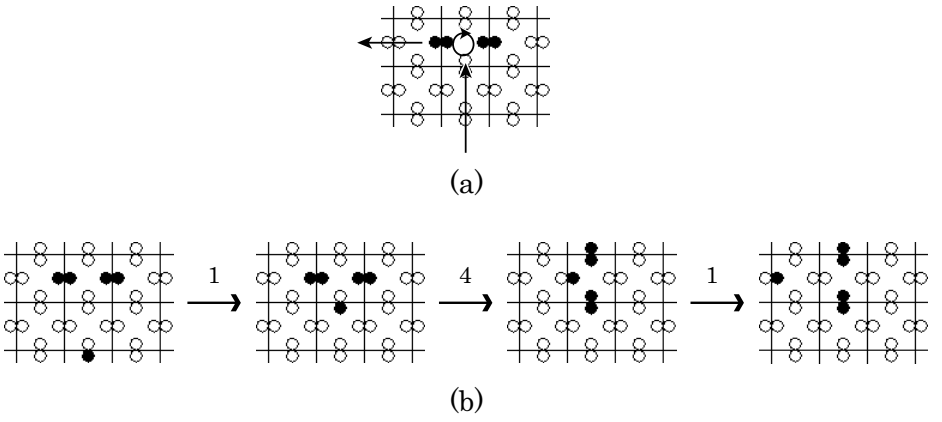


Fig. 8. (a) A rotation bar in horizontal position (H-state). (b) A signal from the south turning to the west, while setting the bar to a vertical position

5 Concluding Remarks

In this paper, we introduced a model that is capable of universal reversible computation in the sense that it can simulate a universal reversible Turing ma-

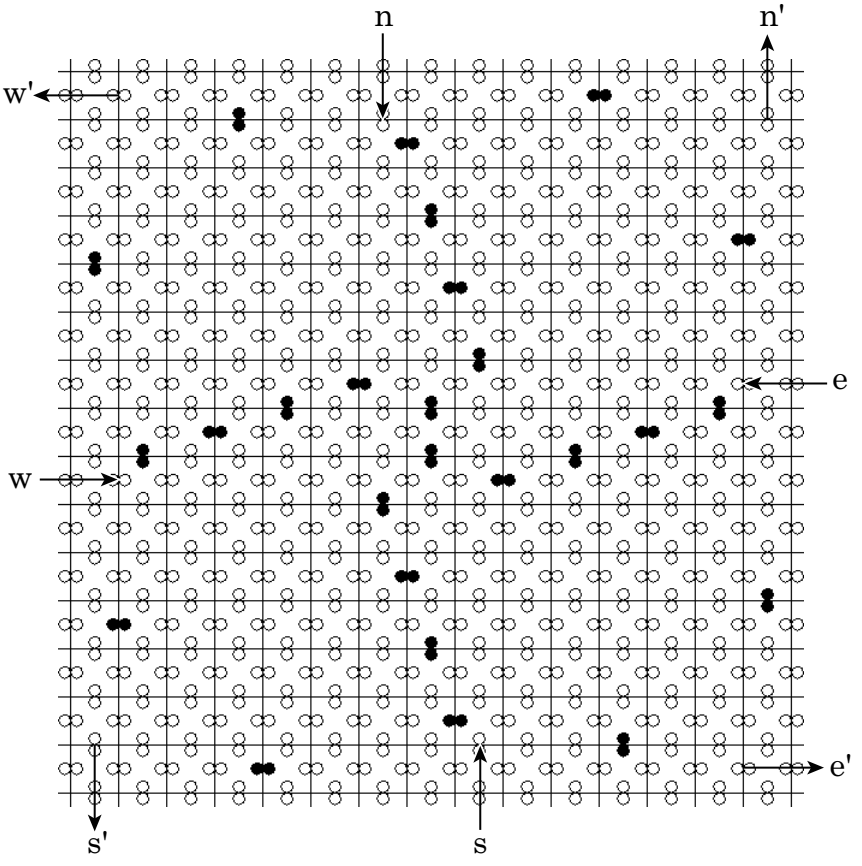


Fig. 9. An RE in V-state. Positions of the input and output paths of this RE slightly differ from the RE in Fig. 1

chine. For our implementation, we proposed a self-timed cellular automaton that uses only four rotation-symmetric transition rules to describe the interactions between its cells. Since only a single signal is used at any time to conduct computations, the ordering of the operations is strictly defined. Together with the local reversibility of the transition rules, this is sufficient to guarantee backward determinism of the model. The system is not reversible in the traditional sense, as the signal may undergo arbitrary delays at any time. In synchronous reversible systems, such behavior is not permitted, because signals that are static in two successive clock cycles, should remain so indefinitely, due to the reversibility condition.

If more than one signal at a time would be allowed in the STCA, a whole new situation would occur. Though the ordering of the operations associated with each of the individual signals would still be uniquely defined, provided the

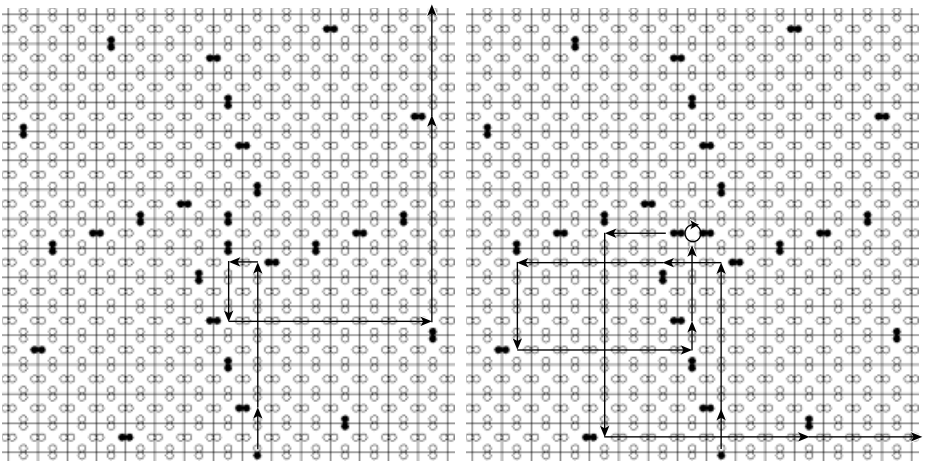


Fig. 10. Traces of a signal moving in the configuration represening an RE when the rotation bar is (a) parallel or (b) vertical

signals would not interact with each other, the ordering of the operations in the overall system would not. This brings up the question of what reversibility in asynchronous systems actually means. Does it incorporate being able to follow backwards the trace of events in a system including the order and timing in which events take place? Alternatively, can the order be interpreted in a more loose sense, for example allowing events to be delayed and allowing independent events to change their order of occurrence with respect to each other? Finally, for the construction of reversible asynchronous computers an important question is how the picture changes if events interact with each other. We hope this paper initiates more discussion about these intriguing questions.

Acknowledgements. We would like to thank Prof. Nobuyuki Matsui and Teijiro Isokawa at Himeji Institute of Technology, Dr. Katsunobu Imai at Hiroshima University for their useful discussions. We also thank the anonymous referees for their valuable comments.

References

1. Frank, M., Vieri, C., Ammer, M.J., Love N., Margolus, N.H., Knight T. Jr.: A scalable reversible computer in silicon. In: Calude, C.S., Casti, J., Dinneen, M.J. (eds.): Unconventional Models of Computation, Springer-Verlag, Singapore (1998) 183–200
2. Fredkin, E., Toffoli, T.: Conservative logic. Int. J. Theoretical Physics **21** (3-4) (1982) 219–253

3. Hertling, P.: Embedding cellular automata into reversible ones. In: Calude, C.S., Casti, J., Dinneen, M.J. (eds.): Unconventional Models of Computation, Springer-Verlag, Singapore (1998) 243–256
4. Imai, K., Morita, K.: A computation-universal two-dimensional 8-state triangular reversible cellular automaton. Theoretical Computer Science **231** (2) (2000) 181–191
5. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Research and Development **5** (1961) 183–191
6. Margolus, N.: Physics-like models of computation. Physica D **10** (1984) 81–95
7. Morita, K.: A simple universal logic element and cellular automata for reversible computing. Lecture Notes in Computer Science **2055** (2001) 102–113
8. Morita, K., Tojima, Y., Imai, K.: A simple computer embedded in a reversible and number-conserving two-dimensional cellular space. Multi. Val. Logic **6** (2001) 483–514
9. Peper, F., Isokawa, T., Kouda, N., Matsui, N.: Self-timed cellular automata and their computational ability. Future Generation Computer Systems **18** (7) (2002) 893–904
10. Peper, F., Lee, J., Adachi, S., Mashiko, S.: Laying out circuits on asynchronous cellular arrays: towards feasible nanocomputers. (submitted)
11. Toffoli, T.: Computation and construction universality of reversible cellular automata. J. Computer and System Sciences **15** (1977) 213–231

General-Purpose Parallel Simulator for Quantum Computing

Jumpei Niwa[1], Keiji Matsumoto[2], and Hiroshi Imai[1,2]

[1] Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan.
[2] Quantum Computation and Information Project, ERATO, Japan Science and Technology Corporation, 5-28-3 Hongo, Bunkyo-ku, Tokyo Japan.

Abstract. With current technologies, it seems to be very difficult to implement quantum computers with many qubits. It is therefore of importance to simulate quantum algorithms and circuits on the existing computers. However, for a large-size problem, the simulation often requires more computational power than is available from sequential processing. Therefore, simulation methods for parallel processors are required.
We have developed a general-purpose simulator for quantum algorithms/circuits on the parallel computer (Sun Enterprise4500). It can simulate algorithms/circuits with up-to *30 qubits*. In order to test efficiency of our proposed methods, we have simulated Shor's factorization algorithm and Grover's database search, and we have analyzed robustness of the corresponding quantum circuits in the presence of both decoherence and operational errors. The corresponding results, statistics and analyses are presented in this paper.

Keywords: quantum computer simulator, Shor's factorization, Grover's database search, parallel processing, decoherence and operational errors

1 Introduction

Modern computers, not only PCs, but also super-computers, are based on the semiconductor switches. As their sizes have got smaller, their processing speed has got faster and their processing data has got larger. However, it has become apparent that if development of computing technology is to follow the Moore's law, at some stage, probably within the next 10 to 20 years, the number of atoms in the structures will become so small that underlying physical phenomena will not follow the laws of classical physics, but the laws of quantum physics. Therefore, it will be necessary to take quantum effects into account when designing new, more powerful computers.

A quantum computer [3],[4] could solve efficiently some important algorithmic problems using superposition and interference principles of quantum mechanics. When an atom or a particle is used as a quantum bit, then quantum mechanics says that it can be prepared as a coherent superposition of two basis states $|0\rangle$ and $|1\rangle$. Strings of qubits can be *entangled* and encode in some sense a

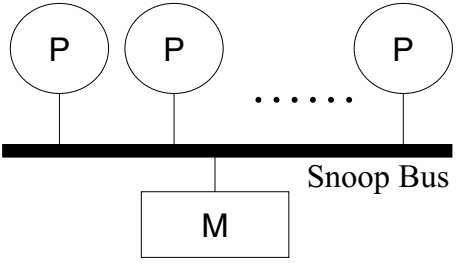
C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 230–251, 2002.

vast amount of information. Quantum computers can support entirely new kinds of computation, with qualitatively new algorithms based on quantum principles. Shor [14] proposed polynomial quantum algorithms for integer factoring and discrete logarithm computation. Grover [5] suggested a quantum search algorithm that achieves quadratic speedup with respect to classical ones.

Having such theoretical results, a natural question is whether we could ever build quantum computers. One of the obstacles is *decoherence*–an interaction of quantum systems with their environment that destroys fragile superposition in the quantum systems in which computations are performing–. Decoherence is caused because the environment is "measuring" the state of a quantum system by interacting with it. Second one is *inaccuracy* (operational errors). Quantum computers are actually analog: that is, they have a continuum of states. For example, one of the most common quantum gates "rotates a quantum bit by an angle θ". When applying this gate, there is always some inaccuracy in the rotation.

With current technologies, it seems to be very difficult to implement quantum computers with many qubits. It is therefore of importance to simulate quantum algorithms and circuits on the existing computers. The purpose of the simulation is

- to investigate quantum algorithms behavior.
  Of course, we have already known that Grover's search algorithm is optimal. That is, no quantum algorithm can search N items using fewer than $\Omega(\sqrt{N})$ accesses to the search oracle. However, we do not know whether Shor's factorization algorithm is optimal. Therefore, it is important to check how effective the improved Shor's factorization algorithm is by simulations.
- to analyze performance and robustness of quantum circuits in the presence of decoherence and operational errors.
  Not to mention, it is necessary to use quantum error-correcting codes to fight decoherence and operational errors. However, these errors may happen not only in the main circuits but also in the quantum error-correcting circuits themselves. Therefore, it is necessary to investigate effects of these errors.



P: Processor, M: Memory
Fig. 1. SMP (Symmetric Multi-Processors).

Simulations often require more computational power than is usually available on sequential computers. Therefore, we have developed a simulation method for parallel computers. That is, we have developed a general-purpose simulator for quantum algorithms and circuits on a parallel computer: Symmetric Multi-Processor (shown in Figure 1).

2 Basic Design

2.1 Registers

The simulation is for circuit model of quantum computation. A set of n qubits is called a *register* of size n. The general state of the n-qubit register is

$$|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle, \text{ where } \alpha_i \in \mathcal{C} \ , \ \sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \ .$$

That is, the state of an n-qubit register is represented by a unit-length complex vector in $\mathcal{H}_{2^n}$. In a classical computer, to store a complex number $\alpha = x + iy$, one needs to store a pair of real numbers (x, y). In our implementation each real number will be represented by a *double precision word*. The double precision word is 16 bytes (64bits) on most of the computers. 2^{n+4} bytes memory are therefore required to deal with the state of an n-qubit register in a classical computer.

2.2 Evolution

The time evolution of an n-qubit register is determined by a unitary operator of $\mathcal{H}_{2^n}$. The size of the matrix is $2^n \times 2^n$. In general, it requires $2^n \times 2^n$ space and $2^n(2^{n+1} - 1)$ arithmetic operations to perform classically such an evolution step.

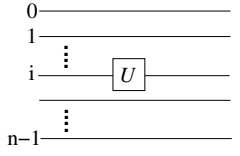


Fig. 2. Single qubit gate.

However, we mostly use operators that have simple structures when we design quantum circuits. That is, an evolution step is performed by applying a unitary operator (2×2) to a single qubit (a single qubit gate) or by applying the controlled unitary operator such as a C-NOT gate. It requires only 2×2 space and $3 \cdot 2^n$ arithmetic operations to simulate such an evolution step as explained below.

Single Qubit Gates. Suppose that the MSB (most significant qubit) is 0-th qubit. When a unitary matrix $\boldsymbol{U} = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$ is applied to the i-th qubit (Figure 2), the overall unitary operation applied to the n-qubit register state has the form $X = (\bigotimes_{k=0}^{i-1} I) \bigotimes U \bigotimes (\bigotimes_{k=i+1}^{n-1} I)$. $2^n \times 2^n$ matrix X is the sparse regular matrix shown in Figure 3. That is, all the S_i are the same. We therefore

do not have to generate X explicitly. We have to only store the 2×2 matrix U. Since there are only 2 non-zero elements for each row in X, the evolution step (i.e., multiply of a matrix and a vector) is simulated in $3 \cdot 2^n$ arithmetical operations.

$$X = \underbrace{\begin{pmatrix} S_0 & & & & 0 \\ & S_1 & & & \\ & & \cdots & & \\ & & & S_{2^i-2} & \\ 0 & & & & S_{2^i-1} \end{pmatrix}}_{2^n} \text{ where } S_k = \underbrace{\begin{pmatrix} u_{11} & \mathbf{0} & u_{12} & \mathbf{0} \\ \cdots & & \cdots & \\ \mathbf{0} & u_{11} & \mathbf{0} & u_{12} \\ u_{21} & \mathbf{0} & u_{22} & \mathbf{0} \\ \cdots & & \cdots & \\ \mathbf{0} & u_{21} & \mathbf{0} & u_{22} \end{pmatrix}}_{2^{n-i}}$$

$$(0 \leq k < 2^i)$$

Fig. 3. Total unitary matrix.

Parallelization

Of course, the evolution step $(X|\phi\rangle)$ can be executed in parallel. Let 2^P be the number of processors available in the simulating system. The evolution step is decomposed into a sequence of submatrix-subvector multiplication M_k $(0 \leq k < 2^i)$. M_k is defined as $S_k\phi_k$, that is, as the multiplication of a submatrix S_k $(2^{n-i} \times 2^{n-i})$ and a subvector ϕ_k whose length is 2^{n-i} (shown in Figure 4).

$$X|\phi\rangle = \begin{pmatrix} S_0 & & & & 0 \\ & S_1 & & & \\ & & \cdots & & \\ \hline & & \cdots & & \\ & & & \cdots & \\ \hline & & & \cdots & \\ 0 & & & & S_{2^i-2} \\ & & & & & S_{2^i-1} \end{pmatrix} \begin{pmatrix} \phi_0 \\ \phi_1 \\ \cdots \\ \hline \cdots \\ \cdots \\ \hline \cdots \\ \phi_{2^i-2} \\ \phi_{2^i-1} \end{pmatrix} \begin{matrix} \} \, (processor\ 0) \\ \cdots \\ \cdots \\ \cdots \\ \} \, (processor\ 2^P) \end{matrix}$$

$$\text{where } \phi_k = \begin{pmatrix} \alpha_{k2^{n-i}} \\ \alpha_{k2^{n-i}+1} \\ \cdots \\ \alpha_{(k+1)2^{n-i}-2} \\ \alpha_{(k+1)2^{n-i}-1} \end{pmatrix} (0 \leq k < 2^i)$$

Fig. 4. Computation decomposition in the general case.

Note that there are no data-dependencies between M_k and M_l $(k \neq l)$. Therefore, M_k and M_l can be executed in parallel. We assign a sequence of computa-

tion: $\underbrace{M_{p2^{i-P}}, M_{p2^{i-P}+1}, \ldots, M_{(p+1)2^{i-P}-1}}_{2^{i-P}}$ to a processor p $(0 \leq p < 2^P)$. That is, the processor p computes 2^{i-P} submatrix-subvector multiplications, and the rests of multiplications are performed in other processors in parallel. After each processor has finished computations that were assigned to it, it executes a synchronization primitive, such as the barrier, to make its modifications to the vector (ϕ), that is, the state of the register visible to other processors.

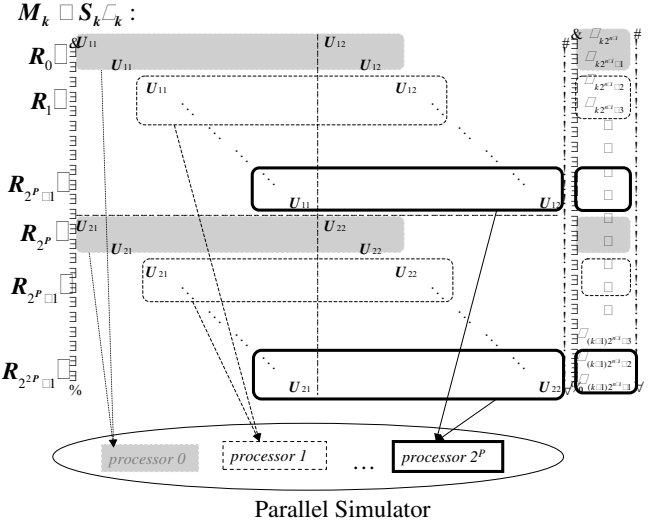


Fig. 5. Computation decomposition in the large subblock case.

When the number of submatrices is smaller than the number of processors (i.e., $2^i < 2^P$), it is inefficient to assign the computation $M_k (= S_k \phi_k,\ 0 \leq k < 2^i))$ to one processor as described above. It can cause a load imbalance in the simulation system. In this case, we should decompose the computation M_k itself to improve parallel efficiency. Each submatrix S_k is then divided into 2^{P+1} chunks of rows. Each chunk of rows, R_j $(0 \leq j < 2^{P+1})$, contains the adjoining $2^{n-i-(P+1)}$ rows of S_k. The multiplications using the chunk of rows R_j and R_{2^P+j} are assigned to a processor j as described in the Figure 5. This decomposition is applied to all M_k computations $(0 \leq k < 2^i)$.

Note that the computation using j-th row of the submatrix must be always paired with that using $(j + 2^{n-i-1})$-th row when we use an "in-place" algorithm (i.e., The results of $X|\phi\rangle$ are stored in $|\phi\rangle$).

That is, multiplications using the chunk of rows R_j and R_{2^P+j} are assigned to the same processor j. This is because there are dependencies across processors. Consider for example the following case.

$$\begin{bmatrix} xu_{11}+yu_{12} \\ \cdots \\ \cdots \\ xu_{21}+yu_{22} \\ \cdots \\ \cdots \end{bmatrix} = \begin{bmatrix} u_{11} & \cdots & \mathbf{0} & u_{12} & \cdots & \mathbf{0} \\ \cdots & & \cdots & \cdots & & \cdots \\ \mathbf{0} & \cdots & u_{11} & \mathbf{0} & \cdots & u_{12} \\ u_{21} & \cdots & \mathbf{0} & u_{22} & \cdots & \mathbf{0} \\ \cdots & & \cdots & \cdots & & \cdots \\ \mathbf{0} & \cdots & u_{21} & \mathbf{0} & \cdots & u_{12} \end{bmatrix} \begin{bmatrix} x \\ \cdots \\ \cdots \\ y \\ \cdots \\ \cdots \end{bmatrix}$$

If the first element is computed and the result $(xu_{11}+yu_{12})$ is stored before the fourth element is computed, then the result of fourth element computation is not $xu_{21}+yu_{22}$, but $(xu_{11}+yu_{12})u_{21}+yu_{22}$. However, this is wrong. To avoid this situation, all the processors have to execute barrier operations before storing the results of computations. However, a barrier operation per store operation can cause heavy overheads.

Therefore, the first element computation and fourth element computation should be assigned to the same processor. Then, the data-dependencies are not cross-processor but in-processor.

First, the processor computes $xu_{11}+yu_{12}$ and stores the result in a temporary variable t_1 on the local storage-area (i.e., stack). Second, the processor itself computes the result $xu_{21}+yu_{22}$ and stores it in the fourth element. Third, the processor stores the contents of the temporary variable t_1 in the first element. In this way, we can avoid the above wrong situation without performing synchronization primitives. If there are no overheads for parallel execution, the time complexity is thus reduced to $O(2^{n-P})$ where 2^P is the number of processors available in the system.

Controlled Qubit Gates

Suppose that a unitary matrix $\boldsymbol{U} = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix}$ is applied to the i-th qubit if and only if the c-th bit (controlled bit) is 1.

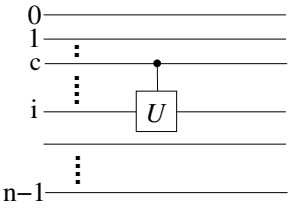


Fig. 6. Controlled qubit gate.

Let CTX be the overall unitary matrix ($2^n \times 2^n$). First, we consider the matrix X mentioned in Section2.2 as if there were no controlled bits. Then, for each j $(0 \leq j < 2^n - 1)$, the j-th row of CTX ($CTX[j]$) is defined as follows.

$$CTX[j] = \begin{cases} X[j] & \text{the } c\text{-th bit in } j \text{ is 1} \\ I[j] & \text{the } c\text{-th bit in } j \text{ is 0} \end{cases}$$

where $I[j]$ is j-th row of the unit matrix I and $X[j]$ is j-th row of the matrix X. In this case, we also do not have to generate CTX or X explicitly. We have only to store the 2×2 matrix U. It is easy to extend this method to deal with the case of many controlled bits. The evolution step is executed in parallel as described in Section 2.2. Therefore, the simulation time is $O(2^{n-P})$ if there are no overheads for parallel execution (2^P is the number of processors available in the simulation system.)

f-controlled U gate is also simulated when f is a boolean function. It is similar to the controlled U gate. But in this case, the U gate is applied to the target bit iff $f(c) = 1$ (c-th bit is the controlled bit). This is used in the simulation of Grover's Search Algorithm [5].

Measurement Gates

Let us consider a measurement of the j-th qubit of an n-qubit register $|\phi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$. The corresponding observable is $\mathcal{B}_j = \{E_j^0, E_j^1\}$ where $E_j^0(E_j^1)$ is the 2^{n-1} dimensional subspace of the $\mathcal{H}_{2^n}$ spanned by all basic vectors having 0(1) in the j-th element. The measurement of j-th qubit gives

$$\begin{cases} 0 \text{ with probability } \sum_{i(i_j=0)} |\alpha_i|^2 \ (= p_0) \\ 1 \text{ with probability } \sum_{i(i_j=1)} |\alpha_i|^2 \ (= p_1) \end{cases},$$

where i_j denotes the j-th bit in i.

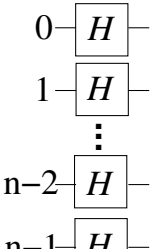
The post-measurement state becomes $|\phi'\rangle$ as follows.

$$|\phi'\rangle = \begin{cases} \frac{1}{\sqrt{p_0}} \sum_{i(i_j=0)} \alpha_i |i\rangle \\ \frac{1}{\sqrt{p_1}} \sum_{i(i_j=1)} \alpha_i |i\rangle \end{cases},$$

This measurement process is simulated in $O(2^n)$ step as follows.

1. $p_0 = 0$ and $p_1 = 0$
2. For each integer k $(0 \leq k < 2^n)$, if j-th bit of k is 0, $p_0 = p_0 + |\alpha_k|^2$. Otherwise (that is, j-th bit of k is 1), $p_1 = p_1 + |\alpha_k|^2$.
3. A random number r, $0 \leq r < 1$, is generated
4. When $r < p_0$, We consider that 0 is measured. Otherwise, 1 is measured.
5. For each integer k $(0 \leq k < 2^n)$, when 0 is measured, if j-th bit of k is 0, we set $\alpha_k = \frac{\alpha_k}{\sqrt{p_0}}$. If j-th bit of k is 1, we set $\alpha_k = 0$. When 1 is measured, if j-th bit of k is 0, we set $\alpha_k = 0$. If j-th bit of k is 1, we set $\alpha_k = \frac{\alpha_k}{\sqrt{p_1}}$

2.3 Basic Circuits



The Hadamard transform H_n is defined as follows,

$$H_n|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y \in 0,1^n} (-1)^{x \cdot y} |y\rangle, \text{for } x \in \{0,1\}^n.$$

H_n is implemented by the circuit that is displayed on the left, where $\boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Note the simulation requires $O(n2^{n-P})$ time, if there are no overheads for parallel execution (2^P is the number of processors available in the simulation system).

Quantum Fourier Transform. Quantum Fourier transform (QFT) is a unitary operation that essentially performs DFT on quantum register states. QFT maps a quantum state $|\phi\rangle = \sum_{x=0}^{2^n-1} \alpha_x |x\rangle$ to the state $\sum_{x=0}^{2^n-1} \beta_x |x\rangle$, where

$$\beta_x = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} \omega^{xy} \alpha_y, \quad \omega = e^{2\pi i/2^n}.$$

The circuit implementing the QFT is shown in the Figure 7. $\boldsymbol{H}$ is the Hadamard gate, and $\boldsymbol{R_d}$ is the phase shift gate denoted as $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^d} \end{pmatrix}$.

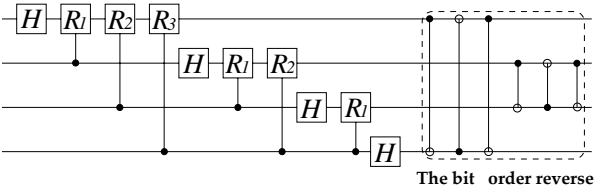


Fig. 7. The QFT_{2^n} circuit ($n = 4$).

For general n, this circuit is of size $O(n^2)$[1]. Therefore, the evolution step is simulated in $O(n^2 2^{n-P})$ time if there are no overheads for parallel execution (There are 2^P processors available in the system.). Of course, we can reduce the circuit size to $O(n \log(n/\epsilon))$ [1],[2], if we settle to the implementation with a fixed accuracy (ϵ), because the controlled phase shift gates acting on distantly separated qubits contribute only exponentially small phases. In this case, the evolution step is simulated in $O(n \log(n/\epsilon) 2^{n-P})$ steps if there are no overheads for parallel execution.

If we regard QFT transform as a *black box operator*, we do not have to use this quantum circuit in the simulator to perform QFT transformation. We can use fast Fourier transform (FFT) in the simulator instead of the QFT circuit if we suppose that QFT circuit has no error. FFT algorithm requires only $O(n2^{n-P})$ steps if there are no overheads for parallel execution. Of course, FFT gives the exact solution. We use the 8-radix in-place FFT algorithm.

Arithmetical circuits. Arithmetical circuits are important for quantum computing [16]. In the Shor's factoring algorithm [14], arithmetical circuits are needed to compute modular exponentiation. Therefore, according to Ref [8], we have implemented the modular exponentiation circuit by using constant adders, constant modular adders and constant multipliers. $x^a (\mod N)$ can be computed using the decomposition,

[1] There is a new quantum circuit that computes QFT (modulo 2^n) that has the size $O(n(\log n)^2 \log\log n)$ [2].

$$x^a (\text{mod } N) = \prod_{i=0}^{l-1} \left((x^{2^i})^{a_i} (\text{mod } N) \right), \text{ where}$$

$$a = \sum_{i=0}^{l-1} a_i 2^i (= a_{l-1} a_{l-2} \ldots a_0 \text{ (binary representation)})$$

Thus, modular exponentiation is just a chain of multiplications where each factor is either 1 ($a_i = 0$) or x^{2^i} ($a_i = 1$). Therefore, the circuit is constructed by the pairwise controlled constant multipliers[2].

Let N be an n bit number, and a a $2n$ bit number (that is, l is equal to $2n$ in the above equation.) in the Shor's factoring algorithm because a is as large as N^2. $n+1$ qubits are required as the work-space for the controlled multiplier and $n+4$ for the controlled adders. The total number of required qubits becomes $5n+6$.

The circuit is constructed with the $O(l)$ (that is, $O(n)$) pairwise controlled constant multipliers. A controlled constant multiplier consists of $O(n)$ controlled constant modular adders. A controlled constant modular adder consists of 5 controlled constant adders. A controlled constant adder consists of $O(n)$ XOR (C-NOT) gates. Thus, one modular exponentiation circuit requires $O(n^3)$ gate. Details are described in Ref [8]. It is simulated in $O(n^3 2^{n-P})$ steps if there are no overheads for parallel execution (2^P is the number of processors available in the simulation system.).

3 Error Model

3.1 Decoherence

We consider a quantum depolarizing channel as the decoherence error model. In this channel, each qubit is left unchanged with probability $1-p$. Otherwise, each of the following possibilities has probability $p/3$: the qubit is negated, or its phase if changed, or it is both negated and its phase is changed.

3.2 Operational Errors

In general, all single qubit gates can be generated from *rotations*

$$\boldsymbol{U_R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix},$$

and *phase shifts,*

$$\boldsymbol{U_{P1}}(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \text{ and } \boldsymbol{U_{P2}}(\phi) = \begin{pmatrix} e^{i\phi} & 0 \\ 0 & 1 \end{pmatrix}.$$

For example, $H = \boldsymbol{U_R}(\frac{\pi}{4})\boldsymbol{U_{P1}}(\pi)$, and NOT $= \boldsymbol{U_R}(\frac{\pi}{2})\boldsymbol{U_{P1}}(\pi)$. The simulator represents inaccuracies by adding small deviations to the angles of rotation θ and ϕ. Each error angle is drawn from the Gaussian distribution with the standard deviation (σ).

[2] Of course, we must classically compute the numbers $x^{2^i} (\text{mod} N)$

4 Preliminary Experiments

We describe the simulation environment and some experiments about basic quantum circuits.

4.1 Simulation Environment

We have designed a simulator for the parallel computer: Sun Enterprise 4500 (E4500). E4500 has 8 UltraSPARC-II processors (400MHz) with 1MB E-cache and 10GB memory. The system clock is 100MHz. OS is Solaris 2.8 (64bit OS). The simulator is written in the C language and the compiler we use is Forte Compiler 6.0. The compiler option "`-xO5 -fast -xtarget=ultra2 -xarch=v9`" is used. We use the solaris thread library for multi-processor execution. Under this environment, if we use an in-place algorithm, *30-qubit quantum register states can be simulated.*

4.2 Quantum Fourier Transform

Table 1 shows QFT execution time by the simulator that uses QFT-circuit and (classical) FFT algorithm. Numerical error value ranges from 10^{-15} to 10^{-14}. Recall that 2^P is the number of processors available in the simulation system. FFT algorithm requires $O(n2^{n-P})$ steps, and QFT circuit requires $O(n^2 2^{n-P})$ steps for n-qubit quantum register, if there are no overheads for parallel execution. The execution time is increased exponentially with respect to n. Table 1 shows that the execution time of FFT is about $20 \sim 30$ times as fast as that of QFT-circuit. Both execution times are decreased when the number of processors is increased. The speedup-ratios on 8-processor execution are about 4 to 5. The reason why the speedup-ratios on 8-processor execution are not 8 is that the parallel execution has some overheads that single processor execution does not have. The parallel execution overheads are operating system overheads (multi-threads creation, synchronization, and so on), load imbalance, memory-bus saturation, memory-bank conflict, false sharing and so on. For small-size problems, the ratio of overheads to the computation for parallel execution is relatively large and speedup-ratios on multi-processor execution may be less than 4. The decoherence and operational errors experiment for QFT is described in Section 5.

4.3 Hadamard Transform

Table 2 shows Hadamard Transform (HT) execution time. HT circuit requires $O(n2^{n-P})$ steps for n-qubit quantum register. The speedup-ratio on 8-processor execution becomes about 5.

Effect of Errors. We have investigated the decrease of the $|0\rangle\langle 0|$ term in the density matrix for the 20-qubit register.

Table 1. QFT execution time (sec).

Qubits (n)	Algorithm	Num. of Procs			
		1	2	4	8
20	Circuit	26.08	7.25	5.01	5.33
	FFT	1.21	0.92	0.72	0.53
22	Circuit	124.78	66.96	38.03	23.40
	FFT	5.01	3.71	2.79	1.83
24	Circuit	643.02	331.98	183.01	137.7
	FFT	20.00	12.61	8.40	5.84
26	Circuit	2745.56	1469.73	799.57	526.82
	FFT	113.29	73.08	48.39	32.84
28	Circuit	12597.8	6738.13	3661.51	2338.19
	FFT	567.19	319.16	205.98	142.01
29	Circuit	31089.6	16790.6	9189.68	5811.49
	FFT	1232.16	697.68	423.00	286.29

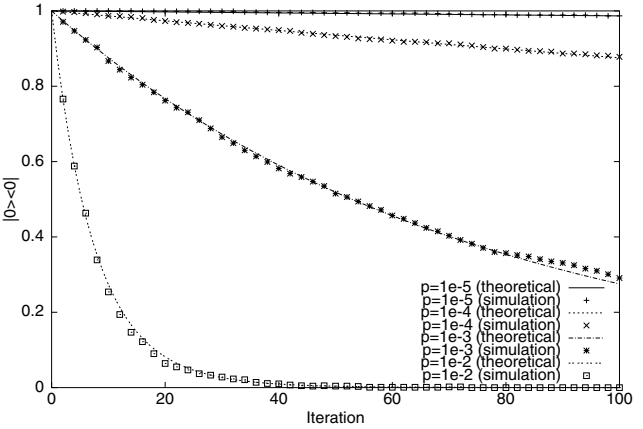


Fig. 8. Decrease of the $|0\rangle\langle 0|$ term in the density matrix (20 qubits).

Table 2. HT execution time (sec).

Qubits (n)	Num. of Procs			
	1	2	4	8
20	2.38	1.18	0.76	0.40
22	10.85	5.73	3.20	1.35
24	46.94	24.96	13.40	9.58
26	205.81	109.97	58.83	38.71
28	887.40	467.71	253.82	167.31
29	2027.9	1081.1	592.08	395.81

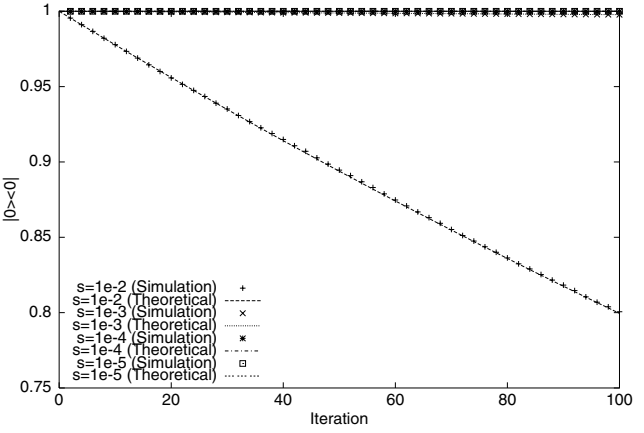


Fig. 9. Decrease of the $|0\rangle\langle 0|$ term in the density matrix (20 qubits).

Decoherence Errors

We have analyzed decoherence in the HT circuit on the depolarizing channel. Of course, the simulation deals with pure states. Therefore, the experiments were repeated 10000 times and we use the average values. Each experiment uses different initial random seed. The initial state of the quantum register is $|00\ldots0\rangle = |0\rangle$. HT circuit is applied to the quantum register over and over. The x-axis in Figure 8 shows the even iteration number. If there are no errors (i.e., the error probability is 0) and the number of iterations is even, the state remains to be $|0\rangle$ and $|0\rangle\langle 0|$-term in the density matrix remains 1. Figure 8 shows how decoherence errors degrade for the $|0\rangle\langle 0|$-term. The noise degrades the $|0\rangle\langle 0|$-term significantly if the error probability is greater than 10^{-3}. When the error probability is 10^{-2}, the $|0\rangle\langle 0|$-term is decreased in exponential order in proportional to the number of iterations.

In this easy case, we can compute $|0\rangle\langle 0|$-term in the density matrix also theoretically. First, consider the 1 qubit case. Let p be the error probability and ρ_k be the density matrix after the HT circuit is applied to the quantum register k times. The density matrix ρ_{k+1} is then calculated as follows.

$$\rho_{k+1} = (1-p)H\rho_k H^* + \frac{p}{3}\sigma_x H\rho_k H^* {\sigma_x}^* + \frac{p}{3}\sigma_y H\rho_k H^* {\sigma_y}^* + \frac{p}{3}\sigma_z H\rho_k H^* {\sigma_z}^*.$$

When the initial state of the quantum register is $|0\rangle$ and k is even, ρ_k is calculated as follows

$$\rho_k = \frac{1}{2}\begin{pmatrix} 1+(1-\frac{4}{3}p)^k & 0 \\ 0 & 1-(1-\frac{4}{3}p)^k \end{pmatrix}.$$

In the $n-$qubit case, we can calculate the density matrix similarly if the initial state of the quantum register is $|0,\ldots,0\rangle$ and k is even. $|0\rangle\langle 0|$-term of ρ_k is

$$(\frac{1+(1-\frac{4}{3}p)^k}{2})^n.$$

Figure 8 also shows this theoretical value of $|0\rangle\langle 0|$-term in the density matrix if $p = 10^{-5} \sim 10^{-2}$ and $n = 20$. We can see that the simulations and the theoretically computations yield almost the same result.

Operational Errors

The simulator implements "inaccuracies" by adding small deviations to two angles of rotations. Since $H = \boldsymbol{U_R}(\frac{\pi}{4})\boldsymbol{U_{P1}}(\pi)$, we add small deviations x and y to $\frac{\pi}{4}$ and π respectively. That is, we use $H(x,y) = \boldsymbol{U_R}(\frac{\pi}{4}+x)\boldsymbol{U_{P1}}(\pi+y)$ as H gate in this experiment. x and y are drawn from the Gaussian distribution with the standard deviation (σ). As mentioned above, the experiments were executed 10000 times and we use the average value. Each experiment uses different initial random seed. Figure 9 shows how operational errors degrade the $|0\rangle\langle 0|$-term when $\sigma = 10^{-5} \sim 10^{-2}$ and $n = 20$. The $|0\rangle\langle 0|$-term is not affected by the operational error if σ is less than 10^{-2}.

In this case, we can also compute theoretically the $|0\rangle\langle 0|$-term in the density matrix. First, consider the 1 qubit case. Let ρ_k be the density matrix after HT circuit is applied to the quantum register k times. The density matrix ρ_{k+1} can be calculated as follows.

$$\rho_{k+1} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} H(x,y)\rho_k H(x,y)^* p(x)p(y)dxdy,$$

where $p(z) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{z^2}{2\sigma^2}}$. If the initial state of the quantum register is $|0\ldots 0\rangle = |0\rangle$, ρ_k can be expressed as follows

$$\rho_k = \frac{1}{2}\begin{pmatrix} 1+e^{-\frac{\sigma^2}{4}9k} & 0 \\ 0 & 1-e^{-\frac{\sigma^2}{4}9k} \end{pmatrix}.$$

As for the general $n-$qubit case, we can calculate the density matrix similarly if the initial state of the quantum register is $|0\ldots,0\rangle$, and k is even. $|0\rangle\langle 0|$-term of ρ_k is then

$$(\frac{1+e^{-\frac{\sigma^2}{4}9k}}{2})^n.$$

Figure 9 also shows this theoretical value of $|0\rangle\langle 0|$-term in the density matrix if the standard deviation $\sigma = 10^{-5} \sim 10^{-2}$ and $n = 20$. It follows from the theoretical computation that $|0\rangle\langle 0|$-term decreases exponentially with respect to the number of iterations k.

Operational and Decoherence Errors

Each element of Table 3 represents the $|0\rangle\langle 0|$-term of the density matrix after HT is applied to the state $|0\rangle$ of a 20-qubit register 10000 times. The combined effect of two factors may be worse than in case of each factor alone. That is to say, the effect seems to be the product of each factor. Table 3 shows this situation.

Table 3. Combined effects for HT.

Decoherence(p) \ Operational(σ)	0	10^{-5}	10^{-4}	10^{-3}
0	1.0000	1.0000	0.9999	0.9977
10^{-5}	0.9870	0.9870	0.9849	0.9797
10^{-4}	0.9010	0.9010	0.8909	0.8780
10^{-3}	0.2910	0.2790	0.2779	0.2664

5 Experiments

5.1 Shor's Factorization Algorithm

We investigate behavior of Shor's factorization algorithm. The point is

- how effective the improved algorithm [15] is
- effects of decoherence errors and operational errors

First, we review the algorithm briefly.

Input. An l bit odd number n that has at least two distinct prime factors.
Output. A nontrivial factor of n

1. Choose an arbitrary $x \in \{1, 2, \ldots, n-1\}$
2. (Classical Step) Compute $d = \gcd(x, n)$ using Euclid's algorithm. If $d > 1$, output d and stop.
3. (Quantum Step) Try to find the order of x:
   a) Initialize an l-qubit register and a $2l$-qubit register to state $|0\rangle|0\rangle$.
   b) Apply HT to the second register.
   c) Perform modular exponentiation operation, that is, $|0\rangle|a\rangle \rightarrow |x^a (\mathrm{mod}\ n)\rangle|a\rangle$
   d) Measure the first register and apply the QFT to the second register and measure it. Let y be the result.
4. (Classical step) Find relatively prime integers k and r $(0 < k < r < n)$, s.t. $|\frac{y}{2^{2l}} - \frac{k}{r}| \leq \frac{1}{2^{(2l+1)}}$ by using the continued fraction algorithm. If $x^r \not\equiv 1 (\mathrm{mod}\ n)$, or if r is odd, or if $x^{r/2} \equiv \pm 1 (\mathrm{mod}\ n)$, output "failure" and stop.
5. (Classical step) Compute $d_\pm = \gcd(n, x^{\frac{r}{2}} \pm 1)$ using Euclid's algorithm. Output numbers $d_\pm$ and stop.

When the simulator performs all the step-3 operations (not only QFT but also modular exponentiation) on the quantum circuit, $5l + 6$ qubits are totally required, as described in Section 1. Therefore, the simulator can only deal with 4-bit integer n $(5l + 6 <= 30 \rightarrow l \leq 4)$. The 4-bit integer that satisfies the input property is only 15. We have tried to factor 15 on the simulator. Beyond our expectation, modular exponentiation is computationally much heavier than QFT.

Modular exponentiation requires $O(l^3 2^{l-P})$ steps and QFT on the circuit requires $O(l^2 2^{l-P})$ steps, when there are 2^P processors available in the simulation system and there are no overheads for parallel execution. Of course,

Table 4. Execution time in Shor's factorization algorithm, when $n = 15$ and $x = 11$ (All quantum operations are executed on the circuit).

Modular exponentiation	QFT
18184 (sec)	0.64270 (sec)

in the classical computer, modular exponentiation consists of basic operations such as addition, multiplication and division. However, these basic operations are not so heavy if the classical computer is used, because it has the dedicated non-reversible circuit (the so-called ALU –arithmetic logic unit–). This situation suggests that a brand-new fast quantum algorithm for arithmetic operations is required to factor larger numbers. 15 is not enough to investigate behavior of Shor's factoring algorithm. In order to factor much larger number in a reasonable time, the simulator performs the step 3(c) and the step 3(d) classically. That is, the modular exponentiation are computed classically and QFT is computed by FFT algorithm in the simulator. In this case, the simulator does not need to generate the first register. Therefore, the simulator can factor about $14 \sim 15$-bit integers (for example, 23089).

The factoring algorithm succeeds with the probability greater than

$$\begin{aligned}\mathrm{Prob}_{\mathrm{succ}}(n) &= p_{step2} + (1 - p_{step2})p_{step3\sim4} \\ &= (1 - \frac{\phi(n)}{n-1}) + \frac{\phi(n)}{n-1} \cdot (\frac{1}{2} \cdot \frac{4}{\pi^2} \frac{e^{-\gamma}}{\log\log n})\end{aligned}$$

where p_{step2} denotes the probability that the step 2 succeeds and $p_{step3\sim4}$ denotes the probability that step 3 and the step 4 succeed and γ is the Euler constant, and $\phi(n)$ is the Euler number of n. If the above algorithm is repeated $O(1/\mathrm{Prob}_{\mathrm{succ}}(n))$ times, the success probability can be as close to 1 as desired.

We choose an $n = pq$ where p and q are prime numbers. This kinds of integers are chosen in an RSA cryptosystem because it is believed that it is hard to factor such integers easily. Recall that $\phi(n) = (p-1)(q-1)$ for such integers. We have experimented with several RSA-type $14 \sim 15$-bit integers.

The simulator repeats the above algorithm until a nontrivial factor of n is found, and records the number of iterations. The experiment was executed 100 times, and we calculate the average of these recorded iterations. We have compared the simulation values with the theoretical number of needed iterations (i.e.,$1/\mathrm{Prob}_{\mathrm{succ}}(n)$). The results are shown in the Table 5. Theoretical values (**Theoretical**) are about only $2 \sim 4$ times as large as simulation values (**Original**). Although much more simulations are required, the theoretical values seem to be fairly good.

As suggested in Ref [15], the algorithm was optimized to perform less quantum computation and more (classical) post-processing.

1. *Neighbor y Check*
   No relatively prime integers k and r are found by using the continued fraction algorithm, then it is wise to try $y \pm 1$, $y \pm 2$.
2. *GCD Check*
   Even if $x^r \not\equiv 1 \pmod{n}$, try to compute $d_{\pm} = \gcd(n, x^{\frac{r}{2}} \pm 1)$.

Table 5. Number of needed iterations for Shor's factoring algorithm.

n	Num. of Iterations		
	Theoretical	Simulation	
		Original	Improved
$21311(= 211 \cdot 101)$	15.79	6.690	1.760
$21733(= 211 \cdot 103)$	15.85	8.990	2.356
$22999(= 211 \cdot 109)$	16.00	6.360	1.730
$22523(= 223 \cdot 101)$	15.88	5.480	1.770
$22927(= 227 \cdot 101)$	15.91	3.790	1.470
$22969(= 223 \cdot 103)$	15.94	8.050	2.070
$23129(= 229 \cdot 101)$	15.92	7.133	1.636

3. *Small Factor Check*
   If $x^r \not\equiv 1 (\bmod\ n)$, it is wise to try $2r, 3r \ldots$. This is because if $\frac{y}{2^{2l}} \approx \frac{k}{r}$, where k and r have a common factor, this factor is likely to be small. Therefore, the observed value of $\frac{y}{2^{2l}}$ is rounded off to $\frac{k'}{r'}$ in the lowest terms.
4. *LCM Check*
   If two candidates for r, r_1 and r_2, have been found, it is wise to test $\mathrm{lcm}(r_1, r_2)$ as a candidate for r.

We have tested how much the algorithm is improved by these modifications. The results are also shown in Table 5 (**Improved**). The number of iterations is reduced to about $1/5 \sim 2/5$. The detailed effect of the improved algorithm is described in Table 6. Each element of Table 6 represents ratio s/f where s

Table 6. Detailed effect of improved algorithm

n	Ratio of Success/Failure			
	1(Neighbor)	2(GCD)	3(SF)	4(LCM)
21311	27/9	52/19	12/4	3/4
23129	27/9	52/19	12/4	3/4
22999	37/6	47/79	13/8	2/58
22969	41/8	22/82	31/22	1/28
22927	25/3	35/49	18/2	1/28
22523	37/6	45/76	18/22	7/54

means the number of success iterations and f the number of failure iterations. For example, for $n = 23129$, the first optimization, "Neighbor Check" is performed for $27 + 9 = 36$ iterations and a candidate of the order is found successfully in 27 iterations. It seems that the second optimization "GCD Check" works well for all n that we have experimented with. From this result, we can see that even if

$x^r \not\equiv 1 (\mathrm{mod}\ n)$, $d_{\pm} = \gcd(n, x^{\frac{r}{2}} \pm 1)$ often becomes a factor of n. That is, even if the candidate r is not equal to $ord(x)$ (an order of x), r may be a divisor of $ord(x)$. That is, $\mathbf{N} \ni \exists a > 1,\ a \cdot r = ord(x)$. In this case, the following equation holds when r is even.

$$\begin{aligned} 0 (\mathrm{mod}\ n) &\equiv x^{ord(x)} - 1 \\ &\equiv (x^r - 1)(x^{(a-1)r} + x^{(a-2)r} + \ldots + 1) \\ &\equiv (x^{r/2} - 1)(x^{r/2} + 1)(x^{(a-1)r} + x^{(a-2)r} + \ldots 1) \end{aligned}$$

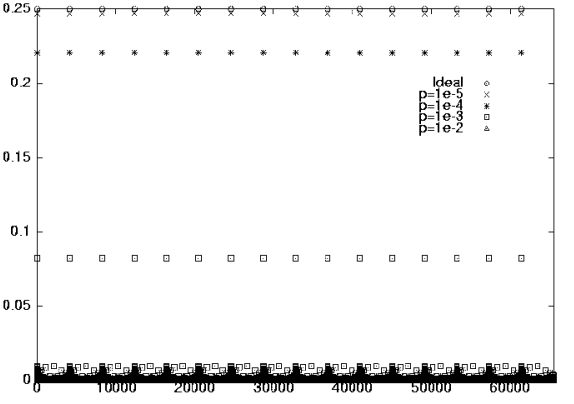
Thus, there is the possibility that n and $x^{\frac{r}{2}} \pm 1$ have a common non-trivial factor.

5.2 Effect of Errors

We have analyzed decoherence and operational errors in the QFT circuit.

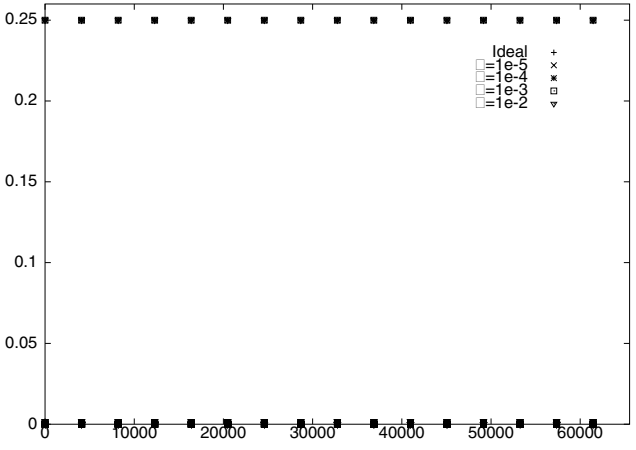
Decoherence Errors

We assume that each qubit is left intact with probability $1 - p$ and it is affected by each of the error operators $\sigma_x, \sigma_y, \sigma_z$ with the same probability $\frac{p}{3}$, each time the register is applied by the controlled rotation gate $\boldsymbol{R_d}$. Figure 10 shows the amplitude amplification phase by the QFT circuit on the depolarizing channel in Shor's factorization algorithm (Step 3 (d)), when $n = 187$ and $x = 23$. The y-axe in the Figure 10 shows the amplitude. The experiment was executed 1000 times and we use the average. If the error probability is greater than 10^{-3}, it is hard to use QFT circuit for period estimation.



p	0	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Iterations	1.9569	2.1000	2.3201	6.0606	327.00

Fig. 10. Amplitude amplification by QFT in the presence of decoherence error (top) and the required number of iterations (bottom) (16 qubits).



σ	0	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Iterations	1.9569	1.9841	2.0283	1.9015	1.9607

Fig. 11. Amplitude amplification by QFT in the presence of operational error (top) and the required number of iterations (bottom) (16 qubits).

Operational Errors

In the simulator, "inaccuracies" are implemented by adding small deviations to angles of rotations of $\boldsymbol{R_d}$. We consider $H_n = \boldsymbol{U_R}(\frac{\pi}{4})\boldsymbol{U_{P1}}(\pi)$, and NOT $= \boldsymbol{U_R}(\frac{\pi}{2})\boldsymbol{U_{P1}}(\pi)$. The simulator also represents inaccuracies by adding small deviations to these angles of rotations. The error is drawn from the Gaussian distribution with the standard deviation (σ). As mentioned above, the experiment was executed 1000 times and we use the average value. Figure 11 shows the amplitude amplification phase by QFT in Shor's factorization algorithm (Step 3(d)), when $n = 187$ and $x = 23$. It seems that the period extraction by using QFT is not affected by the operational error.

Operational and Decoherence Errors

We investigate also the combined effect of operational and decoherence errors. Table 7 shows the results. Each element of the table represents the *fidelity*. The fidelity is defined as the inner product of the correct state and the simulated state with errors.

The combined effect of two factors may be worse than each factor alone. That is to say, the effect seems to be the product of each factor. However, when the decoherence rate is relatively higher, the small-deviation operational error can improve the results contrary to our expectations.

5.3 Grover's Search Algorithm

Suppose that a function $f_k : \{0,1\}^n \to \{0,1\}$ is an oracle function such that $f_k(x) = \delta_{xk}$. The G-iteration is defined as $-H_n V_{f_0} H_n V_{f_k}$. The sign-changing

Table 7. Combined effects for QFT (16bit)

Decoherence(p) \ Operational(σ)	0	10^{-5}	10^{-4}	10^{-3}	10^{-2}
0	1.0000	0.9999	0.9999	0.9999	**0.9998**
10^{-5}	0.9880	**0.9840**	0.9860	0.9880	0.9848
10^{-4}	0.8837	0.8897	0.8827	**0.8801**	0.8980
10^{-3}	0.3287	0.3399	0.3332	**0.3209**	0.3363
10^{-2}	0.0027	**0.0015**	0.0019	0.0017	0.0031

operator V_f is implemented by using the f-controlled NOT gate and one ancillary bit. Figure 12 shows the circuit of Grover's algorithm.

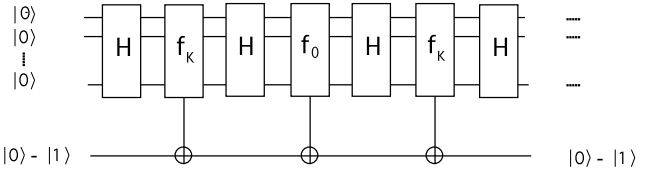


Fig. 12. The circuit of Grover's algorithms.

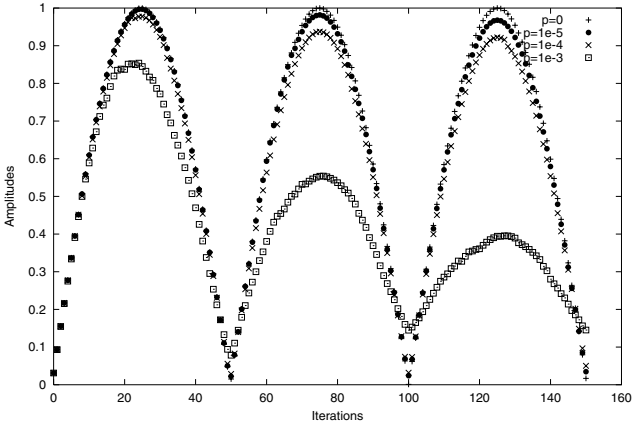


Fig. 13. Decrease of the amplitude of the correct element in the presence of decoherence errors (10 qubit).

Effect of Errors. We have analyzed the impacts of decoherence and operational errors in the circuit for Grover's algorithm. We assume again that the depolarizing channel is used. "Inaccuracies" are implemented by adding small deviations to the angles of these rotations. Each error angle is drawn again from the Gaussian distribution with the standard deviation (σ).

Figure 13 and 14 show the impacts of errors for a 10-qubit register. The experiments were repeated 1000 times and we use the average values. If there are no errors, by plotting the amplitude of the correct element (that is, k) we get a sine curve. However, the amplitudes are decreased as G-iterations are repeated in the presence of errors. Figure 13 shows the impacts of decoherence error. We can see that the decoherence error affects the period of the sine-curve. Figure 14 shows the impacts of operational errors. It seems that the operational error does not affect the period of the sine-curve.

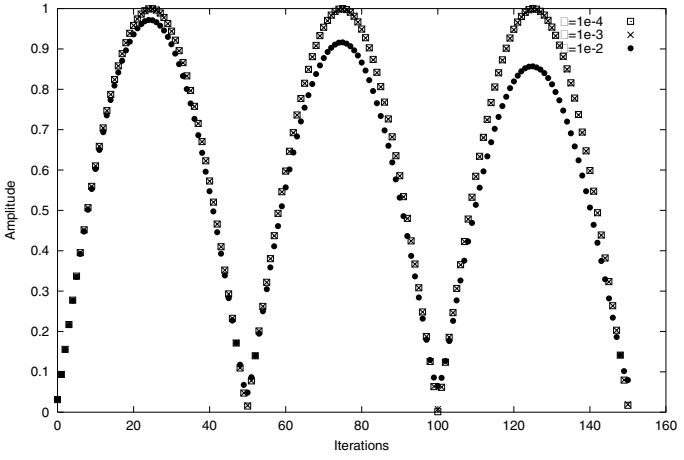


Fig. 14. Decrease of the amplitude of the correct element in the presence of operational errors (10 qubit).

6 Related Works

There are many quantum simulators for quantum circuit model of computation [9],[13],[11],[12]. QDD[13] aims to use Binary Decision Diagram in order to represent the states of quantum register. QCL[11] and OpenQubit[12] both use complex number representation of quantum states like our simulator. In addition, QCL tries to establish a high-level, architecture-independent programming language. Obenland's simulator [9], [10] is based on an actual physical experimental realization and it uses parallel processing like our simulator. Although it runs on the distributed-memory multi-computers, our simulator runs on the shared-memory multi-computers. Therefore, in our simulator, there is no need to distribute and collect states of the quantum register. In addition, our simulator uses more efficient evolution algorithms and adopts (classical) FFT algorithms for fast simulation of the large-size problems. Our simulator does not depend on any actual physical experimental realizations. It is not easy to say which realizations are best at the moment. In other words, our simulator is more general-purpose.

Gui et al. investigated the effects of gate imperfections (operational errors) in Grover's search and Shor's factorization by performing numerical simulations [7], [6]. But they do not consider decoherence errors. Our simulator deals not only with operational errors but also with decoherence errors.

7 Conclusion

We have developed a parallel simulator for quantum computing on a parallel computer (Sun, Enterprise4500). Up-to *30 qubits* can be handled. We have performed Shor's factorization and Grover's database search by using the simulator. Our results show that the improved Shor's factorization algorithm is really effective. We analyzed robustness of the corresponding quantum circuits in the presence of decoherence and operational errors. If the decoherence rate is greater than 10^{-3}, it seems to be hard to use both quantum algorithms in practice.

For future work, we will investigate the correlation between decoherence and operational errors. That is, why small-deviation operational errors can improve the results when the decoherence rate is relatively higher. Furthermore, we will try to use quantum error-correcting code to fight decoherence and operational errors.

References

1. A. Barenco, A. Ekert, K. Suominen, and P. Torma. Approximate quantum fourier transform and decoherence, 1996.
2. R. Cleve and J. Watrous. Fast parallel circuits for the quantum fourier transform. In *IEEE Symposium on Foundations of Computer Science*, pages 526–536, 2000.
3. David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London A*, 400:97–117, 1985.
4. David Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London A*, 425:73–90, 1989.
5. Lov K. Grover. A fast quantum mechanical algorithm for database search. In *ACM Symposium on Theory of Computing*, pages 212–219, 1996.
6. Hao Guo, Gui Lu Long, and Yang Sun Li. Effects of imperfect gate operations in shor's prime factorization algorithm. *Chinese Chemical Society*, 48(3):449–454, 2001.
7. Gui Lu Long, Yan Song Li, Wei Lin Zhang, and Chang Cun Tu. Dominant gate imperfection in grover's quantum search algorithm. *Physical Review A*, 61(4):042305, 2000.
8. Cesar Miquel, Juan Pablo Paz, and Roberto Perazzo. Factoring in a dissipative quantum computer. Los Alamos Physics Preprint Archive, http://xxx.lanl.gov/abs/quant-ph/9601021, 1996.
9. K. Obenland and A. Despain. A parallel quantum computer simulator. In *High Performance Computing*, 1998.
10. Kevin Mark Obenland. *Using Simulation To Access The Feasibility Of Quantum Computing*. PhD thesis, University of Southern California, 1998.
11. Bernhard Ömer. Quantum programming in qcl. Master's thesis, Institute of Information Systems Technical University of Vienn na, January 2000.

12. Yan Protzker, Jonathan Blow, and Joe Nelson. OpenQubit 0.2.0 `http://www.ennui.net/~quantum/`, December 1998.
13. QDD ver.0.2, `http://home.plutonium.net/~dagreve/qdd.html`, March 1999.
14. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factori ng. In *IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.
15. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete log arithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
16. Vlatko Vedral, Adriano Barenco, and Artur K. Ekert. Quantum networks for elementary arithmetic operations. Physical Review A, 54(1):147-153, 1996.

Towards Additivity of Entanglement of Formation

Toshiyuki Shimono

Department of Computer Science, University of Tokyo, Japan
shimono@is.s.u-tokyo.ac.jp

Abstract. *Quantum entanglement* is the most precious and most unusual quantum computation and communication resource. It is therefore of importance to understand how much of entanglement particular quantum states have, and what are the laws and limitations of transformation of entanglement from one form to another. To deal with this problem, it is necessary to use proper measures of entanglement that have nice properties. One of the most natural of such measures is that of entanglement of formation, and one of the major and long standing open problems concerning that measure is whether it is additive. In this paper we attack this hard problem. We show that for an important special case of states entangled formation is indeed additive. We also discuss this additivity problem and its merit in a broader context.

1 Introduction

Quantum entanglement is the most inherently quantum feature of quantum information, computation and communication. Due to quantum entanglement, quantum physics exhibits puzzling non-locality phenomena that lead to mysterious and weird consequences.

At the same time, quantum entanglement is increasingly considered as a very new and important resource for quantum computation and communication.

Entanglement as a resource allows:

- to perform transformations that are not possible otherwise (entanglement can act as a catalyst);
- to speed-up some computations;
- to economize communications;
- to increase capacity of quantum communication channels;
- to implement perfectly secure information transmissions;

Since entanglement is a resource, it is of large importance to determine the amount of entanglement in particular quantum states, to understand how well we can transform entanglement from one form to another, what are the laws and limitations of entanglement sharing among various parties, which transformations do not change the amount of entanglement and so on. In order to deal

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 252–263, 2002.

well with all these, and many other problems, we need to have a good measure, or measures, of entanglement.

The last task turned out to be much more difficult than it was expected. It is nowadays quite clear that there is no single measure of entanglement and that we need to explore a variety of measures of entanglement.

As usually, a natural way of searching for good measures of entanglement is to determine first which useful and nice properties such measures of entanglement should have, and, in parallel, to explore those measures that seems to be natural, either from a physical point of view, or from a mathematical point of view, or from an informatics point of view, and to determine which properties they have and what are the relations between them.

In the rest of this introduction we first introduce basic concepts concerning Hilbert spaces, density matrices and entanglement. Afterwards, we discuss and analyze basic measures of entanglement of pure and mixed bipartite states, and the importance of the additivity problem.

1.1 Basic Concepts

A finite dimensional Hilbert space, say n-dimensional, notation $\mathcal{H}_n$, is a vector space of n-dimensional complex vectors, on which the **inner product**, $\langle x|y\rangle$, of two (column) vectors $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ and $\boldsymbol{y} = (y_1, \ldots, y_n)^T$, is defined by $\langle x|y\rangle = \sum_{I=1}^{n} x_I^* y_I$.

The norm of a vector x is then defined as $\sqrt{\langle x|x\rangle}$. Vectors of norm 1 correspond to the intuitive concept of quantum states. Vectors are usually written using so-called "ket notation" as (column) "ket-vectors" $|x\rangle$. A "bra vector" $\langle x|$, corresponding to $|x\rangle$, denotes the functional by which to each ket-vector $|y\rangle$ the complex number $\langle x|y\rangle$ is associated. In other words, bra vectors $\langle x|$ are row vectors – conjugate versions of the column vectors corresponding to $|x\rangle$.

Two quantum states are considered as orthogonal if their inner product is zero. A basis of $\mathcal{H}_n$ is called orthonormal if all its states have norm one and are mutually orthogonal.

Time evolution (computation) of a quantum system is performed by a **unitary operator** and a step of such an evolution can be seen as a multiplication of a vector (state) $|\psi\rangle$ by a **unitary matrix**[1], i.e. as $\boldsymbol{U}|\psi\rangle$. Unitary operators preserve norms and their eigenvalues having absolute value 1.

If a Hilbert space $\mathcal{H}_A$ corresponds to a quantum system of Alice and a Hilbert space $\mathcal{H}_B$ corresponds to a quantum system of Bob, then the tensor product $\mathcal{H}_A \otimes \mathcal{H}_B$ of $\mathcal{H}_A$ and $\mathcal{H}_B$ corresponds to the composed quantum system of Alice and Bob. (If the Hilbert space $\mathcal{H}_A$ ($\mathcal{H}_B$) has, as a vector space, basis $\{\alpha_I\}_I$ ($\{\beta_j\}_j$), then the tensor products of vectors[2] $\{\alpha_I \otimes \beta_j\}_{I,j}$ form the basis of $\mathcal{H}_A \otimes \mathcal{H}_B$.

[1] Matrix $\boldsymbol{U}$ is **unitary** if $\boldsymbol{U} \cdot \boldsymbol{U}^\dagger = \boldsymbol{U}^\dagger \cdot \boldsymbol{U} = I$, where $\boldsymbol{U}^\dagger$ is a matrix obtained from $\boldsymbol{U}$ by transposition and complex conjugation of its elements.

[2] Tensor product of two vectors $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$ is the mn-dimensional vector $\boldsymbol{x} \otimes \boldsymbol{y}$ with components $(x_1 y_1, \ldots, x_1 y_m, x_2 y_1, \ldots, x_n y_m)$. Tensor

Tensor product of n copies of two-dimensional Hilbert spaces (qubits) is said to be an n-qubit quantum register. A general state of an n-qubit register has the form

$$|\phi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle, \quad \left(\sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1 \right),$$

where $\{|x\rangle\}_{x\in\{0,1\}^n}$ represents a set of 2^n basis states in $\mathcal{H}_{2^n}$. A multiplication by a $2^n \times 2^n$ matrix corresponds then to a quantum computation step on n qubit registers.

In so-called **open quantum systems**, in which there is an interaction of the system with the environment, the very basic concepts are that of **mixed states** and of **density matrices**.

A mixed state is a probability distribution on pure states – what correspond to the case that a source produces various pure states, each with a fixed probability.

A density matrix ρ is defined by $\rho = \sum_{I=1}^{k} \lambda_I |\phi_I\rangle\langle\phi_I|$, where $|\phi_I\rangle$ are pure states and $\sum_{I=1}^{k} \lambda_I = 1$, with all $\lambda_I > 0$. Such a density matrix is said to correspond to the mixed state at which the pure state $|\phi_I\rangle$ is produced with the probability λ_I.

The set of all density matrices of a Hilbert space $\mathcal{H}$ will be denoted by $\mathcal{S}(\mathcal{H})$. The following two operations on (density) matrices are of key importance for the theory of open quantum systems.

Trace of a matrix $\boldsymbol{X}$, notation $\operatorname{Tr} \boldsymbol{X}$, is the sum of diagonal elements of $\boldsymbol{X}$, which is a single number. One physical meaning of the trace operation is that the average value of the measurement of a mixed state ρ with respect to an observable $\mathcal{A}$ is given by $\operatorname{Tr} \mathcal{A}\rho$.

Tracing out of a system of Bob from a density matrix $\rho \in \mathcal{H}_A \otimes \mathcal{H}_B$, notation $\rho_A = \operatorname{Tr}_B \rho$, is a density matrix that shows how the state ρ is seen from an observer that sees only the system Alice.[3]

An alternative way is to define a density matrices as the matrix satisfying the following conditions: (I) ρ is a linear operator on $\mathcal{H}$, (ii) $\rho = \rho^\dagger \geq 0$, (iii) $\operatorname{Tr} \rho = 1$.

1.2 Basic Definitions Concerning Entanglement

The definition of entanglement of pure bipartite states is simple and natural. A pure state $|\phi\rangle$ of a bipartite quantum system $\mathcal{H}_A \otimes \mathcal{H}_B$ is **entangled**, if $|\phi\rangle$ is not a tensor product of a pure state from $\mathcal{H}_A$ and a pure state from $\mathcal{H}_B$. An example of such a state is the so called *EPR state*

product of two matrices, $\boldsymbol{A} = (a_{ij})$ of dimension n and $\boldsymbol{B} = (b_{kl})$ of dimension m, is the matrix $\boldsymbol{C}$, of dimension mn, composed of matrices $c_{ij} = a_{ij}\boldsymbol{B}$.

[3] Technically, if dimension of $\mathcal{H}_A$ ($\mathcal{H}_B$) is n (m), then ρ_A is obtained from ρ, if, in the decomposition of ρ into submatrices of dimension m, each submatrix is replaced by its trace, that is by a single number.

$$|\mathrm{EPR}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

of the quantum system $\mathcal{H}_2 \otimes \mathcal{H}_2$.

The case of entanglement of mixed states of bipartite systems seems to be slightly less natural. A mixed state ρ is **entangled** if ρ cannot be written as a convex combination of the tensor products of mixed states

$$\rho = \sum_{I=1}^{k} p_I \, \rho_{A,I} \otimes \rho_{B,I},$$

where $\rho_{A,I}$ ($\rho_{B,I}$) are mixed states of the quantum system $\mathcal{H}_A$ ($\mathcal{H}_B$).

Actually, in both definitions there is the same idea behind: a state is entangled if it cannot be created by two parties provided they perform quantum operations only on their own subsystems and, in addition, they communicate only classically.

Both of the definitions generalized naturally also to the multipartite case, but we will not do that here. We only comment that entanglement of multipartite states is a much more complex issue and that it is even much less clear how to define measures of entanglement for multipartite states. One natural way to deal with this problem is to reduce, somehow, at least to some extent, the study of multipartite entanglement to bipartite one. In such a case measures of entanglement of bipartite states are again of importance.

1.3 Entanglement Measures for Pure Bipartite States

In the case of pure bipartite states, the situation seemed to be quite easy. It has been believed that in such a case there is only one good measure of entanglement. Namely, von Neumann reduced entropy between $\mathcal{H}_A$ and $\mathcal{H}_B$, notation $E(\cdot)$, is defined as follows for $\rho \in \mathcal{S}(\mathcal{H}_A \otimes \mathcal{H}_B)$.

Definition.

$$E(|\psi\rangle) := S_{\mathrm{vN}}(\mathrm{Tr}_A \, |\psi\rangle\langle\psi|) = S_{\mathrm{vN}}(\mathrm{Tr}_B \, |\psi\rangle\langle\psi|),$$

where von Neumann entropy S_{vN} of a density matrix ρ is defined as $S_{\mathrm{vN}}(\rho) = -\mathrm{Tr}\,\rho \log_2 \rho$. It is equal to $-\sum_I \lambda_I \log_2 \lambda_I$ where λ_I are eigenvalues of ρ.

The properties of von Neumann reduced entropy are well known. Here are some of them:

- Invariance condition : $S_{\mathrm{vN}}(\rho) = S_{\mathrm{vN}}(U \rho U^\dagger)$ for any unitary matrix U.
- Additivity : $S_{\mathrm{vN}}(\rho_1 \otimes \rho_2) = S_{\mathrm{vN}}(\rho_1) + S_{\mathrm{vN}}(\rho_2)$.[4]

[4] $S_{\mathrm{vN}}(\rho_1 \otimes \rho_2)$ is considered as the entropy between $\mathcal{H}_A^{\otimes 2} = \mathcal{H}_A \otimes \mathcal{H}_A$ and $\mathcal{H}_B^{\otimes 2} = \mathcal{H}_B \otimes \mathcal{H}_B$, i.e. between Alice's side and Bob's side, where $\rho_1 \otimes \rho_2$ is a mixed state of $(A \otimes B)^{\otimes 2}$. The same argument will be needed for the entropy of $\rho^{\otimes n} = \rho \otimes \cdots \otimes \rho$, which will appear later.

– Subadditivity : For a density matrix ρ of a bipartite system it holds $S_{\mathrm{vN}}(\rho) \leq S_{\mathrm{vN}}(\rho_A) + S_{\mathrm{vN}}(\rho_B) \leq S_{\mathrm{vN}}(\rho_A \otimes \rho_B)$, where $\rho_A = \mathrm{Tr}_B\, \rho, \rho_B = \mathrm{Tr}_A\, \rho$.

As discussed later, it has been shown that the above entanglement measure $E(\cdot)$ is in a sense unique [1]. Namely, that it is the uniquely defined measure of entanglement satisfying the first five from the following set of axioms for entanglement measures.

Axioms.

Separability: $E(\rho) = 0$ if ρ is separable.
Normalization: $E(\rho) = 1$ if ρ is a Bell state (maximally entangled two qubits).
Monotonicity: $E(\Lambda(\rho)) \leq E(\rho)$ if Λ is a LQCC.[5]
Continuity: Let $\{A_I\}_{I\geq 0}$ and $\{B\}_{I\geq 0}$ be a sequence of Hilbert spaces and ρ_I, σ_I be density matrices of Hilbert spaces $A_I \otimes B_I$ such that $||\rho_I - \sigma_I||_1 \to 0$ for $I \to \infty$, then

$$\frac{E(\rho_I) - E(\sigma_I)}{1 + \log \dim(A_I \otimes B_I)} \to 0 \ (I \to \infty).$$

Additivity: $E(\rho^{\otimes n}) = nE(\rho)$.
Subadditivity: $E(\rho \otimes \sigma) \leq E(\rho) + E(\sigma)$.
Regularization: The following limit exists.

$$E^{\infty}(\rho) = \lim_{n\to\infty} \frac{E(\rho^{\otimes n})}{n}.$$

1.4 Measures of Entanglement for Mixed Bipartite States

For mixed bipartite states, a variety of measures of entanglement has already been defined. Perhaps the main ones are *entanglement of formation*, *entanglement of distillation*, *entanglement of relative entropy* and *concurrence*. **Entanglement of formation** is defined through the average of von Neumann reduced entropy as

$$E_f(\rho) := \inf \sum_j p_j E(|\phi_j\rangle),$$

where the infimum is taken over all possible pure-state decompositions of $\rho = \sum_j p_j |\phi_j\rangle\langle\phi_j|$. **Entanglement of distillation**, $E_d(\rho)$, is the average amount of maximally entangled states that can be distilled from several copies of ρ.

Another important measures is **entanglement of relative entropies** $E_r^D(\rho)$. They count the minimal "distance" of ρ to a set D of states (that are separable or not-distillable, or have a similar property). These measures have interesting properties and serve as upper (lower) bounds for entanglement of distillation (formation).

All these measures of entanglement are heavy to compute because they require some optimization. In the case of bipartite systems of dimension $2 \otimes 2$ and

[5] LQCC stands for Local Quantum operations + Classical Communications.

$3 \otimes 2$ called **concurrence** due to Wootters [6,7], of large importance is easily computable, defined as follows:

$$C(\rho) = \max(\lambda_1 - \lambda_2 - \lambda_3 - \lambda_4\ , 0),$$

where λ_i are, in the descending order, eigenvalues of the matrix $\rho\hat{\rho}$ where $\hat{\rho} = (\sigma_y \otimes \sigma_y)\rho^*(\sigma_y \otimes \sigma_y)$, and $\sigma_y = \left(\begin{smallmatrix} 0 & -\mathrm{I} \\ \mathrm{I} & 0 \end{smallmatrix}\right)$ is a Pauli matrix. ρ^* is a matrix each element of which is the complex conjugate of the corresponding element of the matrix ρ.

Importance of this measure stems from the fact that it allows to compute entanglement of formation of a density matrices rather easily, according to the rule:

$$E_f(\rho) = H\left(\frac{1 + \sqrt{1 - C^2(\rho)}}{2}\right),$$

where $H(x)$ is the binary entropy function, i.e. $H(x) = -x\log_2 x - (1-x)\log_2(1-x)$.

These measures of $E_f(\cdot)$, $E_r^D(\cdot)$ have so-called asymptotic version, defined by $E_\cdot^\infty(\rho) = \lim_{n\to\infty} \frac{E_\cdot(\rho^{\otimes n})}{n}$. Asymptotic version of the entanglement of formation is known to be equal to so-called **cost of entanglement**[8], and it is denoted by E_c.

It is natural to ask when non-asymptotic and asymptotic versions of a measure of entanglement are identical. This is surely the case if a measure of entanglement is additive. That is if it holds $E(\rho_1 \otimes \rho_2) = E(\rho_1) + E(\rho_2)$ for density matrices.

1.5 Known Results Concerning Additivity of Entanglement of Formation

Concerning the additivity of entanglement of formation, only a few results have been known. Vidal et al. [3] showed that additivity holds for certain states supported on $\mathcal{H}_3 \otimes \mathcal{H}_6$ and this is the non-trivial first example. Matsumoto et al. [4] showed that additivity of entanglement of formation holds for a family of mixed states supported on $\mathcal{H}_d \otimes \mathcal{H}_{d^2}$ by utilizing the additivity for unital qubit channels [5].

2 New Additivity Result

In this section we prove our main result, namely that entanglement of formation is additive for tensor product of two three-dimensional bipartite antisymmetric states.

Actually, we give only a sketch of the proof in this section. Detailed proofs, quite involved, of two main technical facts are given in Appendix.

Let us start with an introduction of our notations and concepts. $\mathcal{H}_-$ will stand for an antisymmetric Hilbert space, which is a subspace of a bipartite

Hilbert space $\mathcal{H}_{AB} := \mathcal{H}_A \otimes \mathcal{H}_B$, where both $\mathcal{H}_A$ and $\mathcal{H}_B$ are 3−dimensional Hilbert spaces, spanned by basic vectors $\{|i\rangle\}_{i=1}^3$. It is already known [2] that $\mathcal{H}_-$ is three-dimensional Hilbert space, spanned by states $\{|i,j\rangle\}_{ij=23,31,12}$, where the state $|i,j\rangle$ is defined as $\frac{|i\rangle|j\rangle-|j\rangle|i\rangle}{\sqrt{2}}$.[6] Let $\mathcal{H}_-^{\otimes n}$ be the tensor product of n copies of $\mathcal{H}_-$. These copies will be discriminated by the upper index as $\mathcal{H}_-^{(j)}$, for $j = 1 \ldots n$. $\mathcal{H}_-^{(j)}$ will then be an antisymmetric subspace of $\mathcal{H}_A^{(j)} \otimes \mathcal{H}_B^{(j)}$.

2.1 Proof of the Main Result

It has been shown in [2] that $E_f(\rho) = 1$ for any mixed state $\rho \in \mathcal{S}(\mathcal{H}_-)$. This result will play the key role in our proof. We prove now that :

Theorem.

$$E_f(\rho_1 \otimes \rho_2) = E_f(\rho_1) + E_f(\rho_2) = 2 \tag{1}$$

for any $\rho_1, \rho_2 \in \mathcal{S}(\mathcal{H}_-)$.

Proof. To prove this theorem, it is sufficient to show that

$$E_f(\rho_1 \otimes \rho_2) \geq 2 \tag{2}$$

since the subadditivity $E_f(\rho_1 \otimes \rho_2) \leq E_f(\rho_1) + E_f(\rho_2) = 2$ is trivial.[7] To prove (2), we first show that

$$E(|\psi\rangle\langle\psi|) \geq 2, \text{ for any pure state } |\psi\rangle \in \mathcal{H}_-^{\otimes 2}. \tag{3}$$

Using the Schmidt decomposition, the state $|\psi\rangle$ can be decomposed as follows:

$$|\psi\rangle = \sum_{i=1}^{3} \sqrt{p_i}\, |\psi_i^{(1)}\rangle \otimes |\psi_i^{(2)}\rangle,$$

where $p_1, p_2, p_3 > 0, p_1 + p_2 + p_3 = 1$, and $\{|\psi_i^{(j)}\rangle\}_{i=1}^3$ is an orthonormal basis of the Hilbert space $\mathcal{H}_-^{(j)}$, for $j = 1, 2$.[8] First, we will use the following fact, to be proven in the Appendix.

---

[6] The space $\mathcal{H}_-$ is called antisymmetric because by swapping the position of two qubits in any of its states $|\psi\rangle$ we get the state $-|\psi\rangle$.

[7] Indeed, it holds $E_f(\rho_1 \otimes \rho_2) = \inf \sum p_i E(|\psi_i\rangle\langle\psi_i|) \leq \inf \sum p_i^{(1)} p_i^{(2)} E(|\psi_i^{(1)}\rangle\langle\psi_i^{(1)}| \otimes |\psi_i^{(2)}\rangle\langle\psi_i^{(2)}|) = \inf \sum p_i^{(1)} E(|\psi_i^{(1)}\rangle\langle\psi_i^{(1)}|) + \inf \sum p_i^{(2)} E(|\psi_i^{(2)}\rangle\langle\psi_i^{(2)}|) = E_f(\rho_1) + E_f(\rho_2)$ where $(p_i^{(j)}, |\psi_i^{(j)}\rangle)$ are subject to the condition of $\rho_j = \sum_i p_i^{(j)} |\psi_i^{(j)}\rangle\langle\psi_i^{(j)}|$.

[8] Note that this Schmidt decomposition is with respect to $\mathcal{H}_-^{(1)} : \mathcal{H}_-^{(2)}$, or, it could be said that with respect to $\left(\mathcal{H}_A^{(1)} \otimes \mathcal{H}_B^{(1)}\right) : \left(\mathcal{H}_A^{(2)} \otimes \mathcal{H}_B^{(2)}\right)$, not with respect to $\left(\mathcal{H}_A^{(1)} \otimes \mathcal{H}_A^{(2)}\right) : \left(\mathcal{H}_B^{(1)} \otimes \mathcal{H}_B^{(2)}\right)$, where ":" indicates how to separate the system into two subsystems for the decomposition.

Lemma 1. *If $\{|\psi_i\rangle\}_{i=1}^3$ is an orthonormal basis of $\mathcal{H}_-$, then there exists an unitary operator U, acting on both $\mathcal{H}_A$ and $\mathcal{H}_B$, such that $U \otimes U$ maps the states $|\psi_1\rangle, |\psi_2\rangle, |\psi_3\rangle$ into the states $|2,3\rangle, |3,1\rangle, |1,2\rangle$, respectively.*

Therefore, by Lemma 1, there exist unitary operators $U^{(1)}, U^{(2)}$ such that

$$\left(U^{(1)} \otimes U^{(1)} \otimes U^{(2)} \otimes U^{(2)}\right)|\psi\rangle = \sum_{ij=23,31,12}^{i,j} \sqrt{p_{ij}}\, |i,j\rangle \otimes |i,j\rangle \;=:\; |\psi'\rangle,$$

where $p_{23} := p_1,\ p_{31} := p_2,\ p_{12} := p_3$.

As the next, we use the following fact, also to be proven in Appendix.

Lemma 2.

$$E(|\psi'\rangle\langle\psi'|) \geq 2, \quad \textit{if} \quad \begin{cases} p_{23}, p_{31}, p_{12} \geq 0 \\ p_{23} + p_{31} + p_{12} = 1 \end{cases}.$$

Local unitary operators do not change von Neumann reduced entropy, and therefore $E(|\psi\rangle\langle\psi|) = E(|\psi'\rangle\langle\psi'|) \geq 2$. That is, the claim (3) is proven.

We are now almost done. Indeed, the entanglement of formation is defined as

$$E_f(\rho) = \inf_{[(p_i,\psi_i)]_i \in \Delta(\rho)} \sum_i p_i E(|\psi_i\rangle\langle\psi_i|)$$

where

$$\Delta(\rho) = \left\{ [(p_i,\psi_i)]_i \;\middle|\; \begin{array}{l} \sum_i p_i = 1, p_i > 0 \text{ for all } i \\ \sum_i p_i |\psi_i\rangle\langle\psi_i| = \rho, \langle\psi_i|\psi_i\rangle = 1 \text{ for all } i \end{array} \right\}$$

and it is known that all $|\psi_i\rangle$ induced from $\Delta(\rho)$ satisfy $|\psi_i\rangle \in \text{Range}(\rho)$, where $\text{Range}(\rho)$ is sometimes called the image space of the matrix ρ, which is the set of $\rho|\psi\rangle$ with $|\psi\rangle$ running over the domain of ρ. Hence

$$E_f(\rho) \geq \inf\left\{ E(|\psi\rangle\langle\psi|) \;\middle|\; |\psi\rangle \in \text{Range}(\rho), \langle\psi|\psi\rangle = 1 \right\}.$$

Since $\rho_1 \otimes \rho_2 \in \mathcal{S}(\mathcal{H}_-^{\otimes 2})$, $\text{Range}(\rho_1 \otimes \rho_2) \subseteq \mathcal{H}_-^{\otimes 2}$, henceforth (2) is proven. Therefore (1) have been shown. □

3 Conclusions and Discussion

Additivity of the entanglement of formation for two three-dimensional bipartite antisymmetric states has been proven in this paper. The next goal could be to prove additivity for more than two antisymmetric states. Perhaps the proof can utilize the lower bound value of the reduced von Neumann entropy. Of course, the main goal is to show that entanglement of formation is additive, in general. However, this seems to be a very hard task.

A. Appendix

We provide here proofs of two facts used in the proof of our main result.

Lemma 1. *If $\{|\psi_i\rangle\}_{i=1}^3 \subset \mathcal{H}_-$ is an orthonormal basis, there exists an unitary operator U, acting on both $\mathcal{H}_A$ and $\mathcal{H}_B$, such that $U \otimes U$ maps the states $|\psi_1\rangle, |\psi_2\rangle, |\psi_3\rangle$ into the states $|2,3\rangle, |3,1\rangle, |1,2\rangle$, respectively.*

Proof. Let us start with notational conventions. In the following, ${}^T\square$ stands for the transpose of a matrix, $\square^*$ stands for taking complex conjugate of each element of a matrix, $\square^\Theta$ denotes the transformation defined later.

Let U be represented as $\left(\begin{smallmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{smallmatrix}\right)$ with respect to the basis $|1\rangle, |2\rangle, |3\rangle$.[9] Lengthy calculations show that when a 9×9 dimensional matrix $U \otimes U$ is considered as mapping from $\mathcal{H}_-$ into $\mathcal{H}_-$, it can be represented by the following 3×3 dimensional matrix, with respect to the basis $|2,3\rangle, |3,1\rangle, |1,2\rangle$,

$$U^\Theta := \begin{pmatrix} u_{22}u_{33} - u_{23}u_{32} & u_{23}u_{31} - u_{21}u_{33} & u_{21}u_{32} - u_{22}u_{31} \\ u_{32}u_{13} - u_{33}u_{12} & u_{33}u_{11} - u_{31}u_{13} & u_{31}u_{12} - u_{32}u_{11} \\ u_{12}u_{23} - u_{13}u_{22} & u_{13}u_{21} - u_{11}u_{23} & u_{11}u_{22} - u_{12}u_{21} \end{pmatrix}.$$

One can then show that

$$U^\Theta \cdot {}^T U = (\det U) \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right),$$

and by multiplying with U^* from the right in the above equation, one obtain $U^\Theta = (\det U) \cdot U^*$, since U is an unitary matrix, and ${}^T U \cdot U^*$ is equal to the identity matrix.

Since $\{|\psi_i\rangle\}_{i=1,2,3}$ is an orthonormal basis of $\mathcal{H}_-$, there exists an unitary operator on $\mathcal{H}_-$ such that $|\psi_1\rangle \mapsto |2,3\rangle, |\psi_2\rangle \mapsto |3,1\rangle, |\psi_3\rangle \mapsto |1,2\rangle$, and let Θ_ψ be the corresponding matrix with respect to the basis $\{|i,j\rangle\}_{ij=23,31,12}$.

Let $U_\psi := (\det \Theta_\psi)^{\frac{1}{2}} \cdot \Theta_\psi^*$.[10] It holds $U_\psi^\Theta = \Theta_\psi$. [11] Therefore $U_\psi \otimes U_\psi = U'_\psi$. The operator U_ψ is the one needed to satisfy the statement of Lemma 1. □

Lemma 2.

$$E(|\psi'\rangle\langle\psi'|) \geq 2 \quad \textit{if} \quad \begin{cases} |\psi'\rangle = \sum_{ij=23,31,12}^{i,j} \sqrt{p_{ij}}\, |i,j\rangle|i,j\rangle \\ p_{23}, p_{31}, p_{12} \geq 0 \\ p_{23} + p_{31} + p_{12} = 1 \end{cases}.$$

[9] For mathematicians, an operator and its matrix representation are different objects, but for convenience, we identify U with $\left(\begin{smallmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{smallmatrix}\right)$.

[10] In the above definition it does not matter which of two roots of $\det \Theta_\psi$ are taken

[11] Indeed, $U_\psi^\Theta = (\det U_\psi) U_\psi^* = (\det \Theta_\psi)^{\frac{3}{2}} \det \Theta_\psi^* \cdot ((\det \Theta_\psi)^{\frac{1}{2}})^* \Theta_\psi = \Theta_\psi$. Note that $\det U_\psi = \det(\det(\Theta_\psi)^{\frac{1}{2}}\, \Theta_\psi^*) = (\det \Theta_\psi)^{\frac{3}{2}} \det \Theta_\psi^*$ because Θ_ψ^* is a 3×3 matrix.

Proof. Let $p_{32} := p_{23}, p_{13} := p_{31}, p_{21} := p_{12}$. Then it holds,

$$\begin{aligned}
|\psi'\rangle &= \sum_{1\le i<j\le 3}^{i,j} \sqrt{p_{ij}}\,|i,j\rangle|i,j\rangle \\
&= \frac{1}{2}\sum_{1\le i<j\le 3}^{i,j} \sqrt{p_{ij}}\,\{|ii;jj\rangle - |ij;ji\rangle - |ji;ij\rangle + |jj;ii\rangle\} \\
&= \frac{1}{2}\sum_{1\le i\ne j\le 3}^{i,j} \sqrt{p_{ij}}\,\{|ii;jj\rangle - |ij;ji\rangle\},
\end{aligned}$$

where $|i_1 i_2; i_3 i_4\rangle$ denotes the tensor product $|i_1\rangle\otimes|i_2\rangle\otimes|i_3\rangle\otimes|i_4\rangle$, $|i_1\rangle \in \mathcal{H}_A^{(1)}$, $|i_2\rangle \in \mathcal{H}_A^{(2)}$, $|i_3\rangle \in \mathcal{H}_B^{(1)}$ and $|i_4\rangle \in \mathcal{H}_B^{(2)}$, and the condition $1 \le i \ne j \le 3$ actually means "$1 \le i \le 3$ and $1 \le j \le 3$ and $i \ne j$". This convention will be used also in the following.

We are now going to calculate the reduced matrix of $|\psi'\rangle\langle\psi'|$, which we will denote as Ξ, and it will be decomposed into the direct sum as follows.

$$\begin{aligned}
\Xi &:= \operatorname*{Tr}_{\mathcal{H}_B^{(1)}\otimes\mathcal{H}_B^{(2)}} |\psi'\rangle\langle\psi'| \\
&= \frac{1}{4}\sum_{\substack{1\le i\ne j\le 3\\ 1\le k\ne l\le 3}}^{i,j,k,l} \sqrt{p_{ij}p_{kl}} \operatorname*{Tr}_{\mathcal{H}_B^{(1)}\otimes\mathcal{H}_B^{(2)}} \left(\begin{array}{l} |ii;jj\rangle\langle kk;ll| - |ii;jj\rangle\langle kl;lk| \\ \quad - |ij;ji\rangle\langle kk;ll| + |ij;ji\rangle\langle kl;lk| \end{array} \right) \\
&= \frac{1}{4}\sum_{\substack{1\le i\ne j\le 3\\ 1\le k\ne l\le 3}}^{i,j,k,l} \sqrt{p_{ij}p_{kl}} \operatorname*{Tr}_{\mathcal{H}_B^{(1)}\otimes\mathcal{H}_B^{(2)}} \left(|ii;jj\rangle\langle kk;ll| + |ij;ji\rangle\langle kl;lk|\right) \\
&= \frac{1}{4}\sum_{\substack{1\le i\ne k\le 3\\ 1\le j\ne k\le 3}}^{i,j,k} \sqrt{p_{ik}p_{jk}}\,|ii\rangle\langle jj| + \frac{1}{4}\sum_{1\le i\ne j\le 3}^{i,j} p_{ij}\,|ij\rangle\langle ij| \\
&\cong \frac{1}{4}\begin{pmatrix} p_{12}+p_{13} & \sqrt{p_{13}p_{23}} & \sqrt{p_{12}p_{23}} \\ \sqrt{p_{13}p_{23}} & p_{12}+p_{23} & \sqrt{p_{12}p_{13}} \\ \sqrt{p_{12}p_{23}} & \sqrt{p_{12}p_{13}} & p_{13}+p_{23} \end{pmatrix} \oplus \frac{1}{4}(p_{12})^{\oplus 2} \oplus \frac{1}{4}(p_{13})^{\oplus 2} \oplus \frac{1}{4}(p_{23})^{\oplus 2}, \qquad (4)
\end{aligned}$$

where $\oplus$ denotes the direct sum of matrices, and $\square^{\oplus n}$ denotes the direct sum of n copies of the same matrix.

We need to get eigenvalues of Ξ in order to calculate reduced von Neumann entropy

$$E(|\psi'\rangle\langle\psi'|) = -\operatorname{Tr}(\Xi\log_2\Xi) = -\sum_{\lambda:\text{e.v.of }\Xi} \lambda\log_2\lambda.$$

In this case, fortunately, the eigenvalues can be determined explicitly from the expression (4). They are the following ones:

$$\left(\frac{1-\cos\theta}{6}, \frac{1-\cos(\theta+\frac{2\pi}{3})}{6}, \frac{1-\cos(\theta+\frac{4\pi}{3})}{6}, \frac{p_{12}}{4}, \frac{p_{12}}{4}, \frac{p_{13}}{4}, \frac{p_{13}}{4}, \frac{p_{23}}{4}, \frac{p_{23}}{4}\right) \qquad (5)$$

for a certain $-\frac{\pi}{3} < \theta \leq \frac{\pi}{3}$.[12] These eigenvalues are denoted as $(\lambda_1, \lambda_2, \ldots, \lambda_9)$, respectively. Although $\lambda_4, \ldots, \lambda_9$ are trivial, $\lambda_1, \lambda_2, \lambda_3$ are the roots of the cubic polynomial

$$g(\lambda) := \lambda^3 - \frac{1}{2}\lambda^2 + \frac{1}{16}\lambda - \frac{p_{12}\, p_{13}\, p_{23}}{16} \tag{6}$$

that is the characteristic polynomial function of the cubic matrix that appeared in the expression (4). We must solve this cubic equation to obtain (5). The cubic equation $g(\lambda) = 0$ is in Cardan's irreducible form,[13] because Ξ is the density matrix. In such a case, the roots of the cubic equation are

$$\alpha + \beta\cos\theta, \alpha + \beta\cos(\theta + \frac{2\pi}{3}), \alpha + \beta\cos(\theta + \frac{4\pi}{3}). \tag{7}$$

One can easily show that $\lambda_1 + \lambda_2 + \lambda_3 = 3\alpha$, and $\lambda_1^2 + \lambda_2^2 + \lambda_3^2 = 3\alpha^2 + \frac{3}{2}\beta^2$. If $\lambda_1, \lambda_2, \lambda_3$ are equal to the roots of the cubic equation $\lambda^3 + a_1\lambda^2 + a_2\lambda + a_3 = 0$, then $\lambda_1 + \lambda_2 + \lambda_3 = -a_1$, $\lambda_1^2 + \lambda_2^2 + \lambda_3^2 = a_1^2 - 2a_2$. Taking $a_1 = -\frac{1}{2}, a_2 = \frac{1}{16}$ from the expression (6), we get the following system of equations $3\alpha = \frac{1}{2}, 3\alpha^2 + \frac{3}{2}\beta^2 = \frac{1}{8}$, and $(\alpha, \beta) = \left(\frac{1}{6}, -\frac{1}{6}\right)$ is sufficient. Applying into (7), we complete (5).

Our idea is now to show that

$$E(|\psi'\rangle\langle\psi'|) = \sum_{i=1}^{9}(-\lambda_i \log_2 \lambda_i) \geq 2. \tag{8}$$

This will be shown if we prove that it holds

$$\sum_{i=1}^{3}(-\lambda_i \log_2 \lambda_i) \geq 1 \text{ and } \sum_{i=4}^{9}(-\lambda_i \log_2 \lambda_i) \geq 1. \tag{9}$$

The second inequalities is easy to verify by simple calculations. To finish the proof of the lemma we therefore need to show that

$$\sum_{i=1}^{3}(-\lambda_i \log_2 \lambda_i) \geq 1. \tag{10}$$

Without loss of generality, we assume $\theta \in \left[0, \frac{\pi}{3}\right]$.[14] Clearly, $\lambda_1 \in \left[0, \frac{1}{12}\right]$ and $\lambda_2, \lambda_3 = \frac{1}{4} - \frac{\lambda_1 \pm \sqrt{\lambda_1 - 3\lambda_1^2}}{2} \in \left[\frac{1}{12}, \frac{1}{3}\right]$ (λ_2, λ_3 can be regarded as the solution of the following systems of equations: $\lambda_1 + \lambda_2 + \lambda_3 = \frac{1}{2}, \lambda_1^2 + \lambda_2^2 + \lambda_3^2 = \frac{1}{8}$) . You can also show that

$$-z\log_2 z \geq \begin{cases} (\log_2 12)\, z & \text{if } z \in \left[0, \frac{1}{12}\right] \\ \frac{1}{2} + \frac{\log_e 4 - 1}{\log_e 2}(z - \frac{1}{4}) - 4(z - \frac{1}{4})^2 & \text{if } z \in \left[\frac{1}{12}, \frac{1}{3}\right]. \end{cases} \tag{11}$$

[12] The exact value of θ will be no importance for us.

[13] A cubic equation is said to be in Cardan's irreducible form if its three roots are real.

[14] The sequence of $\{\lambda_i\}_{i=1}^3$ doesn't change if θ is replaced by $-\theta$. Thus we can change the assumption $\theta \in \left(-\frac{\pi}{3}, \frac{\pi}{3}\right]$, into $\theta \in \left[0, \frac{\pi}{3}\right]$.

The first inequality of (11) is easily confirmed. On the other hand, one way of the proof of the second inequality is as follows: Let $f(z) := \left(-z\log_2 z\right) - \left(\frac{1}{2} + \frac{\log_e 4-1}{\log_e 2}(z-\frac{1}{4}) - 4(z-\frac{1}{4})^2\right)$. Differentiating this expression by z once and twice, we can get the increasing and decreasing table as follows.

z	$\frac{1}{12}$		$\frac{1}{8\log_e 2}$		$\frac{1}{4}$		$\frac{1}{3}$
$f(z)$	$+$	$\curvearrowright$	$+$	$\searrow$	0	$\nearrow$	
$f'(z)$				$-$	0	$+$	
$f''(z)$		$-$	0	$+$	$+$	$+$	

The table indicates $f(z) \geq 0$ for $z \in \left[\frac{1}{12}, \frac{1}{3}\right]$. Now we indeed get the lower bounds by polynomial functions.

Combining all of the above inequalities we get (10) as

$$-\sum_{i=1}^{3} \lambda_i \log_2 \lambda_i \geq 1 + \left(\frac{\log_e 3 + 2}{\log_e 2} - 2\right)\lambda_1 + 4\lambda_1^2 \geq 1 \quad .$$

so that (9) and (8) are successively shown and that our proof is finished. □

References

1. Matthew J. Donald, Michal Horodecki, Oliver Rudolph, The uniqueness theorem for entanglement measures (2001), quant-ph/0105017.
2. G. Vidal, W. Dür, J. I. Cirac, Entanglement cost of antisymmetric states (2002), quant-ph/0112131, version 2.
3. G. Vidal, W. Dür, J. I. Cirac, Entanglement Cost of Bipartite Mixed States (2002), Physical Review Letters, **89**, 027901.
4. K. Matsumoto, A. Winter, T. Shimono (2002), quant-ph/0206148.
5. Christopher King, Additivity for unital qubit channels (2001), quant-ph/0103156.
6. Scott Hill, William K. Wootters, Entanglement of a Pair of Quantum Bits (1997) Physical Review Letters, **78**, 5022-5025; quant-ph/9703041.
7. W. K. Wootters, Entanglement of Formation of an Arbitrary State of Two Qubits (1998), Physical Review Letters, **80**, 2245-2248; quant-ph/9709029.
8. P. Hayden, M. Horodecki, B. Terhal, The Asymptotic Entanglement Cost of Preparing a Quantum State, Journal of Physics A: Mathematical and General **34** (2001) 6891-6898; quant-ph/0008134.

Membrane Computing: When Communication Is Enough

Petr Sosík and Jiří Matýsek

Institute of Computer Science, Silesian University, 746 01 Opava, Czech Republic
petr.sosik@fpf.slu.cz

Abstract. We study the computational power of P system, the mathematical model of cellular membrane systems whose operations are motivated by some principles of regulated transfer of objects (molecules) through membranes and simple mutual reactions of these objects.
The original model of P system describes several possible types of operations applicable to these objects, resulting in universal computational power. We show that P systems with symbol objects keep their universal computational power even if we restrict ourselves to catalyzed transport of objects through labelled membranes without their change or mutual reactions. Each transport operation is initiated by a complex of at most two objects. Moreover we do not need some other mathematical tools of P systems like priorities of operators or dissolution or creation of membranes to reach the universal computational power.
In the second part of the paper we present a communicating P-system computing optimal parallel algorithm for finding maximum of a given set of integers. We therefore demonstrate that despite the simplicity of the model, it is (theoretically) capable to solve nontrivial computing tasks in a highly parallel and effective way.

1 Introduction

Due to the recent progress in biotechnologies and physics (especially in the quantum theory), interests of researchers in computer science is attracted to unconventional computing models inspired from nature. In natural information processing we can often see degree of parallelism, effectivity and robustness unreachable by our machines *in silico*. An overview of some recent models and ideas of "natural computers" can be found e.g. in [1], [11].

This paper deals with theoretical computing power of so-called *P systems*, computing models inspired from nature, based on the idea that membranes play a key role in cellular information system. They were introduced by Gheorghe Păun in [10] and [9] and until now more than 160 papers concerning P systems appeared. For an up-to-date information the reader is referred to www pages at `http://dna.bio.disco.unimib.it/psystems/`.

The basic ingredient of a P system is a topological membrane structure consisting of membranes hierarchically embedded in the outermost *skin* membrane. Every membrane encloses a *region* possibly containing other membranes; the

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 264–275, 2002.

part delimited by the membrane labelled by k and its inner membranes is called *compartment* k. A region delimited by a membrane not only may enclose other membranes but also specific objects (used in the multiset sense) and operators — evolution rules for objects.

Any object, alone or together with one more object, evolves, can be at each step transformed in other objects due to evolution rules, can pass through one membrane, and can eventually dissolve the membrane in which it is placed or even create another membrane. All objects evolve at the same time, in parallel; in turn, all membranes are active in parallel. The evolution rules can be hierarchized by a priority relation, given in the form of a partial order relation; always, the rule with the highest priority among the applicable rules is actually applied. If the objects evolve alone, then the system is said to be non-cooperative; if there are rules which specify the evolution of several objects at the same time, then the system is cooperative; an intermediate case is that where certain objects — *catalysts*, appear together with other objects in evolution rules and they are not modified by the use of the rules.

We can distinguish three basic types of P systems due to the mathematical representation of objects: (i) the systems with symbol objects, where each membrane contains a multiset of these objects; (ii) the systems with string objects, where sets of them are taken into the account, and (iii) the systems with "worm" objects combining the principles (i) and (ii), i.e. working with multisets of strings.

The representation of objects determines also the possible types of evolution rules. The number of recently studied variants of P systems is rather large and we refer to [1], [11], or to the above cited web pages for more detailed information.

From the "wet-implementation" point of view, however, some of the operations introduced in this scenario of membrane computing are in practice possible only in certain biochemical context or under certain circumstances and cannot be applied arbitrarily. For instance, the original definitions in [3], [9] and others allow "creative" or "destructive" evolution rules with unbalanced number of objects (representing molecules or ions) on the left- and the right-hand side of the rule. There is also assumed exact common clock synchronizing biochemical reactions within the whole system, or priority relation between the rules regardless on concentration of the objects.

These capabilities may, from some points of view, lead to computationally "too strong" models and, hence, also the properties of P systems with more restrictions are of interest. See e.g. [5] for a proposed set of operations which are used in some successful laboratory experiments of so called aqueous computing. This paper represents another step in this direction introducing a variant of P system with symbol objects, whose operations are restricted to the catalyzed transport of objects through membranes, keeping the universal computational power of the system. A similar approach is studied recently also in [8].

The system starts always in a configuration when each region contains only a finite number of objects. We nevertheless need to allow the presence of an arbitrarily large number of objects within the system (otherwise we'd stay at

the level of finite automaton). This can be done naturally by allowing transport of the objects from the space surrounding the system through the skin membrane and by the assumption that the outer space contains an unlimited number of such objects.

2 Communicating P System

We fix some basic notation first. For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation; the *empty string* is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. Any subset of V^+ is called a *λ-free (string) language*. For a string $w \in V^*$, we denote by $m(w)$ the multiset of symbols w consists of. For more notions as well as basic results from the theory of formal languages, the reader is referred to [2], [12].

Definition 1. *A communicating P system of degree $n, n \geq 1$, is a construct*

$$\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n, i_o),$$

where:

(i) *V is an alphabet; its elements are called objects;*
(ii) *μ is a membrane structure of degree n, with the membranes and the regions labeled in a one-to-one manner by natural numbers;*
(iii) *$w_i, 1 \leq i \leq n$, are strings from V^* representing by their Parikh vectors multisets over V associated with the regions $1, 2, \ldots, n$ of μ;*
(iv) *$R_i, 1 \leq i \leq n$, are finite sets of evolution rules over V associated with the regions $1, 2, \ldots, n$ of μ; An evolution rule is a pair $u \to v$ adopting the form*
  - *$a \to a_\tau$, or*
  - *$ab \to a_{\tau_1} b_{\tau_2}$, or*
  - *$ab \to a_{\tau_1} b_{\tau_2} c_{come}$,*

  where $a, b, c \in V$ and $\tau, \tau_1, \tau_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq n\}$).
(v) *i_o is a number between 1 and n which specifies the output membrane of Π.*

In this paper we restrict ourselves to the membrane structures forming a rooted tree. More general structures as asymmetric graphs are considered in [1].

The components $w_1, \ldots, w_n$ constitute the *initial configuration* of Π. In general, any sequence $w'_1, \ldots, w'_n$, with $m(w'_j)$ multisets over V, $1 \leq j \leq n$, is called a *configuration* of Π. For two configurations $C_1 = (w'_1, \ldots, w'_n)$ and $C_2 = (w''_1, \ldots, w''_n)$ we write $C_1 \Longrightarrow C_2$, and we say that we have a *transition* from C_1 to C_2, if we can pass from C_1 to C_2 by using the evolution rules appearing in $R_1, \ldots, R_n$ in the following manner.

Consider a rule $u \to v$ in a set R_i, $1 \leq i \leq n$. If the objects mentioned by u, with the multiplicities specified by u, appear in w'_i (that is, the multiset identified by u is included in $m(w'_i)$), then these objects can evolve according to the rule $u \to v$. If more than one rule can be applied simultaneously to a single object, one of these rules is nondeterministically chosen.

All objects to which a rule *can* be applied and which are not processed by another rule *must* be the subject of a rule application. All objects in u are "consumed" by using the rule $u \to v$, that is, the multiset identified by u is subtracted from $m(w_i')$. The result of using the rule is determined by v.

- If an object appears in v in a form a_{here}, then it will remain in the same region i. (When specifying rules, instead of a_{here} we often write simply a, the indication "here" is omitted.)
- If an object appears in v in a form a_{out}, then a will exit the membrane i and will become an element of the region immediately outside it (thus, it will be adjacent to the membrane i from which it was expelled). In this way, it is possible that an object leaves the system itself if it goes out of the skin membrane.
- If an object appears in a form a_{in_q}, then a will be added to the multiset $m(w_q')$, providing that a is adjacent to the membrane q. If a_{in_q} appears in v and the membrane q is not one of the membranes contained in the region i, then the application of the rule is not allowed.
- An object in a form a_{come} may appear only in a rule in the set R_1, contained in the region enclosed by the skin membrane. During an application of such a rule, a is imported through the skin membrane from the outer space and will become an element of the region enclosed by the skin membrane.

All these operations are done in parallel, for all possible applicable rules $u \to v$, for all occurrences of multisets u in the region associated with the rules, for all regions at the same time. The system continues these parallel steps until there remain any applicable rules in any compartment of Π. The multiset of objects contained in the output membrane i_o at the moment when the system halts is considered as the *result* of the computation of Π. For an example of communicating P system see section 5.

3 Register Machine

In this section we briefly recall the concept of the Minsky register machine. Minsky showed e.g. in [7] that the universal computational power can be reached by the abstract machine using a finite number of registers for storing arbitrarily large nonnegative integers. The machine runs a program consisting of numbered instructions of several simple types. Several variants of the machine with different number of registers and different instruction sets were shown to be computationally universal. The basic instruction types we use here are:

a' add 1 to the content of the register a and continue with the next instruction

$a^-(k)$ if the content of the register a is nonzero, then subtract 1 from it and continue with the next instruction, else continue with the k-th instruction

H halt the machine

We can assume that H is used only as the last instruction of each program. Minsky proved in [7], sec. 11.2, that any Turing machine can be simulated by a register machine with the above instruction types and five registers.

It follows from the construction given by Minsky that each partial recursive function f can be computed by the register machine mentioned above, starting with the argument value n and ending with the value $f(n)$ in some register. Let us further assume that if the function $f(n)$ is undefined for some $n \geq 0$, then the corresponding register machine never halts with input n.

4 The Universality of Communicating P Systems

We show that any partially recursive function can be computed by a communicating P system. The result follows from the fact that any register machine computing a program P can be simulated by a communicating P system. Informally, each register r_i corresponds to a separate membrane labelled E_i. Each instruction i_j of P is simulated by a separate membrane labelled I_j, which contains two supplementary membranes labelled H_j and J_j. We label all the membranes (except the skin membrane) by the combination of a letter and a number for better understanding. The whole membrane structure is indicated in Fig. 1.

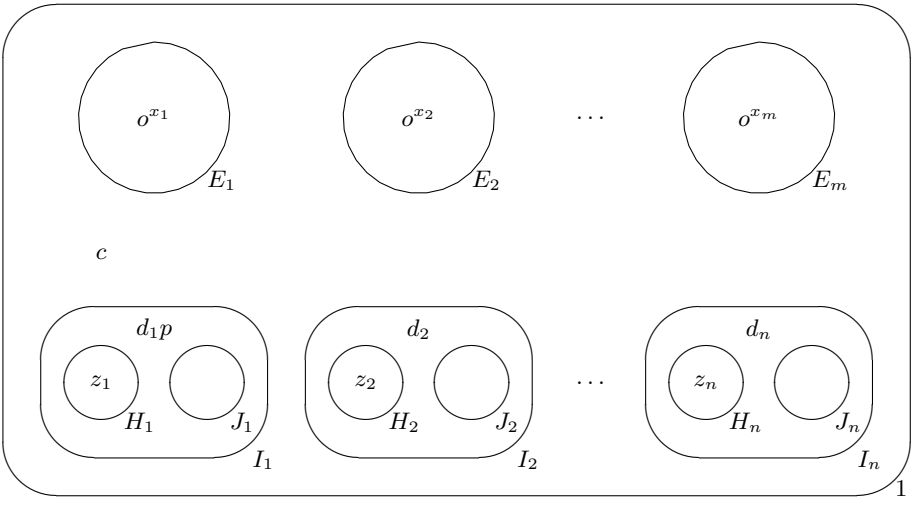


Fig. 1. The structure and objects of the P system simulating register machine.

There is a special object p within the P system, playing the role of the program counter. Its presence activates the membrane I_j to perform the computation corresponding to the instruction i_j. There are moreover objects labelled by

o, playing the roles of counter elements. The value of the register r_i corresponds at each moment to the number of o's in the membrane E_i.

Theorem 1. *An arbitrary recursively enumerable function $f : \mathbb{N} \longrightarrow \mathbb{N}$ can be computed by a communicating P system.*

Proof. Consider a register machine M with the registers $\mathbf{r}_1, \ldots, \mathbf{r}_m$, $m \geq 1$. Let P be the program for M with n instructions i_1, i_2, ..., i_n, computing f. Let P assume the input and the output value in register $\mathbf{r}_q$, $1 \leq q \leq m$.

We construct the communicating P system

$$\Pi = (V, \mu, w_1, \ldots, w_k, R_1, \ldots, R_k, E_q),$$

where

$$\begin{aligned} k &= 3n + m + 1; \\ V &= \{c, o, p\} \cup \{d_i, z_i \mid 1 \leq i \leq n\}; \\ \mu &= [_1 [_{E_1}]_{E_1} \ldots [_{E_m}]_{E_m} [_{I_1} [_{H_1}]_{H_1} [_{J_1}]_{J_1}]_{I_1} \ldots [_{I_n} [_{H_n}]_{H_n} [_{J_n}]_{J_n}]_{I_n}]_1; \end{aligned}$$

Initial contents of the compartments are given by

$$\begin{aligned} w_1 &= c, \\ w_{E_i} &= o^{x_i}, \quad 1 \leq i \leq m, \ x_i \text{ being the initial value in the register } r_i, \\ w_{I_1} &= d_1 p, \\ w_{I_i} &= d_i, \quad 2 \leq i \leq n, \\ w_{H_i} &= z_i, \quad 1 \leq i \leq n, \\ w_{J_i} &= \lambda, \quad 1 \leq i \leq n. \end{aligned}$$

The sets of rules of Π depend on the instructions of P and are constructed as follows.

For each j, $1 \leq j < n$, such that $i_j = a'$ for some a, $1 \leq a \leq m$, we define

$$\begin{aligned} R_{I_j} &= \{pd_j \to pd_{j;out}, \ pc \to p_{out}c, \ cd_j \to c_{out}d_j\}, \\ R_{H_j} &= \emptyset, \\ R_{J_j} &= \emptyset, \end{aligned}$$

and there is a set of rules in R_1 defined as

$$\{cd_j \to c_{in_{I_j}} d_j o_{come}, \ d_j o \to d_j o_{in_{E_a}}, \ d_j p \to d_{j;in_{I_j}} p_{in_{I_{j+1}}}\}.$$

For each j, $1 \leq j < n$, such that $i_j = a^-(k)$ for some a, k, $1 \leq a \leq m$, $1 \leq k \leq n$ (let's assume $j \neq k$, otherwise the program P would never stop), we define

$$\begin{aligned} R_{I_j} = \{&pd_j \to pd_{j;out}, \ pc \to p_{out}c_{in_{H_j}}, \ cz_j \to c_{in_{J_j}} z_{j;out}, d_j z_j \to d_{j;in_{J_j}} z_{j;in_{H_j}}, \\ &cd_j \to c_{out}d_j\}, \\ R_{H_j} = \{&cz_j \to c_{out}z_{j;out}\}, \\ R_{J_j} = \{&cd_j \to c_{out}d_j\}, \end{aligned}$$

and moreover R_1 contains the set of rules

$\{cd_j \to c_{in_{I_j}} d_{j;in_{E_a}},\ d_jp \to d_jp_{in_{I_{j+1}}},\ d_jo \to d_jo_{out},\ pz_j \to p_{in_{I_k}} z_{j;in_{E_a}},$
$d_jz_j \to d_{j;in_{I_j}} z_{j;in_{I_j}}\},$

and R_{E_a} contains the set of rules

$$\{d_jo \to d_{j;out}o_{out},\ d_jz_j \to d_{j;out}z_{j;out}\}.$$

The last instruction $i_n = H$ is represented by the membrane I_n containing no rules. At the beginning of simulation of an instruction i_j of M, $1 \leq j \leq n$, the "program counter" object p is located in the membrane I_j (initially $j = 1$). Then the system behaves as follows:

- If $i_j = a'$, $1 \leq a \leq m$, then the "messenger" object d_j is emitted to the region 1 by the rule $pd_j \to pd_{j;out}$. This allows transport of one "counter" object o from the surrounding space due to the rule $cd_j \to c_{in_{I_j}} d_jo_{come}$. The object o is immediately transported into the membrane E_a thanks to the rule $d_jo \to d_jo_{in_{E_a}}$. Meantime the object p (catalyzed by the "return messenger" c) is expelled from the membrane I_j by the rule $pc \to p_{out}c$ and transported to the membrane I_{j+1} via the rule $d_jp \to d_{j;in_{I_j}} p_{in_{I_{j+1}}}$. The messenger object d_j returns back to I_j.
- If $i_j = a^-(k)$, $1 \leq a \leq m$, $1 \leq k \leq n$, then again the object d_j goes to the region 1 and then immediately to the region E_a thanks to the rules $pd_j \to pd_{j;out}$ and $cd_j \to c_{in_{I_j}} d_{j;in_{E_a}}$.
  (i) If there is an object o in E_a, it is transported out of E_a together with d_j by the rule $d_jo \to d_{j;out}o_{out}$ and then expelled from the system to the outer space by the rule $d_jo \to d_jo_{out}$. The object p (which in the meantime returned back to the region 1) is transported to the membrane I_{j+1} via the rule $d_jp \to d_jp_{in_{I_{j+1}}}$.
  (ii) If there is no object o in E_a, the messenger d_j stays at E_a for several steps. Then the "zero messenger" z_j is emitted out from H_j to I_j and then to region 1. It catalyzes the transport of p into I_k and moves to E_a by the rule $pz_j \to p_{in_{I_k}} z_{j;in_{E_a}}$. Then both messengers d_j and z_j returns back to I_j thanks to the rule $d_jz_j \to d_{j;in_{I_j}} z_{j;in_{I_j}}$.

If the object p enters the halting region I_n (containing no rules), the system halts since no more rules are applicable. Then the number of o's in the output membrane E_q represents the result of the computation.

In this way the system Π simulates a sequence of instructions of the program P beginning by the first instruction i_1 and ending by reaching the instruction H. The system Π is deterministic and it follows from the above description that after performing each instruction the number of o's within the membranes $E_1, \ldots, E_m$ equals to the contents of registers $\mathbf{r}_1, \ldots, \mathbf{r}_m$. Hence after performing the instruction H the number of o's within the membrane E_q equals the output of the program P.

5 An Optimal Parallel Algorithm Finding Maximum

Communicating P systems allow simple and natural implementation of some parallel algorithms. Recall some basic notation of theory of parallel algorithms first (for more information we refer to [6]).

Let Q be a computational problem whose sequential time complexity is $T^*(n)$, i.e. there exists a sequential algorithm solving Q in $\mathcal{O}(T^*(n))$ steps and this limit cannot be improved. A parallel algorithm solving the problem Q in time $T(n)$ using $W(n)$ operations is called *optimal*, if $W(n) = \Theta(T^*(n))$, i.e. the total number of used operations is asymptotically equal to the sequential time complexity, regardless to the number of steps $T(n)$ of the parallel algorithm.

As an example we present a P system $\Pi_{\max}$ implementing an optimal parallel algorithm for finding maximum of n nonnegative integers in $\mathcal{O}(\log n)$ time. We consider the parallel application of one rule to an arbitrary number of objects in one of the regions of $\Pi_{\max}$ as one elementary operation of P system.

The sequential time complexity of finding the maximum is $T^*(n) = n$. Let's choose $n = 2^m$, $m \geq 1$ for simplicity. Our P system

$$\Pi_{\max} = (\{a, b, c, d\}, \mu, w_1, \ldots, w_k, R_1, \ldots, R_k, P_0)$$

contains $k = 5(n-1)$ membranes structured into $m + 2 = \log n + 2$ nested levels. The whole membrane structure consists of four groups of the membranes labelled by $0, \ldots, 2n-2$, $P_0, \ldots, P_{n-2}$, $Q_0, \ldots, Q_{n-2}$ and $F_1, \ldots, F_{n-2}$. The skin membrane labelled by 0 is considered as level one. Each membrane labelled by i, $0 \leq i < n-1$, contains the membranes $2i+1$, $2i+2$, P_i and Q_i. Further the membrane P_i, $1 \leq i < n-1$, contains the membrane F_i. All the other membranes contain no further nested membranes. The output membrane is P_0. An example of the structure for $n = 4$ is in Figure 2.

Let $(x_1, \ldots, x_n)$ be the input vector of n nonnegative integers. The membrane $n-2+i$, $1 \leq i \leq n$, contains the multiset of symbols $\{a^{x_i}\}$, if i is odd, and $\{b^{x_i}\}$ otherwise.

Further every membrane P_j at level ℓ, $3 \leq \ell \leq m+1$, where $2^{\ell-2} - 1 \leq j \leq 2^{\ell-1} - 2$, contains initially the multiset

- $\{c^y\}$, if $m - \ell$ is odd and j is odd,
- $\{d^y\}$, if $m - \ell$ is odd and j is even,
- $\{a^y\}$, if $m - \ell$ is even and j is odd,
- $\{b^y\}$, if $m - \ell$ is even and j is also even,

where y is a sufficiently large integer chosen so that $y \geq \max\{x_1, \ldots, x_n\}$. All the other membranes are initially empty.

The sets of rules in the membranes j and P_j, $1 \leq j < n-1$, at level $\ell = \lfloor log(j+1) \rfloor + 1$ are

(i) $R_j = \{ab \to a_{in_{P_j}} b_{in_{Q_j}}, a \to a_{in_{P_j}}, b \to b_{in_{P_j}}, c \to c_{out}\}$, $R_{P_j} = \{ac \to a_{in_{F_j}} c_{out}, bc \to b_{in_{F_j}} c_{out}\}$ for $m - \ell$ odd and j odd,

(ii) $R_j = \{ab \to a_{in_{P_j}} b_{in_{Q_j}}, a \to a_{in_{P_j}}, b \to b_{in_{P_j}}, d \to d_{out}\}$, $R_{P_j} = \{ad \to a_{in_{F_j}} d_{out}, bd \to b_{in_{F_j}} d_{out}\}$ for $m - \ell$ odd and j even,

(iii) $R_j = \{cd \to c_{in_{P_j}} d_{in_{Q_j}}, c \to c_{in_{P_j}}, d \to d_{in_{P_j}}, a \to a_{out}\}$, $R_{P_j} = \{ca \to c_{in_{F_j}} a_{out}, da \to d_{in_{F_j}} a_{out}\}$ for $m - \ell$ even and j odd,

(iv) $R_j = \{cd \to c_{in_{P_j}} d_{in_{Q_j}}, c \to c_{in_{P_j}}, d \to d_{in_{P_j}}, b \to b_{out}\}$, $R_{P_j} = \{cb \to c_{in_{F_j}} b_{out}, db \to d_{in_{F_j}} b_{out}\}$ for $m - \ell$ even and j even otherwise.

The membrane 0 contains only the rules $R_0 = \{ab \to a_{in_{P_0}} b_{in_{Q_0}}, a \to a_{in_{P_0}}, b \to b_{in_{P_0}}\}$ for m even, respectively $R_0 = \{cd \to c_{in_{P_0}} d_{in_{Q_0}}, c \to c_{in_{P_0}}, d \to d_{in_{P_0}}\}$ for m odd. The remaining membranes labelled by j, $n-1 \le j \le 2n-2$, contain the sets of rules

$$R_j = \begin{cases} \{a \to a_{out}\} & \text{if } j \text{ is odd,} \\ \{b \to b_{out}\} & \text{otherwise.} \end{cases}$$

All the other membranes are rule-free. Considering Theorem 1, our membrane system could be realized without a priority relation, but the use of priorities (in a very simple way) simplifies the system significantly. Let's define the priority of rules so that all the rules with two symbols at their left-hand side have a priority over the rules with a single symbol at their left-hand side. Moreover, unlike in some models of P systems with priorities, the rules with different priorities can be used simultaneously. Hence, given e.g. the rules $(ab \to a_o ut b_{out}) > (b \to b_{in_2})$ applied to the multiset of objects $m(a^i b^{i+j})$, then i pairs of objects a, b will be transported $out$and j objects b will be transported into the membrane 2 during the same step.

The computation of the system $\Pi_{\max}$ is the following. Consider the membranes $n-1$, resp. n, containing initially the multisets a^{x_1}, resp. b^{x_2}. All these symbols are expelled into the surrounding membrane $j = n/2 - 1$ in the first step. Then

- the rule $ab \to a_{in_{P_j}} b_{in_{Q_j}}$ is applied to $\min\{x_1, x_2\}$ couples (a, b),
- the rule $a \to a_{in_{P_j}}$ is applied to $x_1 - \min\{x_1, x_2\}$ symbols a,
- the rule $b \to b_{in_{P_j}}$ is applied to $x_2 - \min\{x_1, x_2\}$ symbols b.

Hence totally $\max\{x_1, x_2\}$ symbols enters the membrane P_j in the second step. Then due to the rules $ac \to a_{in_{F_j}} c_{out}$, $bc \to b_{in_{F_j}} c_{out}$ the same number of symbols c is transported out of P_j to the membrane j in the third step.

The similar process is in progress simultaneously in all couples of the membranes $(n-3+2i, n-2+2i)$, $1 \le i \le n/2$, at level $m+1$, the usage of symbols c and d being alternated.

Now the whole process is repeated for the membranes at level m but the roles of the symbols a, b and c, d are mutually exchanged. This process, which can be described by a binary tree, continues through levels $m+1$ to 3. Finally at level 2 the process takes two final steps (the third step is omitted due to non-existence of rules at the membrane P_0) and the resulting multiset of symbols corresponding to the value $\max\{x_1, \ldots, x_n\}$ is left within the output membrane P_0. At this moment the computation stops because no rules can be further applied.

Theorem 2. *The membrane system $\Pi_{\max}$ computes the maximum of $(x_1, \ldots, x_n)$ in time $T(n) = \mathcal{O}(\log n)$ using the optimal number of operations $W(n) = \Theta(n)$.*

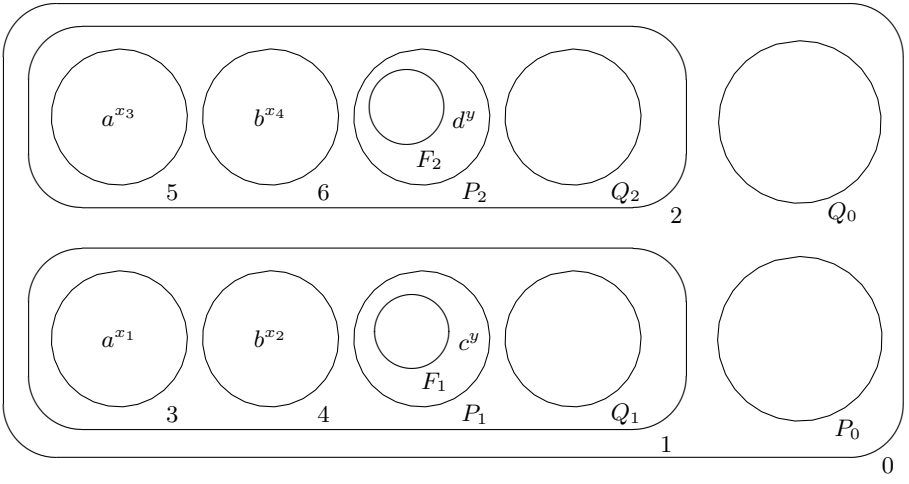


Fig. 2. The membrane structure of the P system $\Pi_{\max}$ computing maximum of four integers.

Proof. It follows from the above description that the necessary number of parallel time steps is

$$T(n) = 3(m-1) + 2 = 3\log n - 1 = \mathcal{O}(\log n).$$

Consider the number of operations W_ℓ necessary in progress of computation from level $\ell+1$ to ℓ, $m \geq \ell \geq 2$. There is $2^{\ell-1}$ membranes labelled by a number at level ℓ, and for each of them 4 or 5 operations are performed during the three time steps described above. (The value 4 takes place when neither of the rules $a \to a_{in_{P_j}}$, $b \to b_{in_{P_j}}$ can be used.) Hence

$$4 \cdot 2^{\ell-1} \leq W_\ell \leq 5 \cdot 2^{\ell-1}.$$

Finally we need $3 \leq W_1 \leq 4$ operations to proceed from level 2 to 1. Taking into the account the sum

$$W(n) = W_1 + \sum_{\ell=2}^{m} W_\ell,$$

we can conclude that

$$4(n-2) + 3 \leq W(n) \leq 5(n-2) + 4,$$

and hence $W(n) = \Theta(n)$.

6 Discussion

As already mentioned above, a similar approach studying pure communication in P systems is independently presented in [8]. The authors show that each

recursively enumerable set of numbers can be generated by a communicating P system with similar restrictions as here. Their model, however, is generative and nondeterministic, using the techniques usual in [2], while the P-system used in the proof of Theorem 1 is deterministic. It means that at each step there is max. one possible target configuration of the system.

The price to pay for this determinism was the necessity to use addressed transport of the molecules, the expressions of the form a_{in_j} on the right-hand side of the rules. In the model presented in [8] the weaker form a_{in} is used, allowing to transport the object a to any of the adjacent membranes chosen nondeterministically.

The need for addressed transport of objects can be highly reduced if we did not restrict ourselves to the membrane structures forming a rooted tree, but applied more general structures as asymmetric graphs mentioned in [1].

We also conjecture that the repertoire of the operations used in our model can be further reduced without a loss of computational power. To make the model more biologically plausible, we could e.g. restrict possible combinations of targets on the right-hand side of rules (discussed also in [8]).

Another difference dwells in the fact that unlike [8], the universality proof of Theorem 1 uses an unbounded number of membranes. Whether these systems therefore establish an infinite hierarchy with respect to the number of membranes remains as an *open question* for future research.

Acknowledgements. Authors are grateful to Gheorghe Păun and Rudi Freund for useful comments to some parts of the paper. The research of Petr Sosík was supported by the Grant Agency of Czech Republic, grant No. 201/02/P079.

References

1. C. S. Calude, Gh. Păun, *Computing with Cells and Atoms* (Taylor & Francis, London, 2001).
2. J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory* (Springer, Berlin, 1989).
3. J. Dassow and Gh. Păun, On the power of membrane computing, *Journal of Universal Computer Science* **5**, 2 (1999), pp. 33–49 (http://www.iicm.edu/jucs).
4. R. Freund, Generalized P systems, *Fundamentals of Computation Theory, FCT'99,* Iasi, 1999, (G. Ciobanu, Gh. Păun, eds.), *LNCS* 1684, Springer-Verlag (1999), pp. 281–292.
5. T. Head, Aqueous Simulations of Membrane Computations, *Romanian J. of Information Science and Technology* **4** (2001), pp. 1–2.
6. J. Jájá, *An introduction to parallel algorithms* (Addison-Wesley, Reading, 1992).
7. M. L. Minsky, *Finite and Infinite Machines* (Prentice Hall, Englewood Cliffs, New Jersey, 1967).
8. A. Paun, Gh. Paun, The Power of Communication: P Systems with Symport/Antiport, *New Generation Computers,* to appear.
9. Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences* **61** 1 (2000), pp. 108–143, and TUCS Research Report 208, 1998 (`http://www.tucs.fi`).

10. Gh. Păun, Computing with Membranes: An Introduction, *Bulletin EATCS* **67** (1999), pp. 139-152.
11. Gh. Păun, *Membrane Computing: an Introduction.* Springer-Verlag, Berlin, 2002 (to appear).
12. A. Salomaa, G. Rozenberg (eds.): *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.

Some New Generalized Synchronization Algorithms and Their Implementations for Large Scale Cellular Automata

Hiroshi Umeo, Masaya Hisaoka, Koshi Michisaka,
Koji Nishioka, and Masashi Maeda

Osaka Electro-Communication University,
Neyagawa-shi, Hatsu-cho 18-8, Osaka, 572-8530, Japan
umeo@umeolab.osakac.ac.jp

Abstract. In this paper, we study a generalized synchronization problem for large scale cellular automata (CA) on one- and two- dimensional arrays. Some new generalized synchronization algorithms will be designed both on O(1)-bit and 1-bit inter-cell communication models of cellular automata. We give a 9-state and 13-state CA that can solve the generalized synchronization problem in optimum- and linear-time on O(1)-bit 1-D and 2-D CA, respectively. The number of internal states of the CA implemented is the smallest one known at present. In addition, it is shown that there exists a 1-bit inter-cell communication CA that can synchronize 1-D n cells with the general on the kth cell in $n+\max(k, n-k+1)$ steps, which is two steps larger than the optimum time. We show that there still exist several new generalized synchronization algorithms, although more than 40 years have passed since the development of the problem.

1 Introduction

In recent years cellular automata (CA) have been establishing increasing interests in the study of modeling real phenomena occurring in biology, chemistry, ecology, economy, geology, mechanical engineering, medicine, physics, sociology, public traffic, etc. Cellular automata are considered to be a nice model of complex systems in which an infinite one-dimensional array of finite state machines (cells) updates itself in synchronous manner according to a uniform local rule.

In this paper, we study a famous firing squad synchronization problem which gives a finite-state protocol for synchronizing a large scale of cellular automata. The synchronization for cellular automata has been known as the firing squad synchronization problem since its development, where it was originally proposed by J. Myhill to synchronize all parts of a self-reproducing cellular automata [11]. The firing squad synchronization problem has been studied extensively in more than 40 years [1-21]. We study the firing squad synchronization problem in a more generalized case, where the initial general is located at any position on the array. Several new generalized synchronization algorithms will be designed

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 276–286, 2002.

both on O(1)-bit and 1-bit inter-cell communication models of cellular automata. An O(1)-bit model is a conventional CA where the amount of communication bits exchanged at one step between neighboring cells is assumed to be O(1)-bit, however, such bit-information exchanged between inter-cells has been hidden behind the definition of conventional automata-theoretic finite state descriptions. On the other hand, a 1-bit inter-cell communication model is a new CA whose inter-cell communication is restricted to 1-bit. We call the model 1-bit CA in short. The number of internal states of the 1-bit CA is assumed to be finite in a usual way. The next state of each cell is determined by the present state of itself and two binary 1-bit inputs from its left and right neighbor cells. Thus the 1-bit CA can be thought to be one of the most powerless and simplest models in a variety of CAs.

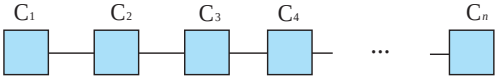


Fig. 1. One-dimensional cellular automaton.

In the next section 2, the generalized firing squad synchronization problem is introduced. Moore and Langdon [12], Szwerinski [15] and Varshavsky, Marakhovsky and Peschansky [20] developed a generalized optimum-time firing algorithm with 17, 10 and 10 internal states, respectively, that fires 1-D n cells in $n-2+\max(k, n-k+1)$ steps, where the general is located on C_k. We propose a new 9-state generalized synchronization algorithm operating in optimum-step in section 3. In addition an $n+\max(k, n-k+1)$-step algorithm is also given on CA_{1-bit}, which is a generalized extension of Mazoyer [8], Nishimura, Sogabe and Umeo [13] and Umeo, Nishimura and Sogabe [16]. In section 4, a 13-state implementation for synchronizing two-dimensional arrays will be given. Szwerinski [15] proposed an optimum-time generalized 2-D firing algorithm with 25600 internal states that fires any $m \times n$ array in $m+n+\max(m,n)-\min(r, m-r+1)-\min(s, n-s+1)-1$ steps. Our 2-D generalized synchronization algorithm is $\max(r+s, m+n-r-s+2)-\max(m,n)+\min(r, m-r+1)+\min(s, n-s+1)-3$ steps larger than the optimum one proposed by Szwerinski [15], however, the number of internal states required for the firing is considerably decreased and it is the smallest one known at present. Due to the space available in this paper, we omit the details of the proofs of theorems given below.

2 Generalized Firing Squad Synchronization Problem

The generalized firing squad synchronization problem is formalized in terms of the model of cellular automata. Fig. 1 shows a finite one-dimensional (1-D) cellular array consisting of n cells. Each cell is an identical (except the end cells)

finite state automaton. The array operates in lock-step mode in such a way that the next state of each cell (except both end cells) is determined by both its own present state and the present states of its right and left neighbors. Let k be any integer such that $1 \leq k \leq n$. All cells (*soldiers*), except the kth cell C_k from the left end, 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 kth cell C_k (*general*) is in *fire-when-ready* state that is an initiation signal to the array. The generalized firing squad synchronization problem [12, 15, 20] is stated as follows: Given an array of n identical cellular automata, including a *general* on the kth cell which is activated at time $t = 0$, we want to give the description (state set and next-state function) of the automata so that, *at some future time*, all the cells will *simultaneously* and, *for the first time*, enter a special *firing* state. The set of states must be independent of n. The tricky part of the problem is that the same kind of soldier with a fixed number of states is required to synchronize, regardless of the length n of the array.

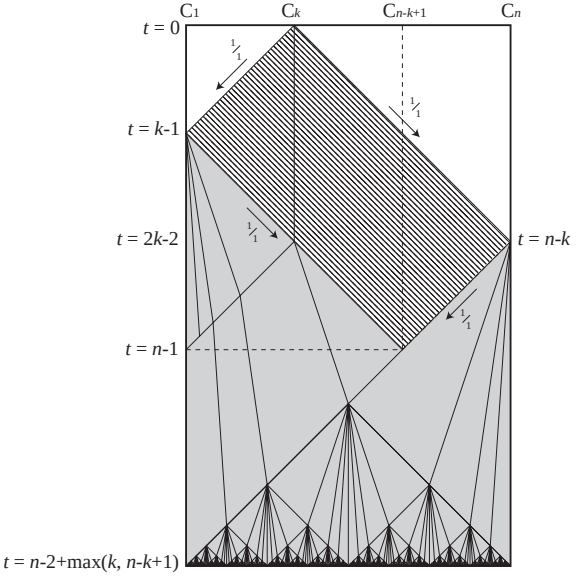


Fig. 2. Time-space diagram for optimum-time generalized firing squad synchronization algorithm.

3 Generalized Synchronization Algorithms on 1-D Arrays

In this section we propose two generalized synchronization algorithms both on O(1)-bit and 1-bit communication models of 1-D cellular automata.

3.1 Generalized Synchronization Algorithm on O(1)-Bit Communication Model

A generalized firing squad synchronization problem on 1-D O(1)-bit-communication CA has been studied by several researchers [12, 15, 20]. Moore and Langdon [12], Szwerinski [15] and Varshavsky, Marakhovsky and Peschansky [20] developed an optimum-time firing algorithm with 17, 10 and 10 internal states, respectively, that fires n cells in $n-2+\max(k, n-k+1)$ steps, where the general is located on C_k. Fig. 2 is a time-space diagram for optimum-time generalized firing scheme. We develop a 9-state generalized synchronization algorithm. In Fig. 3 we show snapshots of our 9-state firing synchronization algorithm.

[Theorem 1] There exists a 9-state one-dimensional CA which can synchronize n cells in exactly optimum $n-2+\max(k, n-k+1)$ steps, where the general is located on C_k.

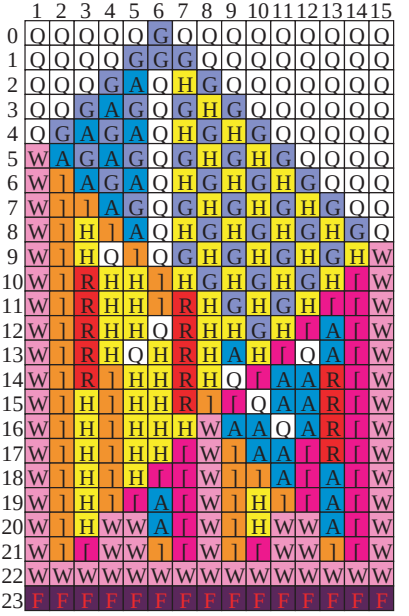


Fig. 3. Snapshots of our 9-state optimum-time generalized firing squad synchronization algorithm.

3.2 Generalized Synchronization Algorithm on 1-Bit Communication Model

In the long history of the study of cellular automata, the amounts of bit-information exchanged at one step between neighboring cells have been assumed

to be O(1)-bit. Here we introduce a new class of cellular automata whose inter-cell communication is restricted to 1-bit. A cellular automaton with 1-bit inter-cell communication (abbreviated by CA_{1-bit}) consists of a one-dimensional array of finite state automaton $A = (Q, \delta)$, where

1. Q is a finite set of internal states.
2. δ is a function, defining 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 receiving binary inputs x and y from its left and right communication links, respectively. Then, at the next step $t+1$, C_i assumes state q and outputs x' and y' to its left and right communication links, respectively. Note that 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 $\delta(q, 0, 0) = (q, 0, 0)$.

The generalized firing squad synchronization problem on CA_{1-bit} can be defined similarly. What is difficult in designing synchronization algorithms on CA_{1-bit} is a strong restriction to the amounts of bit information exchanged between neighbor cells at one step. In our construction additional two steps are required for transmitting a signal generated on the general cell to the nearest end. We illustrate a time-space diagram in Fig. 4(a), where a two-step delayed area is shaded. In addition, we show snapshots of our generalized firing synchronization algorithm for 24 cells with a general on C_8 in Fig. 4(b). Small right and left black triangles ▶ and ◀, shown in the figure, indicate a 1-bit signal transfer in the right or left direction between neighbor cells. A symbol in a cell shows its internal state. The total number of internal states and transition rules of the CA realized on a computer is 282 and 721, respectively. We checked the rule set from $n = 2$ to 100 at any position of the general.

[**Theorem 2**] There exists a CA_{1-bit} which can synchronize n cells in exactly $n+\max(k, n-k+1)$ steps.

4 Generalized Synchronization Algorithms on 2-D Arrays

In this section we consider the generalized synchronization problem on 2-D O(1)-bit communication CA models. Fig. 5 shows a finite two-dimensional cellular array consisting of $m \times n$ cells. A cell on (i, j) is denoted by $\mathrm{C}_{i,j}$. 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 north, south, east and west neighbors. Let r, s be any integer such that $1 \leq r \leq m$, $1 \leq s \leq n$. At time $t = 0$ the general cell $\mathrm{C}_{r,s}$ is in *fire-when-ready* state that is an initiation signal to the array. All cells, except the general cell, are initially in the quiescent state with the property that the next state of a quiescent cell with quiescent neighbors is the quiescent state again. Several 2-D synchronization

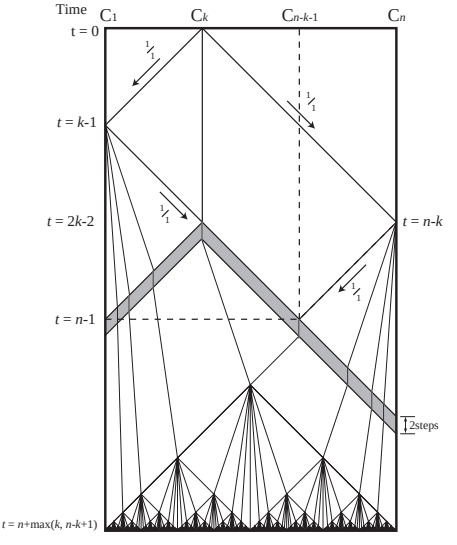


Fig. 4(a)

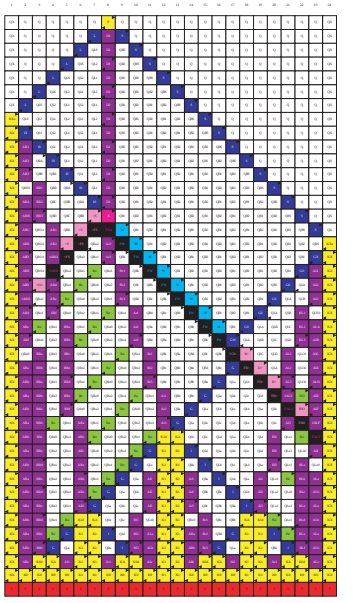


Fig. 4(b)

Fig. 4. Time-space diagram of generalized firing squad synchronization algorithm on CA_{1-bit} (Fig. 4(a)) and snapshots of the computation for 24 cells with a general on C_8 (Fig. 4(b)).

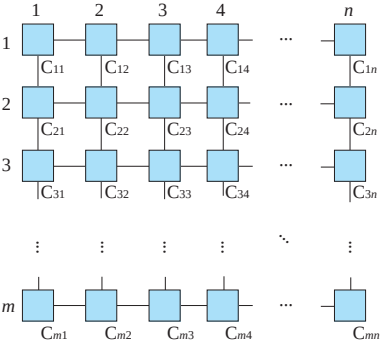


Fig. 5. Two-dimensional cellular automaton.

algorithms and their implementations have been presented in Beyer [2], Grasselli [3], Maeda and Umeo [6], Shinar [14] and Szwerinski [15].

Before presenting the algorithm, we propose a simple and efficient mapping scheme for embedding one-dimensional synchronization algorithms onto two-dimensional arrays. Fig. 6(a) is a time-space diagram for optimum-time generalized synchronization on one-dimensional arrays with the general on C_k. We say that a generalized firing algorithm has a *property A*, where any cell, except the general C_k, keeps a quiescent state *in the area A* of the time-space diagram shown in Fig. 6(a). The one-dimensional generalized firing squad synchronization algorithm with the property A can be easily embedded onto two-dimensional arrays with a small overhead. Fig. 6(b) shows snapshots of our 12-state optimum-time generalized firing squad synchronization algorithm with the property A.

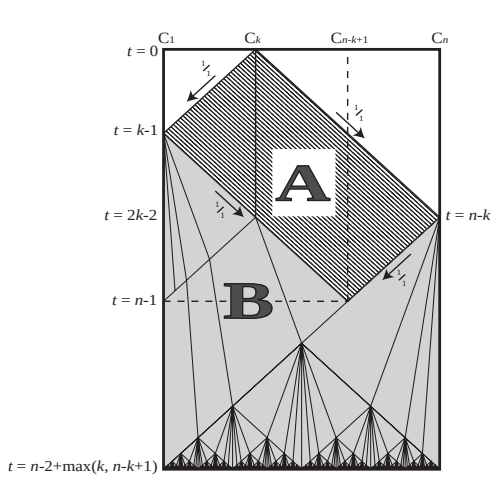


Fig. 6(a)

Fig. 6(b)

Fig. 6. Time-space diagram for optimum-time generalized firing squad synchronization algorithm (Fig. 6(a)) and snapshots of our 12-state optimum-time generalized firing squad synchronization algorithm with the *property A* (Fig. 6(b)).

[**Theorem 3**] There exists a 12-state one-dimensional CA with the property A which can synchronize n cells in exactly optimum $n - 2+\max(k, n-k+1)$ steps, where the general is located on C_k.

Now we consider a 2-D array of size $m \times n$. We divide mn cells into $m+n-1$ groups g_k, $1 \leq k \leq m+n-1$, defined as follows;

$$g_k = \{C_{i,j} | (i-1) + (j-1) = k-1\}.$$

That is, $g_1 = \{C_{1,1}\}$, $g_2 = \{C_{1,2}, C_{2,1}\}$, $g_3 = \{C_{1,3}, C_{2,2}, C_{3,1}\}$, . . . , $g_{m+n-1} = \{C_{m,n}\}$.

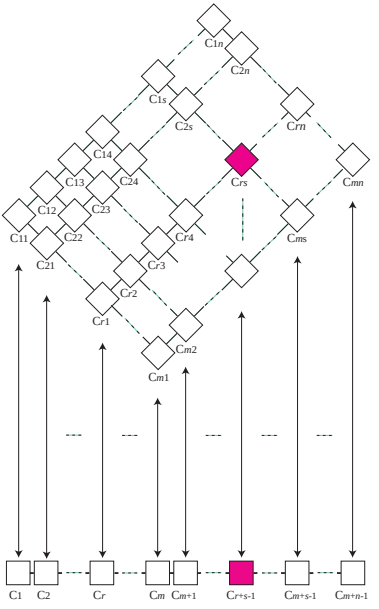


Fig. 7. A correspondence between 1-D and 2-D cellular arrays.

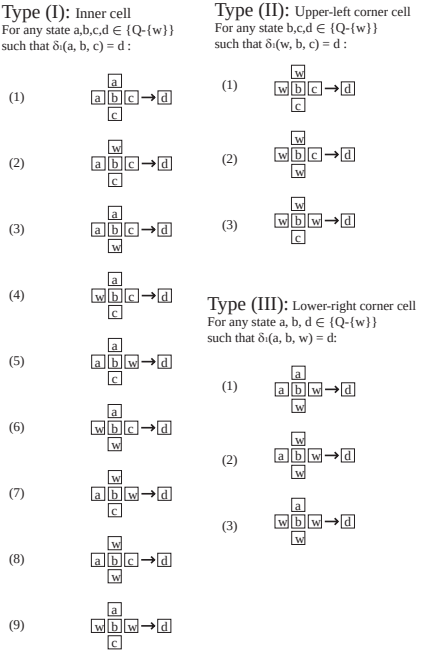


Fig. 8. Construction of transition rules for generalized firing squad synchronization algorithm on 2-D arrays.

Let $M = (Q, \delta_1, w)$ be any one-dimensional CA with the property A that fires ℓ cells being the general on C_k in $T(\ell, k)$ steps, where Q is the finite state set of M, $\delta_1 : Q^3 \to Q$ is the transition function, and $w \in Q$ is the right and left end state. We assume that M has $m + n - 1$ cells. We consider the one-to-one correspondence between the ith group g_i and the ith cell C_i on M such that $g_i \leftrightarrow C_i$, where $1 \le i \le m + n - 1$. See Fig. 7. We can construct a 2-D CA $N = (Q, \delta_2, w)$ so that all cells in g_i simulates the ith cell C_i in real-time and N can fire any $m \times n$ arrays with the general $C_{r,s}$ at time $t = T(m+n-1, r+s-1)$ if and only if M fires 1-D arrays of length $m+n-1$ with the general on C_{r+s-1} at time $t = T(m+n-1, r+s-1)$, where $\delta_2 : Q^5 \to Q$ is the transition function, and $w \in Q$ is the border state of the array.

The transition function δ_2 is constructed as follows: Let $\delta_1(a, b, c) = d$ be any transition rule of M, employed in the area B shown in Fig. 6(b), where $a, b, c, d \in \{Q - \{w\}\}$. Then, N has nine transition rules, as is shown in Type (I) of Fig. 8. The first rule(1) in Type (I) is used by an inner cell that doesn't have any border cells in its four neighbours. The rules (2)-(7) are used by an inner cell that has border cells in its upper, lower, left, right, left lower, and right upper direction, respectively. The rules (8) and (9) correspond to the case where $m = 1$, $n \ge 2$ and $m \ge 2$, $n = 1$, respectively. When $a = w$, that is, $\delta_1(w, b, c) = d$,

where $b, c, d \in \{Q - \{w\}\}$, then N has three rules shown in Type (II). These rules are used by the cell located in left upper corner. The rules (2) and (3) in Type (II) are special cases corresponding to the rules (8) and (9) in Type (I). When $c = w$, that is, $\delta_1(a, b, w) = d$, where $a, b, d \in \{Q - \{w\}\}$, then N has three rules shown in Type (III). These rules are used by the cell located in right lower corner. The rules (2) and (3) in Type (III) are special cases corresponding to the rules (8) and (9) in Type (I).

We show that the 2-D CA N constructed can generate the configuration of M in real-time. Precisely, for any i, $1 \leq i \leq m+n-1$, the state of any cell in g_i at any step is the same each other in g_i and it is exactly the state of C_i at the corresponding step. Let S_i^t, $S_{i,j}^t$ and $S_{g_i}^t$ denote the state of C_i, $C_{i,j}$ at step t and the set of states of the cells in g_i at step t, respectively. Then, we can establish the following lemma.

[**Lemma 4**] The next two statements hold:

1. For any integer i and t such that $1 \leq i \leq m+n-r-s+1$, $r+s+i-3 \leq t \leq T(m+n-1, r+s-1)$, $\parallel S_{g_i}^t \parallel = 1$ and $S_{g_i}^t = S_i^t$. That is, all cells in g_i at step t are in the same state and it is equal to S_i^t, where the state in $S_{g_i}^t$ is simply denoted by $S_{g_i}^t$.
2. For any integer i and t such that $m+n-r-s+2 \leq i \leq m+n-1$, $2m+2n-r-s-i-1 \leq t \leq T(m+n-1, r+s-1)$, $\parallel S_{g_i}^t \parallel = 1$ and $S_{g_i}^t = S_i^t$.

For any 2-D array of M of size $m \times n$ with the general at $C_{r,s}$, $1 \leq r \leq m$, $1 \leq s \leq n$, there exists a corresponding 1-D cellular array N of length $m+n-1$ with the general at C_{r+s-1} such that the configuration of N can be mapped on M and M fires if and only if N fires. Based on the 12-state generalized 1-D algorithm given in [Theorem 3] we get the following 2-D generalized synchronization algorithm that fires in $m+n-1-2+\max(r+s-1, m+n-r-s+1) = m+n+\max(r+s, m+n-r-s+2)-4$ steps. One additional state is required in our construction. The details are omitted. Szwerinski [15] proposed an optimum-time generalized 2-D firing algorithm with 25600 internal states that fires any $m \times n$ array in $m+n+\max(m,n)-\min(r, m-r+1)-\min(s, n-s+1)-1$ steps. Our 2-D generalized synchronization algorithm is $\max(r+s, m+n-r-s+2)-\max(m,n)+\min(r, m-r+1)+\min(s, n-s+1)-3$ steps larger than the optimum one proposed by Szwerinski [15], however, the number of internal states required for the firing is the smallest known at present. In Fig. 9 we show snapshots of the 13-state generalized synchronization algorithm running on rectangular array of size 6×8 with the general on $C_{2,3}$.

[**Theorem 5**] There exists a 13-state 2-D CA that can synchronize any $m \times n$ rectangular arrays in $m+n+\max(r+s, m+n-r-s+2)-4$ steps, where (r, s) is an arbitrary initial position of the general.

5 Summary

We have proposed several new generalized synchronization algorithms for one- and two-dimensional cellular arrays and implemented them on a computer. The

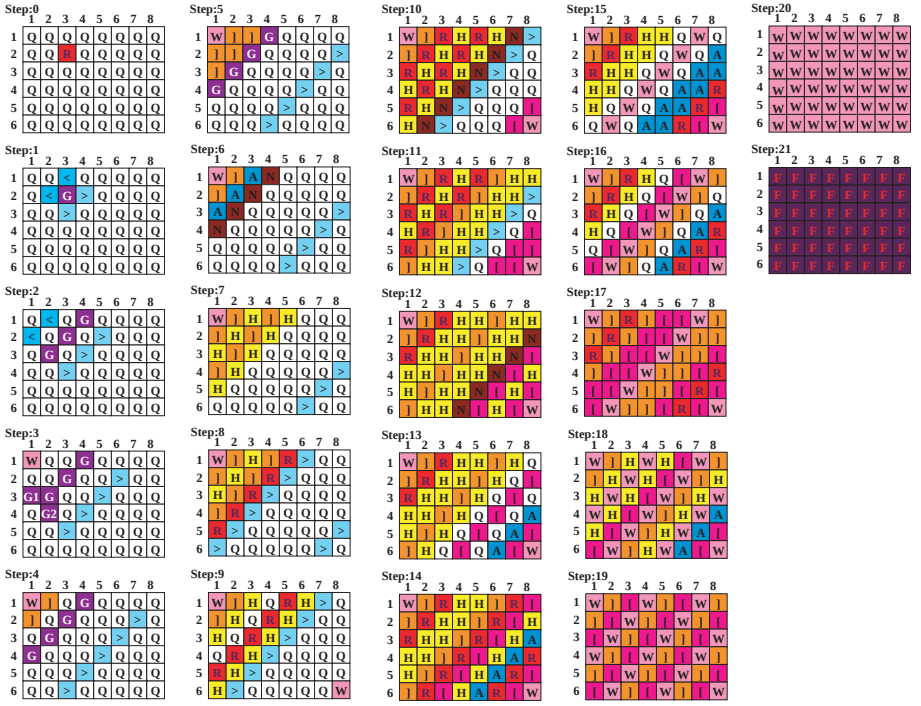


Fig. 9. Snapshots of the 13-state linear-time generalized firing squad synchronization algorithm on rectangular arrays.

number of internal states required for realizing those synchronization algorithms are considerably decreased and they are the smallest ones known at present.

References

1. R. Balzer: An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, vol. 10(1967), pp. 22-42.
2. W. T. Beyer: Recognition of topological invariants by iterative arrays. Ph.D. Thesis, MIT, (1969), pp. 144.
3. A. Grasselli: Synchronization of cellular arrays: The firing squad problem in two dimensions. *Information and Control*, vol. 28(1975), pp. 113-124.
4. Hans-D., Gerken: Über Synchronisations-Probleme bei Zellularautomaten. Diplomarbeit, Institut für Theoretische Informatik Technische Universität Braunschweig, pp.50, (1987).
5. E. Goto: A minimal time solution of the firing squad problem. Dittoed course notes for Applied Mathematics 298, Harvard University, (1962), pp. 52-59.
6. M. Maeda and H. Umeo: A design of two-dimensional firing squad synchronization algorithms and their implementations. *Proc. of The 15th Annual Conference of Japanese Society for Artificial Intelligence*, 2C3-05(2001), pp.1-4.

7. J. Mazoyer: A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, vol. 50(1987), pp. 183-238.
8. J. Mazoyer: On optimal solutions to the firing squad synchronization problem. *Theoretical Computer Science*, vol. 168(1996), pp. 367-404.
9. K. Michisaka, H. Yahara, N. Kamikawa and H. Umeo: A generalization of 1-bit-communication firing squad synchronization algorithm. *Proc. of The 15th Annual Conference of Japanese Society for Artificial Intelligence*, 2C3-06(2001), pp.1-4.
10. M. Minsky: *Computation: Finite and infinite machines*. Prentice Hall, (1967), pp. 28-29.
11. E. F. Moore: The firing squad synchronization problem. in *Sequential Machines, Selected Papers* (E. F. Moore ed.), Addison-Wesley, Reading MA., (1964), pp. 213-214.
12. F. R. Moore and G. G. Langdon: A generalized firing squad problem. *Information and Control*, vol. 12(1968), pp. 212-220.
13. J. Nishimura, T. Sogabe and H. Umeo: A Design of Optimum-Time Firing Squad Synchronization Algorithm on 1-Bit Cellular Automaton. Tech. Rep. of IPSJ, (2000).
14. I. Shinahr: Two- and three-dimensional firing squad synchronization problems. *Information and Control*, vol. 24(1974), pp. 163-180.
15. H. Szwerinski: Time-optimum solution of the firing-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position. *Theoretical Computer Science*, vol. 19(1982), pp. 305-320.
16. H. Umeo, J. Nishimura and T. Sogabe: 1-Bit Inter-cell Communication Cellular Algorithms(*invited lecture*). *Proc. of the Tenth Intern. Colloquium on Differential Equations*, held in Plovdiv in 1999, *International Journal of Differential Equations and Applications*, vol. 1A, no.4(2000), pp. 433-446.
17. H. Umeo, T. Sogabe and Y. Nomura: Correction, optimization and verification of transition rule set for Waksman's firing squad synchronization algorithms. *Proc. of the 4th International Conference on Cellular Automata for Research and Industry*, 2000, pp. 152-160.
18. H. Umeo, M. Maeda and N. Fujiwara: An efficient mapping scheme for embedding any one-dimensional firing squad synchronization algorithm onto two-dimensional arrays. *Proc. of the 5th International Conference on Cellular Automata for Research and Industry*, 2002 (*to appear*).
19. V. I. Varshavsky: Synchronization of a collection of automata with random pairwise interaction. *Autom. and Remote Control*, vol. 29(1969), pp. 224-228.
20. V. I. Varshavsky, V. B. Marakhovsky and V. A. Peschansky: Synchronization of interacting automata. *Mathematical Systems Theory*, vol. 4, no. 3(1970), pp. 212-230.
21. A. Waksman: An optimum solution to the firing squad synchronization problem. *Information and Control*, vol. 9(1966), pp. 66-78.

Relativistic Computers and Non-uniform Complexity Theory*

Jiří Wiedermann[1] and Jan van Leeuwen[2]

[1] Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz
[2] Institute of Information and Computing Sciences, Utrecht University,
Padualaan 14, 3584 CH Utrecht, The Netherlands
j.vanleeuwen@cs.uu.nl

Abstract. Recent research in theoretical physics on 'Malament-Hogarth space-times' indicates that so-called relativistic computers can be conceived that can carry out certain classically undecidable queries in finite time. We observe that the relativistic Turing machines which model these computations recognize precisely the Δ_2-sets of the Arithmetical Hierarchy. In a complexity-theoretic analysis, we show that the (infinite) computations of $S(n)$-space bounded relativistic Turing machines are equivalent to (finite) computations of Turing machines that use a $S(n)$-bounded advice f, where f itself is computable by a $S(n)$-space bounded relativistic Turing machine. This bounds the power of polynomial-space bounded relativistic Turing machines by $\mathsf{TM}/poly$. We also show that $S(n)$-space bounded relativistic Turing machines can be limited to one or two relativistic phases of computing.

What computers can or cannot compute is determined by the laws of physics alone, and not by pure mathematics.
in: D. Deutsch, The Fabric of Reality, 1998.

1 Introduction

A number of phenomena in science have lead to an increased interest in models of computation that are more powerful than classical Turing machines. The phenomena range from evolving systems in biology to non-classical time-models in (theoretical) physics. Whereas these contexts lead to many new questions, it is of interest to study the computational theory of these phenomena in relation to classical complexity theory and its ramifications. In this paper we explore the computational limits and features of the so-called 'relativistic computers' that were recently proposed by Etesi and Németi [7], a kind of computers based on considerations from general relativity theory.

* This research was partially supported by GA ČR grant No. 201/02/1456 and by EC Contract IST-1999-14186 (Project ALCOM-FT).

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 287–299, 2002.

Relativistic computers will be seen to belong to the category of machine models whose powers transcend those of classical Turing machines and thus go beyond the range of the Church-Turing thesis. Models of computation that do not obey the Church-Turing thesis have been investigated since Turing's times (cf. [14]). The existence of such models in the realm of mathematics is already sufficient reason for their investigation. For example, Turing machines with oracles [16] and Turing machines with a restricted type of oracles known as *advice* [12] are at the heart of relativised and non-uniform computational complexity theory, respectively (cf. [1,9]).

In newer studies, there is an increasing number of models that are justified by some physical reality in which they exist or could exist. Among the recent examples are the *evolving interactive systems*, which model the adaptive and potentially infinite computing behavior of Internet-like networks and of other 'living systems', real or artificial [20]. The model has been shown to possess 'super-Turing' computational power (cf. [17], [18]), and illustrates that realistic instances of computational systems can be identified that have this property. Further examples and related studies appear in e.g. [3,4,19].

Independently of this, the quest for computing beyond Turing machine limits has gained interest in theoretical physics. Namely, it has been shown that certain *relativistic space-times theories* license the idea of observing the infinity of certain discrete processes in finite physical time [7]. It is speculated that a similar phenomenon could occur in the realm of quantum computing as well [5]. From the viewpoint of computability theory the resulting *relativistic computers* can be seen as Turing machines with a suitable oracle, but it is the fact that these computers are based on apparently realistic 'thought experiments' and also the nature of these experiments that gives them a special status. The consequences of this observation for the foundations of computing have begun to occupy philosophers and physicists ([7,10]), and recently mathematicians and logicians have begun to investigate even more general models of infinite-time computations [8].

In this paper we aim to appraise the power of *infinite computations* as realized by *relativistic computing* from the viewpoint of relativised and non-uniform computing. In this sense this paper complements the study in [7] where only some consequences for computability theory are discussed. Here we extend the results of [7], after defining a suitable model of *relativistic Turing machines* in Section 2. We observe that relativistic computing has precisely the power of recognizing the Δ_2-sets of the Arithmetical Hierarchy [14]. If the underlying physical theory is accepted, this would lift the barrier of recursiveness 'in nature' to the Δ_2-level, i.e. without violating any feasible thought experiments in general relativity theory. In Section 3 we give a complexity-theoretic characterization of relativistic computing in terms of a far simpler model from computational complexity theory: Turing machines with advice. We prove that under mild assumptions, $S(n)$-space bounded relativistic Turing machines are equivalent to 'classical' Turing machines with $S(n)$-bounded advice, with the further proviso that the advice is relativistically $S(n)$-space bounded computable. Given the

physical background, the required number of relativistic phases of computing is a crucial resource. We show that any language that is relativistically recognizable within $S(n)$-space, can be recognized by a $S(n)$-space bounded relativistic Turing machine that needs at most two relativistic phases (of a certain kind).

We emphasize that relativistic computing models are only claimed by certain theories in contemporary theoretical physics. The validity of the model has not been experimentally verified.

2 Relativistic and Other Computational Models

Our main goal is to compare the *infinite computations* performed by relativistic computers [7] with those performed in finite time by Turing machines *with advice*. In this section we describe the formal models of computations involved and give some basic constructions, cross-pointing to known concepts in recursion theory.

2.1 Relativistic Turing Machines

Recent research in theoretical physics suggests that in certain relativistic space-times, viz. in Malament-Hogarth space-times, it may be possible for a computer to receive the answer to a *yes-or-no question* from an *infinite*, i.e. arbitrarily long computation in finite proper time (cf. [6,7,10]). Our first goal is to cast relativistic computations in the framework of Turing machines with oracles [14].

Informally, a *relativistic Turing machine* (RTM) is a kind of multitape deterministic Turing machine with a separate read-only input and, in the case of transducers, a write-only output tape. That is, we will consider RTMs both as language recognizers and as transducers, computing partial functions that map finite strings of input symbols to similar strings of output symbols. A RTM has a number of special, distinguished states that allow the implementation of *relativistic phases* of computing. More formally, a RTM is a seven-tuple $R = (\Sigma, Q, \delta, q_0, q_F, q_R, S)$ where Σ is a finite alphabet, Q is a finite set of states, δ is a standard (deterministic) transition function, q_0, q_F, and $q_R \in Q$ are the distinguished *initial, final* and *relativistic states*, respectively, and $S \subset Q - \{q_0, q_F, q_R\}$ is the set of *signal states*. Set S consists of a distinguished, so-called *no-signal state* q_N, and one or more *yes-signal states* q_Y.

A *relativistic computation* by a RTM R proceeds as follows, modeling the features described in [7]. On a finite input w, R starts its computations as a standard Turing machine in state q_0. It reads symbols from the input tape, prints symbols to the output tape, rewrites the working tapes, enters new states from $Q - S - \{q_R\}$, etcetera, and continues in the 'classical' way of computing until it enters its relativistic state q_R. At this moment R starts a *relativistic phase* in its computation. R now proceeds like an oracle Turing machine that executes an oracle call, with noted differences in the way the 'relativistic call' is elaborated *and in the way information is extracted from a single call*. Informally, entering q_R is like spawning a copy of the computation and asking: *starting from the current instantaneous description and continuing computing in accordance*

with my transition function, will my copy ever enter a yes-signal state? If so, then *after finite time* a yes-signal state $\in S$ is entered by R, with the state corresponding to the specific yes-signal state that will be entered by the spawned copy. (In [7] the finite moment is referred to as a 'Malament-Hogarth event'.) Otherwise, i.e. if no yes-signal state is entered by the spawned copy by the time of the Malament-Hogarth event, then R enters no-signal state q_N.

In *normal relativistic mode* the spawning process just waits ('observes') until a signal state is entered. All data on the machine's tape remain as they were when entering q_R, and also the heads retain their previous positions. As soon as a signal-state is reached, necessarily after finite time, the present relativistic phase ends, even though the infinite computation that was spawned may still be continuing. In *extended mode* the spawning process can subsequently enter a finite number of yes-signal states from the computation, counting such moments in a buffer of finite size and going from yes-signal state to yes-signal state until either the buffer is full or it 'sees' by the time of the Malament-Hogarth event that no further yes-signal state will be entered (in which case it will transfer to q_N). In either case the relativistic phase ends and R resumes its computation from whatever signal-state it reached. R cannot switch to relativistic phases during a relativistic phase.

The description is only complete if it is specified when and what yes-signal state is to be entered during a relativistic phase, from a given finite set of possibilities. Following [7] we assume that a yes-signal is entered whenever some observable property P holds for the (instantaneous description of the) spawned computation. It means that R must be augmented with a mechanism for checking P in a relativistic phase, i.e. during the spawned computation as it goes to infinity. We allow the mechanism to output signals from a finite set of different yes-answers, corresponding to the different signal-states. In [7] the simple case with $P =$ '*halting*' is considered, which only needs an easy program. The resulting mechanism will be termed a *relativistic oracle* for P.

After a RTM has resumed its classical mode of computation, it can switch to a relativistic phase again (i.e. when q_R is entered again). Thus, a computation of a RTM alternates between classical and relativistic phases of computing. A classical phase begins in a state from $Q - \{q_R\}$ and ends in q_R or q_F. A relativistic phase begins in state q_R and ends in a signal state $\in S$ and, in extended mode, with a value in a buffer of finite size. A RTM can halt only in state q_F. If it works as a transducer, the result of the machine's computation is written on its output tape. We claim:

Postulate A *Relativistic computations are faithfully modeled by computations of ordinary Turing machines using relativistic oracles in normal or extended mode (i.e., by RTMs).*

In [7,21] the underlying assumptions are explained, viz. the observability of spawned infinite processes in finite physical time. Formally, our analysis shows that relativistic oracles are just oracles for 'eventually P' *except* when extended

mode is used. The latter mode was not explicitly distinguished in [7] but will be important.

It is not entirely clear whether repeated relativistic phases of computation during one computation of a RTM are physically feasible. The analysis in [7] does not seem to prevent it. In [21] the possibility of performing several (even an infinite number) of relativistic computational phases during a single computation is allowed. Extended relativistic mode will prove to be a highly effective intermediate. In general one should keep the number of relativistic oracle calls as low as possible, considering the physical justification. Bounding the number of oracle queries for arbitrary oracles is a well-studied issue in complexity theory (cf. [2]), which thus finds an interesting new motivation here.

2.2 Relativistic Computation

To assess the *computational power* of RTMs we assume some familiarity with the classes $\Sigma_i (i \geq 0)$ and $\Pi_i (i \geq 0)$ of the so-called *Arithmetical Hierarchy* in recursion theory, with $\Delta_i = \Sigma_i \cap \Pi_i$ for every $i \geq 0$ (cf. [14]). Recall that $\Sigma_1 =$ *'the recursively enumerable sets'* and $\Delta_1 =$ *'the recursive sets'*.

From a functional point of view the model of RTMs was considered in [7] under the name of *relativistic computer*. Using an obvious connection to classical TM-computations with the Halting Problem as oracle, it was proved in [7] that relativistic computers can decide all languages in Σ_1 and can recursively enumerate all languages in Σ_2, thus showing that relativistic computers go beyond the range of the Church-Turing thesis. The following observation can be made (modulo encodings), improving on [7].

Theorem 1. *Relativistic Turing machines (thus: relativistic computers) recognize precisely the Δ_2-sets of the Arithmetical Hierarchy.*

Proof: First we observe that extended relativistic mode can be simulated by iterating calls in normal relativistic mode, using a carefully tuned procedure in which the computation in a relativistic phase is traced from yes-answer to yes-answer (and the original instantaneous description is re-instated whenever the end of the simulation of the phase in extended mode is reached). The remainder of the argument parallels Post's theorem for Δ_2-sets (see[14]). Let A be recognized by a RTM R, with R necessarily always halting. Without loss of generality R works in normal relativistic mode. Now a recursive predicate $F(x, y, w)$ can be designed such that $w \in A \Leftrightarrow \exists_x \forall_y F(x, y, w)$, as follows. Use the $\exists_x$ to delineate a finite computation by R, extending finitely into those relativistic phases that are called and lead to a yes-signal up to the point that a yes-signal is generated, and use the $\forall_y$ to express that in this computation the relativistic phases that are called and do not lead to a yes-signal indeed do not generate such a signal in all steps to infinity. This shows that $A \in \Sigma_2$. At the same time we have $\overline{A} \in \Sigma_2$ as $\overline{A}$ is relativistically recognizable as well, hence $A \in \Pi_2$. Thus $A \in \Delta_2$.

Now let $A \in \Delta_2$. It means that there are recursive predicates F and G such that $w \in A \Leftrightarrow \exists_x \forall_y F(x, y, w)$ and $w \in \overline{A} \Leftrightarrow \exists_x \forall_y G(x, y, w)$. Now $w \in A$ can

be decided by a RTM R that enumerates the possible values of $x = 0, 1, \ldots$ and checks for each value in two relativistic phases whether $\forall_y F(x, y, w)$ or $\forall_y G(x, y, w)$. For some x, hence in finite time, one of the two must signal yes. □

In the model of RTMs as we described it, we abstracted from the underlying physical relativistic computational system and concentrate on the formal mechanism for invoking the *relativistic phases of computing* and for getting 'signals' from it in finite time. This makes it possible to study the effect of relativistic computations in bounded (memory-)space, very much like the complexity of oracle computations was studied.

For a RTM, we define its *space complexity* $S(n)$ as the maximum size of any instantaneous description in any computation on inputs of length n, taken over all (accepted) inputs of length n. However, *the space consumed by the spawned process in a relativistic phase of computation is not counted* in the space-complexity. Note that this space may be infinite. If only $O(1)$ extra cells are needed in addition to the input, we say that the machine operates *within constant extra space*. Note that $S(n) \geq n$ for all space bounds for RTMs and that for any $S(n) \geq n$, the family of languages recognized by RTMs in space $S(n)$ is closed under complement. There seems to be no justification for defining the time complexity of a RTM.

To illustrate the power of RTMs we present two RTMs in detail that recognize specific undecidable languages. The ideas of the constructions will be used later. Let K be the set of all strings w that are descriptions of Turing machines, denoted by $\langle w \rangle$, that accept their own description. It is well-known that K is not recursive, i.e. $K \in \Sigma_1 \setminus \Pi_1$ [15]. The following observation is straightforward.

Proposition 1. *There is a RTM R of constant extra space complexity that recognizes K. In fact, all languages of $\Sigma_1 \cup \Pi_1$ can be relativistically recognized in constant extra space and using at most one relativistic phase.*

The next proposition shows a more complicated RTM that recognizes a language by means of exponentially many relativistic sub-computations. However, far fewer phases will be seen to suffice as well. The language we consider is $K_a = \{w | MAX(w)\}$, where $MAX(w)$ holds if and only if w satisfies the following condition: $w \in K$ and, if $|w| = n$, then the running time of $\langle w \rangle$ on w is the maximum one over all such w's.

Proposition 2. *There is a RTM T of constant extra space complexity that recognizes language K_a. In fact, T needs at most two relativistic phases in normal mode, or at most one in extended mode.*

Proof: The idea is to design a RTM T that operates as follows, on any input w of length n. First it checks whether $w \in K$ and if so, it systematically generates, in lexicographic order, all descriptions v of TMs of length n. Doing so, T tries to determine those descriptions v that belong to K and satisfy $MAX(v)$. If a v is found that satisfies $w = v$ then T accepts; otherwise T rejects.

Proposition 1 shows that an RTM can easily decide whether a string of length n is a member of K. Assume that T has found two strings, u and v, from K. Now T must decide which of the two machines whose encoding is represented by u and v, has the longer running time. T can classically decide this by alternately simulating one step of each machine, to see which of the two machines halts sooner. However, in some cases this can consume an enormous amount of space which surely cannot in general be bounded by e.g. a polynomial in n. Therefore, for this particular task we design a relativistic strategy for machine T. We equip T with two halting signal states, q_1 and q_2. Entering q_1 will signal that the halting state of machine $\langle u \rangle$ is reached whereas entering q_2 will signal that the halting state of $\langle u \rangle$ is reached. Now, all that T has to do, having remembered both u and v, is to enter relativistic state q_R and spawn the alternating simulation of both machines $\langle u \rangle$ and $\langle v \rangle$ described above into the relativistic phase. When the phase ends, T can tell by the state it is in which of the two simulated machines will run longer. It is now easy to construct a RTM T that recognizes K_a, using linear space.

A RTM T that recognizes K_a in constant extra space can be obtained as follows. After determining that $w \in K$, one can relegate the entire search for (the existence of) a string $u \in K$ of length n for which $\langle u \rangle$ runs longer on u than $\langle w \rangle$ on w to a relativistic phase. Note that this would require only two relativistic phases. By setting a buffer value of 2, the process can be carried out in extended mode in one phase as follows. Carry out the computations of $\langle w \rangle$ on w and of all $\langle u \rangle$ on u simultaneously. Whenever the computation on some u ends before the computation on w, cancel the simulation on u. When the computation on w ends, emit the first yes-signal and proceed with the remaining simulations. The first one that now halts will emit a yes-signal as well, but no further yes-signals are emitted after this. (By the bound on the buffer the relativistic phase ends in fact when a second yes-signal is reached.) T can now decide whether $w \in K_a$ by just looking at the buffer value when it returns from the relativistic phase. □

Using a result from [2] it can be shown that RTMs with $n + 1$ relativistic phases are more powerful than RTMs with up to n relativistic phases, for any $n \geq 0$.

2.3 Non-uniform Computations

We will compare RTMs with *Turing machines with advice* (TM/A's) (cf. [1,9, 12]), to establish the link with non-uniform complexity theory. A TM/A is a classical Turing machine enhanced by an *advice function*. An advice function is a function $f : \mathbf{Z}^+ \to \Sigma^*$. For given n, $f(n)$ is called the advice for length n. An advice is called $S(n)$-bounded if for all n, the length of $f(n)$ is bounded by $S(n)$.

A TM/A operating on an input of size n, is allowed to call the value of its advice function only once during the computation, and it can call it only for the particular n. To realize an advice-call, a TM/A is equipped with a separate *advice tape* (that is initially empty) and a distinguished *advice state*. By

entering the advice state at time t the value of $f(n)$ will appear on the advice tape in a single step at time $(t+1)$. The mechanism of advice is very powerful and can provide a TM/A with highly non-recursive 'assistance'. Without further constraints, advice functions are as powerful as arbitrary oracles. We will be interested in advice functions whose values are bounded in length by some (computable) functions of n. Let $\mathsf{TM}/poly$ be the class of languages recognized by TM/A's with polynomially-bounded advice (cf. [1,9,12]). To prepare for a later construction, we show that $K \in \mathsf{TM}/poly$ (cf. Proposition 1).

Proposition 3. *There is a TM/A A with a linearly bounded advice function f that recognizes the language K and halts on all inputs.*

Proof: Define the advice function f as follows: $f(n) = u$ if and only if $|u| = n$, and u is the lexicographically first string of length n that satisfies $MAX(u)$ (the predicate MAX was defined before the statement of Proposition 2). A default value should be used in case no u with $|u| = n$ of the required property exists. Clearly, f is linearly bounded. On input w of length n, A works as follows. First, it checks whether w is the encoding of some Turing machine M. If so, then A calls its advice to get $f(n) = u$. Then A makes use of the idea we already saw in the proof of Proposition 2: A starts to alternate between simulating one step of M on u, and one step of $\langle w \rangle$ on w. Clearly, the simulation of $\langle u \rangle$ must terminate, and if it terminates earlier than the simulation of M then, thanks to the truth of $MAX(u)$, A can infer that M on w will never stop. Then A rejects w and halts. Otherwise, if the simulation of $\langle w \rangle$ halts sooner than the simulation of $\langle u \rangle$, A accepts w and halts. □

Note that all three previous propositions are related: we have a (non-recursive) language K recognized both by a RTM (Proposition 1) and by a TM/A (Proposition 3). Moreover, the advice function used by the TM/A is computable by a RTM (Proposition 2). In the next section we show that this is not a coincidence.

3 Relativistic Computing vs. Non-uniform Computing

Now we can formulate the main result, relating infinite relativistic and non-uniform finite computations. Let P be an arbitrary observable, i.e. recursive property (e.g. 'halting'), and let $\mathsf{RelSPACE}(P, S)$ be the family of $S(n)-$bounded advice functions that are $S(n)-$space bounded computable by an RTM on input of any length$-n$ string using the relativistic oracle for P (in normal mode). We will prove that space-bounded RTMs, i.e. Turing machines with relativistic oracles, are equivalent to Turing machines with advice, assuming the latter take their advices from a suitable family. Assume that $S(n)$ is space-constructible, and let $\sharp$ be a marker symbol.

Lemma 1. *Let language L be recognized by a $S(n)-$space bounded RTM. Then there is a function $f \in \mathsf{RelSPACE}(P, S)$ such that language $L' = \{w\sharp f(|w|) | w \in L\}$ is recognizable by a linear-space bounded RTM that needs at most two relativistic phases on every input.*

Proof: Let L be recognized by some $S(n)-$space bounded RTM R. First we note that R can be assumed to always halt on every input, by the classical fact that one could prevent it from cycling in its non-relativistic phases using a $S(n)-$space counter. To construct f, consider how R acts on inputs w with $|w| = n$ and try to eliminate the need for 'infinite time' relativistic phases. The idea is to let f provide an upper bound on the length of all relativistic phases in R's computations on inputs of length n that eventually lead to entering a yes-signal state and thus end by R's entering into the corresponding yes-signal state. We do this by letting $f(n) = 0x_n$, where x_n is an instantaneous description of R of length $\leq S(n)$ that triggers the longest relativistic phase of this kind over all such ID's of length $\leq S(n)$ (and thus over all inputs of length n). To let this be well-defined, we let x_n be the lexicographically smallest ID with this property and take $f(n) = \epsilon$ if no ID of this kind exists.

If R could get a purported value x of $f(|w|)$ from somewhere on input w, it might act as follows: verify that $x = f(|w|)$ in one relativistic phase, and run a classical simulation of R in another, replacing every call to the relativistic oracle by a direct computation dovetailed with the simulation of R on x while continuously checking P (and drawing the obvious conclusion if no yes-signal state has been entered by the time the run on x ends).

Following this idea, we design a RTM R' to recognize L' as follows. On receiving its input, R' checks whether it is of the form $w\sharp x$. It then proceeds to check that $x = f(|w|)$ as follows. The strategy is similar to the strategy used in Proposition 2. Note that the computation in any relativistic phase is fully determined by the ID of R at the beginning of this phase, and every ID that triggers a relativistic phase will be bounded in length by $S(n)$, with $n = |w|$. R' first checks whether x triggers a relativistic phase of R that leads to a yes-signal, by actually performing the relativistic phase if it is triggered ('phase 1'). If x passes this test, R' starts up a second relativistic phase ('phase 2') in which it constructs (the value of) $S(n)$, enumerates all ID's $y \neq x$ of length $\leq S(n)$ that R could admit on any input of length n (and that trigger a relativistic phase), starts a simulation of R on x and on all y's, and alternately carries out one step in each of the simulations. Whenever a computation on an ID $y \neq x$ ends (i.e. reaches a yes-signal state) while the computation on x is still going on, the simulation on y ends and is removed (and the yes-signal state is *not* entered). If no simulation on any $y \neq x$ survives before the simulation on x reaches its yes-signal state, the simulation process is sent into an infinite loop (without entering a yes-signal state ever). If some simulations on ID's $y \neq x$ have survived by the time the simulation on x reaches its yes-signal state, the simulation on x ends and is removed (and the yes-signal sate is *not* entered) and the simulation on the remaining ID's $y \neq x$ is continued. Whenever one of these remaining simulations reaches a yes-signal state, this yes-signal state is actually entered. Clearly the outcome of this relativistic phase will tell R' whether x is an ID that triggers the longest relativistic phase that leads to halting. The given argument is easily modified to include the check that x actually is the lexicographically smallest

with this property. So far no more than linear space was used (i.e. linear in $|w\sharp x|$).

Assuming $x = f(|w|)$, R' can now begin the actual recognition process of the string w. In principle R' could avoid all relativistic phases in the simulation of R, because it can bound how far it must go into any such phase in a classical simulation before knowing whether a yes-signal will be reached or not: R' can simply interleave the simulation with the simulation of R on x and use the latter as a yardstick. If no yes-signal is reached before the end of the yardstick, it knows that a yes-signal will never be reached and that it can proceed to q_N. To avoid the space-count for this, R' relegates the recognition process to a relativistic phase ('phase 3'): in this phase it simulates R on w, using (the computation on) x as the yardstick to do a finite classical simulation of every relativistic phase which R would otherwise have to carry out, and emitting a yes-signal if and when the simulation thus carried out actually lead to the recognition that $w \in L$. Observe that phase 3 always halts, because R is always halting. This implies that R' can combine phases 1 and 3 into one: a transfer to q_N will now mean that either x never terminates, or it does and either $w \notin L$ or the computation wasn't long enough to reach the conclusion that $w \in L$, i.e. $x \neq f(|w|)$. Thus, altogether R' needs only two relativistic phases.

Finally, observe that the function f is computable by an RTM in space $S(n)$ on any input of length n, by very much the same argument as employed above. A RTM R'' for it will first construct $S(n)$ space and then use it to enumerate every candidate ID y explicitly, testing for each y whether it qualifies (like we did for x in phase 1) and running a contest between y's that qualify as in the proof of Proposition 2. This can easily be carried out within space $S(n)$. Once the winning string x_n and input i_n has been found (or no string was found to qualify) the value of $f(n)$ can be output. □

By a further argument one can show that the language L' can be recognized using at most *one* relativistic phase in extended mode (but using $S(n)$ space).

We can now prove the main result. In the result we assume that the Turing machines with advice can check property P effectively, either as a subroutine or as an oracle. Recall that functions in the set $\mathsf{RelSPACE}(P, S)$ are always $S(n)-$bounded with $S(n) \geq n$.

Theorem 2. *The following are equivalent, for any language L and RTM-space bound $S(n)$:*

(i) L is recognized by a $S(n)-$space bounded RTM.
(ii) L is recognized by a TM/A that uses an advice function $\in \mathsf{RelSPACE}(P, S)$.

Proof: $(i) \rightarrow (ii)$ Let L be recognized by a $S(n)-$space bounded RTM R. By Lemma 1 there exists a function $f \in \mathsf{RelSPACE}(P, S)$ such that $L' = \{w\sharp f(|w|) | w \in L\}$ is recognizable by a (linear-space) RTM R' that uses at most three relativistic phases. Consider the particular RTM R' that was constructed in Lemma 1 and design a Turing machine T with advice f as follows. On input w of length n, T immediately calls its advice and lays out the string $w\sharp f(|w|)$. T

now prepares to simulate R'. Clearly it can skip phases 1 and 2 of R' and move straight to phase 3. T now carries out the simulation in phase 3 in an entirely classical way just like it was specified for R', using the yardstick idea to 'finitize' the relativistic phases of R. T accepts if and only if the computation leads a yes-signal.

$(ii) \rightarrow (i)$ Let L be recognized by a Turing machine T using an advice function $f \in \mathsf{RelSPACE}(P, S)$. Design a RTM R for L as follows. On input w, R first computes the string $f(n) = f(|w|)$ in space $S(n)$. It then triggers a relativistic phase in which it does the entire simulation of T, having the advice-value available whenever T calls for it. The relativistic phase triggers a yes-signal when T accepts. When R observes the result and enters into a yes-state, it accepts w. Clearly R recognizes L and does so in space $\mathrm{S}(n)$. □

In the theorem we can assume without loss of generality that the machines involved in (i) and (ii) halt on all inputs. Specializing the result to the case considered in [7], let $P =$ '*halting*' and let $\mathsf{RelPSPACE} = \bigcup_k \mathsf{RelSPACE}(P, n^k)$. Let $\mathsf{RTM\text{-}PSPACE}$ denote the class of languages recognizable by RTMs in polynomial space.

Corollary 1. *The following are equivalent, for any language L and $S(n) \geq n$:*

(i) L is recognized by a polynomial-space bounded RTM.
(ii) L is recognized by a TM/A that uses an advice function $\in \mathsf{RelPSPACE}$.

Consequently, $\mathsf{RTM\text{-}PSPACE} \subseteq \mathsf{TM}/poly$.

Using well-known facts from non-uniform computational complexity (cf. [1]) it follows that computations by RTMs are as powerful as computations of infinite families of (finite) non-uniform circuits whose size grows faster than any recursive function. A further observation can be made that shows the 'physical' efficiency of relativistic phases in extended mode.

Theorem 3. *If a language L is recognized by a $S(n)-$space bounded RTM, then it can be recognized by a $S(n)-$space bounded RTM that needs at most two relativistic phases, one in extended mode and one in normal mode.*

Proof: Let L be recognized by a $S(n)-$space bounded RTM R. By Theorem 2, L can be recognized by a TM/A T, using the advice function $f \in \mathsf{RelSPACE}(P, S)$ constructed in the proof of Lemma 1. But a RTM can compute $f(n)$ as follows. Using one relativistic phase in extended mode, the RTM can count the number of ID's y of length $\leq S(n)$ that trigger a relativistic phase that actually leads to a yes-signal ('halts'). Once it knows this number, R can determine in a completely classical dovetail which of the computations on these ID's actually halt and what the lexicographically smallest y is that leads to the longest halting computation among them. (This 'census approach' is well-known in complexity theory, cf. [9].) This shows that L can be recognized by a RTM R that computes like T and computes the advice value when it is needed, using at most one relativistic phase in extended mode for this purpose and

no other relativistic calls. In order to achieve the bounded space complexity, let R relegate the entire (classical) simulation of T to a relativistic phase in normal mode. In this way R remains relativistically $S(n)-$space bounded and recognizes L by means of at most two relativistic phases. □

4 Conclusion

Using arguments from general relativity theory, Etesi and Németi [7] have argued that the Church-Turing thesis may be physically untenable. They have shown that relativistic computers can be thought of that can recognize non-recursive languages. We have formalized the model of relativistic computing and characterized its potential precisely from the viewpoint of computability theory. We also proved a bridging result that links the infinite computations of (space-bounded) relativistic TMs to finite computations by TMs with powerful advice functions.

The result, proving a kind of duality between infinite relativistic and non-uniform finite computations, follows from basic complexity-theoretic considerations and should be of interest for the appraisal of relativistic computing in the context of (theoretical) physics or philosophy. The result can be seen as a further evidence for the emerging central role of non-uniform computation models in capturing the information processing capabilities in natural systems, as formulated in [17]. The result complements the existing set of examples of such kinds of models, including evolutionary interactive systems, the Internet, artificial living systems, social systems, and amorphous computing systems (cf. [18]), by systems operating by the principles of general relativity theory.

References

1. J. L. Balcázar, J. Díaz, J. Gabarró: *Structural complexity I*, Second Edition, Springer-Verlag, Berlin, 1995.
2. R. Beigel, *Query-limited reducibilities*, PhD thesis, Department of Computer Science, Stanford University, Stanford, 1995, available at www.eccc.uni-trier.de/eccc-local/ECCC-Theses/beigel.html.
3. M. Blum, F. Cucker, M. Shub, M. Smale: *Complexity and real computation.* Springer, New York, 1997, 453 p.
4. M. Burgin: How we know what technology can do, *Communications of the ACM* 44 (2001) 83-88.
5. C.S. Calude, Pavlov, B.: Coins, quantum measurements, and Turing's barrier, Technical Report CDMTCS-170, University of Auckland, New Zealand, December 2001.
6. J. Earman, J. Norton: Infinite pains: The trouble with supertasks, in A.Morton and S.P. Stich (eds.), *P. Benacerraf and his critics*, Philosophers and Their Critics Vol 8, Blackwell Publ., Oxford (UK) / Cambridge (MA), 1996, Ch 11, pp 231-261.
7. G. Etesi, I. Németi: Non-Turing computations via Malament-Hogarth space-times, *Int. J. Theor. Phys.* 41 (2002) 341-370, see also: http://arXiv.org/abs/gr-qc/0104023.

8. J.D. Hamkins, A. Lewis: Infinite time Turing machines, *Journal of Symbolic Logic* Vol. 65, No. 2, pp. 567-6-4, 2000.
9. L.A. Hemaspaandra, M. Ogihara: *The complexity theory companion*, Springer-Verlag, Berlin, 2002.
10. J. Hogarth: Non-Turing computers and non-Turing computability, in D. Hull, M. Forbes and R.M. Burian (eds.), *1994 Biennial Meeting of the Philosophy of Science Association* (PSA 1994), Proceedings Vol. One, Philosophy of Science Association, East Lansing, 1994, pp 126-138.
11. J. Hopcroft, R. Motwani, and J.D. Ullman: *Introduction to automata theory, languages and computation*, 2nd Edition, Addison-Wesley, Reading, MA, 2000.
12. R.M. Karp, R.J. Lipton: Some connections between non-uniform and uniform complexity classes, in *Proc. 12th Annual ACM Symposium on the Theory of Computing* (STOC'80), 1980, pp. 302-309.
13. J.D. Norton: What can we learn about ontology of space and time from the theory of relativity?, *PhilSci Archive* (http://philsci-archive.pitt.edu/), ID: PITT-PHILSCI00000138, 2001.
14. H. Rogers Jr, *Theory of recursive functions and effective computability*, McGraw-Hill, New York, 1967.
15. A.M. Turing: On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc.,* 42-2 (1936) 230-265; A correction, *ibid*, 43 (1937), pp. 544-546.
16. A.M. Turing: Systems of logic based on ordinals, *Proc. London Math. Soc. Series* 2, 45 (1939), pp. 161-228.
17. J. van Leeuwen, J. Wiedermann: The Turing machine paradigm in contemporary computing, in: B. Enquist and W. Schmidt (Eds.), *Mathematics unlimited - 2001 and beyond*, Springer-Verlag, 2001, pp. 1139-1155.
18. J. van Leeuwen, J. Wiedermann: Beyond the Turing limit: evolving interactive systems, in: L. Pacholski, P. Ružička (Eds.), *SOFSEM'01: Theory and Practice of Informatics*, 28th Conference on Current Trends in Theory and Practice of Informatics, Lecture Notes in Computer Science Vol. 2234, Springer-Verlag, Berlin, 2001, pp. 90–109.
19. P. Wegner: Why interaction is more powerful than algorithms, *C. ACM* 40, (1997) 315-351.
20. J. Wiedermann, J. van Leeuwen: Emergence of super-Turing computing power in artificial living systems, in: J. Kelemen, P. Sosík (Eds.), *Artificial Life 2001*, 6-th European Conference (ECAL 2001), Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 2001, pp. 55-65.
21. L. Wischik: *A formalisation of non-finite computation*, dissertation submitted to the University of Cambridge towards the degree of Master of Philosophy, June 1997 (available at: www.wischik.com/lu/philosophy/non-finite-computation.html).

Quantum Optimization Problems*
(Extended Abstract)

Tomoyuki Yamakami

School of Information Technology and Engineering
University of Ottawa, Ottawa, Ontario, Canada K1N 6N5
yamakami@site.uottawa.ca

Abstract. Krentel [J. Comput. System. Sci., 36, pp.490–509] presented a framework for combinatorial NP optimization problems that search optimal values of polynomial-size solutions computed deterministically in polynomial time. This paper applies his framework to a quantum expansion of such optimization problems. With the notion of an "universal" quantum function similar to a classical "complete" function, we exhibit canonical quantum optimization problems whose optimal cost functions are universal for certain classes of quantum optimization problems. We also study the complexity of quantum optimization problems in connection to well-known complexity classes.

1 Combinatorial Optimization Problems

Combinatorial optimization problems have arisen in many areas of computer science. A typical example of such optimization problems is the Traveling Salesperson Problem, which asks, from a given map of cities and traveling cost, for a tour of all the cities at the least cost. A combinatorial optimization problem is in general a certain type of a search problem that is to find, among possible candidates, a feasible solution with the maximal (or minimal) value, where values are usually expressed by numbers[1]. NP-complete decision problems naturally induce their optimization counterparts; however, it is believed that not all such optimization problems are equally hard to solve.

Most optimization problems are fit into the following framework. For a partial function f, let $\mathrm{dom}(f)$ denote the *domain* of f.

Definition 1. *A* (combinatorial) optimization problem[2] Π *is a quadruple* (op, I, S, g)*, where op is either* max *(maximization operator) or* min *(minimization operator),* I *is a set of instances,* S *is a set of candidates of (feasible)*

* This work was in part supported by Natural Sciences and Engineering Research Council of Canada and Japan Science and Technology Corporation.

[1] The notations $\mathbb{N}$, $\mathbb{R}$, and $\mathbb{C}$ denote the sets of all natural numbers (i.e., nonnegative integers), of all real numbers, and of all complex numbers, respectively. Let $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. Let $\mathbb{D}$ denote the collection of all dyadic rational numbers (i.e., rational numbers of the form $0.r$ or $-0.r$ for certain finite series r of 0s and 1s).

[2] If $op = \max$, then we call Π a maximization problem and otherwise, we call Π a minimization problem.

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 300–314, 2002.

solutions, and g is a partial cost function from $I \times S$ to $\mathbb{R}$. The instance-solution relation is $R_g = dom(g)$. For each instance $x \in I$, $sol_\Pi(x) = \{s \in S \mid R_g(x,s)\}$ is the set of solutions of x for Π. The optimal cost function g^* is defined as $g^*(x) = op\{g(x,s) \mid s \in sol_\Pi(x)\}$ and the set of optimal solutions of x for Π is $optsol_\Pi(x) = \{s \in sol_\Pi(x) \mid g^*(x) = g(x,s)\}$ for every instance $x \in I$.

By identifying instances (such as, functions, graphs, etc.) with their representations by binary strings, it suffices to consider binary strings as instances given to our optimization problems Π. Thus, we fix our alphabet Σ to be $\{0,1\}$ throughout this paper and let Σ^* be the set of all strings over Σ. We assume the standard dictionary order on Σ^*. For any two strings $s, t \in \Sigma^*$, we use the notation $s < t$ to mean that s precedes y in this order. Write $s \leq t$ when $s < t$ or $s = t$. In this paper, a *polynomial* means a multi-variate polynomial with nonnegative integer coefficients. The reader may refer to, e.g., [17] for a more general treatment of the optimization problems.

1.1 Krentel's Framework for Optimization Problems

From a complexity-theoretical point of view, Krentel [11] laid out a framework for the study of NP optimization problems. He focused on the computational complexity of optimal cost functions for NP optimization problems rather than the complexity of finding their optimal solutions. In his seminal paper, Krentel defined the optimization class OptP to be the collection of optimal cost functions for optimization problems with polynomial-size solutions and polynomial-time computable cost functions. To locate the hardest functions in OptP, Krentel introduced the notion of "completeness" under his metric reduction. The Traveling Salesperson Problem turns out to have a "complete" optimal cost function for OptP [11].

Krentel's framework has been extended in several different manners in the literature [5,12,4]. This paper generalizes Krentel's framework to introduce a generalized optimization class Opt$\mathcal{F}$ for the subsequent sections. Although Krentel dealt with both maximization and minimization problems, we discuss only maximization problems for brevity.

Definition 2. *Let $\mathcal{F}$ be any set of partial functions from $\Sigma^* \times \Sigma^*$ to $\mathbb{R}$. A partial function f from $\Sigma^* \times \Sigma^*$ to $\mathbb{R}$ is in* Opt$\mathcal{F}$ *if there exist a polynomial p and a maximization problem Π with Σ^* as the set of instances and with g as the partial cost function chosen from $\mathcal{F}$ such that (i) $sol_\Pi(x) \subseteq \Sigma^{p(|x|)}$ for every x and (ii) f is the maximal cost function for Π; that is, for every instance x,*

$$f(x) = \max\{g(x,s) \mid (x,s) \in dom(g) \text{ and } s \in \Sigma^{p(|x|)}\}$$

if such an $g(x,s)$ exists. Otherwise, let $f(x)$ be undefined. The class $\mathcal{F}$ is called the underlying class *of* Opt$\mathcal{F}$.

Since, in Definition 2, the cost function g completely characterizes the optimization problem Π, we simply write $sol_g(x)$ for $sol_\Pi(x)$ and $optsol_g(x)$ for $optsol_\Pi(x)$. Any element in $sol_g(x)$ is also called a *classical solution at x for Π*.

We often translate a binary outcome of a Turing machine into a number by identifying Σ^* with $\mathbb{N}$, $\mathbb{Z}$, or $\mathbb{D}$ in a standard manner[3]. We also assume an efficient paring function (i.e., bijection) $\langle\ ,\ \rangle$ from $\Sigma^* \times \Sigma^*$ to Σ^*.

Definition 2 enables us to introduce various optimization classes Opt$\mathcal{F}$ from underlying classes $\mathcal{F}$. Let FP (FPSPACE, resp.) denote the class of polynomial-time (polynomial-space[4], resp.) computable *total* functions from Σ^* to Σ^*. By identifying $\Sigma^* \times \Sigma^*$ with Σ^* (by the paring function), we can view FP and FPSPACE as classes of functions from $\Sigma^* \times \Sigma^*$ to Σ^* and therefore, we can define the quantum optimization class OptFP and OptFPSPACE. Notice that OptFP coincides with Krentel's OptP[5]. It is widely believed that OptFP $\neq$ FP; however, it holds that OptFPSPACE = FPSPACE. Between OptFP and OptFPSPACE lies another important optimization class Opt#P, which is induced from Valiant's counting class #P [18]. Clearly, OptP $\cup$ #P $\subseteq$ Opt#P $\subseteq$ FPSPACE since FP $\subseteq$ #P by identifying binary strings with natural numbers.

In the subsequent section, we attempt to expand, within Krentel's framework, the classical optimization problems to quantum optimization problems and initiate a study of the computational complexity of such problems.

2 From Classical Optimization to Quantum Optimization

Krentel [11] studied the complexity of optimal cost functions g^* induced from given cost functions g in FP. Applying his framework to a quantum context, we can introduce a new notion of a quantum optimization problem. Here, a "quantum optimization problem" roughly refers to an optimization problem, as given in Definition 1, whose cost function is computable in a quantum-mechanical fashion. Our mathematical model of quantum computation is a multi-tape quantum Turing machine (referred to as QTM) defined in [3,16,20]. The formal definition of a QTM is given in Appendix for the reader's sake. In this paper, amplitudes of QTMs are all drawn from $\tilde{\mathbb{C}}$, the set of *polynomial-time approximable* complex numbers (that is, the real and imaginary parts are approximated to within 2^{-k} in time polynomial in the size of input 1^k).

In this paper, we investigate only two types of quantum optimization problems that naturally expand the platform for classical optimization problems and we leave other platforms for quantum optimization problems to the avid reader.

[3] To translate an integer or a dyadic rational number into a string, we use the first bit (called the *sign bit*) of the string to express the sign (that is, + or −) of the number.

[4] In this paper, we require all work tapes of basis Turing machines to be bounded above by certain polynomials and thus, the output strings of the machines are of at most polynomial size.

[5] Even though Krentel originally included the minimization problems to OptP, this paper treats OptP as a set of maximization problems, as in [10].

2.1 Quantum Optimization of Classical Solutions

Quantum computation has attracted a wide range of interests in solving classical problems faster and efficiently. This section pays its special attention to optimization problems that search classical solutions with quantum cost functions chosen from the class FBQP[6]. The quantum optimization class OptFBQP is formally introduced by Definition 2 from underlying class FBQP.

Clearly, FBQP $\subseteq$ OptFBQP. We then consider an upper bound of OptFBQP. A simple way of computing its optimal cost function g^* from any *total* cost function g is illustrated as follows. Let p be a polynomial that defines the size of solutions of each x. We first encode into set A the information on optimal solutions given by g: let $A = \{\langle x, y\rangle \mid \exists z \in \Sigma^{\leq p(|x|)} \forall u \in \Sigma^{p(|x|)} [|yz| = p(|x|) \wedge g(x, yz) \geq g(x, u)]\}$. This implies that $A \in \Sigma_2^{\mathrm{P}}(g)$, where $\Sigma_2^{\mathrm{P}}(g)$ means the second Σ-level of the polynomial hierarchy relative to g. It is not difficult to show that $g^* \in \mathrm{FP}^A \subseteq \mathrm{FP}^{\Sigma_2^{\mathrm{P}}(g)}$. Therefore, we conclude:

Lemma 1. $\mathrm{FBQP} \subseteq \mathrm{OptFBQP} \subseteq \mathrm{FP}^{\Sigma_2^{\mathrm{P}}(\mathrm{BQP})}$.

Lemma 1 leads to the following result.

Proposition 1. $\mathrm{OptFBQP} = \mathrm{FBQP}$ *if and only if* $\mathrm{NP}^{\mathrm{BQP}} = \mathrm{BQP}$.

Proof. (If – part) Assume that $\mathrm{NP}^{\mathrm{BQP}} = \mathrm{BQP}$. This implies that $\Sigma_2^{\mathrm{P}}(\mathrm{BQP}) = \mathrm{BQP}$. By Lemma 1, $\mathrm{OptFBQP} \subseteq \mathrm{FP}^{\Sigma_2^{\mathrm{P}}(\mathrm{BQP})} \subseteq \mathrm{FP}^{\mathrm{BQP}}$. Since $\mathrm{FBQP} = \mathrm{FP}^{\mathrm{BQP}}$, we have $\mathrm{OptFBQP} \subseteq \mathrm{FBQP}$.

(Only If – part) Assume that $\mathrm{OptFBQP} = \mathrm{FBQP}$. Let $A \in \mathrm{NP}^{\mathrm{BQP}}$. We first show that the characteristic function[7] χ_A of A belongs to OptFBQP. Since $A \in \mathrm{NP}^{\mathrm{BQP}}$, there exist a polynomial p and a set $B \in \mathrm{BQP}$ such that, for every x, $x \in A$ if and only if $\exists y[|y| = p(|x|) \wedge \langle x, y\rangle \in B]$. Define $g(x, y)$ to be 1 if $\langle x, y\rangle \in B$ and 0 otherwise. Since $B \in \mathrm{BQP}$, g belongs to FBQP. It thus follows that $\chi_A(x) = \max\{g(x, s) \mid s \in \Sigma^{p(|x|)}\}$ for all x. This implies that $\chi_A \in \mathrm{OptFBQP}$. By our assumption, χ_A is thus in FBQP. Therefore, A belongs to BQP. □

We turn our interest to another class of quantum optimization problems. Similar to Opt#P, we define Opt#QP from underlying class #QP[8], which is a quantum analogue of #P. We first exhibit an example of an Opt#QP-function. Assume that each deterministic Turing machine (referred to as DTM) is effectively encoded into a certain string. The *code* of an amplitude α in $\tilde{\mathbb{C}}$ is the minimal code of any DTM that approximates α to within 2^{-n} in time polynomial in n. Moreover, the *code* of a QTM is the description of the machine

[6] A function f from Σ^* to Σ^* is in FBQP if there exists a polynomial-time, $\tilde{\mathbb{C}}$-amplitude, well-formed QTM M with an output tape such that, for every x, M writes string $f(x)$ in its output tape with probability at least 3/4 [21].

[7] The characteristic function χ_A of A is defined as $\chi_A(x) = 1$ if $x \in A$ and 0 otherwise for every x.

[8] A function f from Σ^* to the unit real interval $[0, 1]$ is in #QP ("sharp" QP) if there exists a polynomial-time well-formed QTM M with $\tilde{\mathbb{C}}$-amplitudes such that, for every x, $f(x)$ is the probability that M accepts x [21].

together with the codes of all amplitudes used by the QTM. Assume that every code is effectively expressed in binary. Let M_U denote Bernstein-Vazirani's universal QTM[9], which, on input $\langle M, x, 1^t, 1^m\rangle$, simulates QTM M on input x for t steps and halts in a superposition $|\phi_{M_U}\rangle$ of final configurations satisfying that $\||\phi_{M_U}\rangle - |\phi_M\rangle\| \leq 1/m$, where $|\phi_M\rangle$ expresses the superposition of configurations of M on x after t steps [3]. For completeness, we assume that if M is not a well-formed QTM then M_U rejects the input with probability 1. Now, the Maximum Qtm Problem (MAXQTM) is defined as follows:

Maximum Qtm Problem: MAXQTM

- instance: $\langle M, x, 1^t, 1^m\rangle$, where M is a $\tilde{\mathbb{C}}$-amplitudes well-formed QTM, t is in $\mathbb{N}$, and m is in $\mathbb{N}^+$.
- question: find the maximal acceptance probability, over all binary strings s of length $|x|$, of M_U on input $\langle M, xs, 1^t, 1^m\rangle$.

It is obvious that MAXQTM belongs to Opt#QP since the function computing the acceptance probability of M_U is in #QP.

In what follows, we discuss the complexity of Opt#QP. Lemma 2 shows that Opt#QP naturally expands Opt#P but stays within the class $\#\mathrm{QP}^{\mathrm{NP}^{\mathrm{PP}}}$, where a relativized class is obtained by replacing QTMs with oracle QTMs.

Lemma 2. *1. For every $f \in$ Opt#P, there exist two functions $g \in$ Opt#QP and $\ell \in$ FP such that, for all x, $f(x) = g(x)\ell(1^{|x|})$.*

2. $\#\mathrm{QP} \subseteq \mathrm{Opt}\#\mathrm{QP} \subseteq \#\mathrm{QP}^{\mathrm{NP}^{\mathrm{PP}}}$.

For the proof of Lemma 2, we need the following fact (obtained from results in [21]): the three statements below are all equivalent: for any set A, (i) $A \in$ PP; (ii) there exist two functions $f, g \in \#\mathrm{QP}$ such that, for every x, $x \in A$ if and only if $f(x) > g(x)$; and (iii) there exist two functions $f, g \in \mathrm{GapQP}$[10] such that, for every x, $x \in A$ if and only if $f(x) > g(x)$.

Proof of Lemma 2. 1) Let f be any function in Opt#P. That is, there exists a polynomial p and a function $h \in \#\mathrm{P}$ such that $f(x) = \max\{h(\langle x, s\rangle) \mid s \in \Sigma^{p(|x|)}\}$. By the argument used in [21], we can define $k \in \#\mathrm{QP}$ and $\ell \in \mathrm{FP}$ that satisfy the following condition: $k(\langle x, s\rangle) = 0$ if $|s| \neq p(|x|)$, and $k(\langle x, s\rangle)\ell(1^{|x|}) = h(\langle x, s\rangle)$ otherwise. For the desired g, define $g(x) = \max\{k(\langle x, s\rangle) \mid s \in \Sigma^{p(|x|)}\}$ for each x.

2) Let f be any function in Opt#QP and take a function $g \in \#\mathrm{QP}$ and a polynomial p such that $f(x) = \max\{g(x, s)(x) \mid |s| = p(|x|)\}$ for all x. Choose the lexicographically minimal string $s_x \in \Sigma^{p(|x|)}$ such that $f(x) = g(x, s_x)$. Define $A = \{\langle x, s, t, y\rangle \mid \exists z(s \leq z \leq t \wedge g(x, z) \geq 0.y)\}$. Note that A belongs to $\mathrm{NP}^{\mathrm{PP}}$. The function f is computed as follows using A as an oracle. Let x be

[9] From the construction of a universal QTM in [3], Bernstein-Vazirani's universal QTM M_U has $\{0, \pm 1, \pm\cos\theta_0, \pm\cos\theta_0, \pm e^{i\theta_0}\}$-amplitudes, where $\theta_0 = 2\pi\sum_{i=1}^{\infty} 2^{-2^i}$.

[10] A function f from Σ^* to $[-1, 1]$ is in GapQP if there exists a polynomial-time well-formed QTM M with $\tilde{\mathbb{C}}$-amplitudes such that, for every x, $f(x)$ is the difference between the acceptance probability of M on input x and the rejection probability of M on x [21].

any input of length n. By a binary search algorithm, we find a string $\hat{y}$ such that (i) $g(x, s_x) \geq 0.\hat{y}$ and (ii) for every $z \in \Sigma^{p(|x|)}$, $g(x, z) < g(x, s_x)$ implies $g(x, z) < 0.\hat{y}$. Then, by decreasing the interval $[s, t]$ in a way similar to the binary search, we can find a string s_0 of length $p(|x|)$ such that $g(x, s_0) \geq 0.\hat{y}$. In the end, we compute $g(x, s_0)$ in a quantum fashion. Thus, f is in $\#\mathrm{QP}^A \subseteq \#\mathrm{QP}^{\mathrm{NP}^{\mathrm{PP}}}$. □

Note that $\mathrm{EQP} \subseteq \mathrm{WQP} \subseteq \mathrm{NQP} \subseteq \mathrm{PP}$, where WQP[11] is a quantum analogue of WPP [7]. Lemma 2(2) yields the following collapse result. For any function f and any number $m \in \mathbb{N}^+$, the notation f^m denotes the function satisfying $f^m(x) = (f(x))^m$ for all x.

Theorem 1. $\mathrm{NP}^{\mathrm{PP}} = \mathrm{EQP} \Longrightarrow \mathrm{Opt}\#\mathrm{QP} = \#\mathrm{QP} \Longrightarrow \mathrm{NP}^{\mathrm{PP}} = \mathrm{WQP}$.

Proof. We show the first implication. Assume that $\mathrm{EQP} = \mathrm{NP}^{\mathrm{PP}}$. By Lemma 2(2), we then obtain $\mathrm{Opt}\#\mathrm{QP} \subseteq \#\mathrm{QP}$ since $\#\mathrm{QP}^{\mathrm{EQP}} = \#\mathrm{QP}$ [21].

For the second implication, assume that $\#\mathrm{QP} = \mathrm{Opt}\#\mathrm{QP}$. Let A be an arbitrary set in $\mathrm{NP}^{\mathrm{PP}}$. There exist a set $B \in \mathrm{PP}$ and a polynomial p such that, for every x, $x \in A$ if and only if $\exists y[|y| = p(|x|) \wedge \langle x, y\rangle \in B]$. As shown in [21], PP coincides with its quantum analogue $\mathrm{PQP}_{\tilde{\mathbb{C}}}$. Let N_B be a polynomial-time, $\tilde{\mathbb{C}}$-amplitude, well-formed QTM that recognizes B with error probability $< 1/2$. Define $f(x) = \max\{\rho_B(x, y) \mid y \in \Sigma^{p(|x|)}\}$, where $\rho_B(x, y)$ to be the acceptance probability of N_B on input $\langle x, y\rangle$. Thus, the function f is in $\#\mathrm{QP}$ and, for every x, (i) if $x \in A$ then $f(x) > 1/2$ and (ii) if $x \notin A$ then $f(x) < 1/2$. Define $g(x) = \max\{f(x), \frac{1}{2}\}$ for every x. This g is clearly in $\mathrm{Opt}\#\mathrm{QP}$ and thus in $\#\mathrm{QP}$. By the definition of g, if $x \in A$ then $g(x) > 1/2$ and otherwise $g(x) = 1/2$. Next, define $h(x) = g(x) - \frac{1}{2}$ for all x's. Since $\mathrm{GapQP} = \#\mathrm{QP} - \#\mathrm{QP}$ [21], h is in GapQP. Thus, h^2 is in $\#\mathrm{QP}$ [21]. For this h^2, it follows that $x \in A$ implies $h^2(x) > 0$ and $x \notin A$ implies $h^2(x) = 0$. At this moment, we obtain that $A \in \mathrm{NQP}$.

Since h^2 is in $\#\mathrm{QP}$, there exists a polynomial p such that, for every x, if $h^2(x) > 0$ then $h^2(x) > 2^{-p(|x|)}$ because the acceptance probability of a QTM is expressed in terms of a polynomial in the transition amplitudes from $\tilde{\mathbb{C}}$. To complete the proof, we define $k(x) = \min\{h^2(x), 2^{-p(|x|)}\}$ for every x. It follows that $x \in A$ implies $k(x) = 2^{-p(|x|)}$ and $x \notin A$ implies $k(x) = 0$. To see that k is in $\#\mathrm{QP}$, consider the fact that $1 - k(x) = \max\{1 - h^2(x), 1 - 2^{-p(|x|)}\}$ is in $\mathrm{Opt}\#\mathrm{QP}$, which is $\#\mathrm{QP}$ by our assumption. Thus, $1-(1-k(x))$ is also in $\#\mathrm{QP}$, which guarantees that k is in $\#\mathrm{QP}$. □

Since $\mathrm{P}^A = \mathrm{PP}^A$ for any PSPACE-complete set A, we have $\mathrm{Opt}\#\mathrm{QP}^A = \#\mathrm{QP}^A$. Note that if $\mathrm{NP}^{\mathrm{PP}} = \mathrm{WQP}$ then $\mathrm{P}^{\mathrm{NQP}} = \mathrm{PP}$. Green [8], however, constructed an oracle B such that $\mathrm{PP}^B \not\subseteq \mathrm{P}^{\mathrm{NQP}^B}$. Hence, we conclude that $\mathrm{NP}^{\mathrm{PP}^B} \neq \mathrm{WQP}^B$, which further implies $\mathrm{Opt}\#\mathrm{QP}^B \neq \#\mathrm{QP}^B$ since Theorem 1 relativizes.

[11] A set S is in WQP ("wide" QP) if there exist two functions $f \in \#\mathrm{QP}$ and $g \in \mathrm{FEQP}$ such that, for every x, (i) $g(x) \in (0, 1] \cap \mathbb{Q}$ and (ii) if $x \in S$ then $f(x) = g(x)$ and otherwise $f(x) = 0$ [21], where we identify a string with a rational number expressed as a pair of integers (e.g., $g(x) = \frac{1}{3}$).

2.2 Quantum Optimization of Quantum Solutions

We investigate the complexity of quantum optimization problems whose solutions are particularly quantum strings (qustrings, in short) of polynomial size, where a *qustring of size* n is a pure quantum state of dimension 2^n. Let Φ_n be the collection of all qustrings of size n and set $\Phi_\infty = \bigcup_{n\in\mathbb{N}^+} \Phi_n$. To distinguish these solutions from classical solutions, we call them *quantum solutions*. In this case, our cost functions are partial functions from $\Sigma^* \times \Phi_\infty$ to $\mathbb{R}$ and their corresponding optimal cost functions are given by maximizing the values of the cost functions over all quantum solutions of polynomial size.

We begin with the general definition.

Definition 3. *Let $\mathcal{F}$ be a set of partial functions from $\Sigma^* \times \Phi_\infty$ to $\mathbb{R}$. A partial function f from Σ^* to $\mathbb{R}$ is in* Qopt$\mathcal{F}$ *if there exist a polynomial p and a maximization problem Π with Σ^* as the set of instances and g as a partial cost function chosen from $\mathcal{F}$ such that, for every instance x,*

$$f(x) = \sup\{g(x, |\phi\rangle) \mid (x, |\phi\rangle) \in dom(g) \text{ and } |\phi\rangle \in \Phi_{p(|x|)}\}$$

if such an $g(x, |\phi\rangle)$ exists. Otherwise, let $f(x)$ be undefined. For simplicity, we say that g witnesses *f* with *p. For each x, let $sol_g(x) = \{|\phi\rangle \in \Phi_{p(|x|)} \mid (x, |\phi\rangle) \in dom(g)\}$ and any element in $sol_g(x)$ is called a quantum solution of x for Π.*

To deal with a function from $\Sigma^* \times \Phi_\infty$ to $\mathbb{R}$, we need to feed a QTM with a pure quantum state as an input. Such a quantum input produces a superposition of initial configurations of the QTM. In this way, we can expand the definition of a function f with domain $\Sigma^* \times \Sigma^*$ to its corresponding function $\check{f}$ with domain $\Sigma^* \times \Phi_\infty$ without altering the basis QTM that defines f. This function $\check{f}$ is called the *quantum extension* of f. By abusing the notation, we use the same symbol f to cope with both functions. We also use the same convention for a set of strings. Using this quantum extension, the class #QP, for instance, can be naturally expanded to a class of functions from $\Sigma^* \times \Phi_\infty$ to $\mathbb{R}$.

In comparison with Opt#QP, we consider the quantum optimization class Qopt#QP. This class contains the problem of computing maximal eigenvalues of certain Hermitian, positive semi-definite, contractive matrices. For a quantum circuit C, $C^\dagger$ denotes its reversing circuit. For simplicity, we use Shor's basis set of quantum gates to describe a quantum circuit. Moreover, we identify C with its corresponding unitary matrix. The *code* of a quantum circuit C is the description of how gates in C are interconnected by quantum channels. Define the Maximum Circuit Eigenvalue Problem (MAXCIR) as follows:

Maximum Circuit Eigenvalue Problem: MAXCIR

- instance: $\langle 1^n, C\rangle$, where $n \in \mathbb{N}$ and C is the code of a quantum circuit acting on n qubits.
- question: find the maximal eigenvalue of matrix $C^\dagger \Pi_1 C$, where the (s,t)-entry of Π_1 is 1 if $s = t \in 1\Sigma^{n-1}$ and 0 otherwise for any $s, t \in \{0,1\}^n$.

Note that, since matrix $C^\dagger \Pi_1 C$ is Hermitian, positive semi-definite, and contractive[12], its maximal eigenvalue coincides with $\|C^\dagger \Pi_1 C\|$, which equals $\sup\{\|\Pi_1 C|\phi\rangle\|^2 \mid |\phi\rangle \in \Phi_n\}$. This implies that MAXCIR belongs to Qopt#QP.

We next examine fundamental properties of Qopt#QP. It turns out that Qopt#QP enjoys important closure properties. Firstly, we note that Qopt#QP is closed under composition[13] with FP-functions; namely, Qopt#QP $\circ$ FP = Qopt#QP. Secondly, Qopt#QP satisfies the following closure properties. Let q be a polynomial, ℓ any function in $\mathbb{N}^{\mathbb{N}}$ with $\ell(n) \in O(\log n)$, and $\{|\phi_x\rangle\}_{x\in\Sigma^*}$ an ℓ-qubit source[14]. If f is a function in Qopt#QP, then the following functions belong to Qopt#QP: (i) $g(x) = \max\{f(\langle x, y\rangle) \mid y \in \{0,1\}^{q(|x|)}\}$, (ii) $g(x) = \sum_{y:|y|=\ell(|x|)} |\langle y|\phi_x\rangle|^2 \cdot f(\langle x, y\rangle)$, (iii) $g(x) = \prod_{y:|y|=\ell(|x|)} f(\langle x, y\rangle)$, and (iv) $g(\langle x, y\rangle) = f(x)^{|y|}$.

Moreover, Qopt#QP satisfies that Qopt(Opt#QP) = Opt(Qopt#QP) = Qopt#QP, where f is in Qopt(Opt#QP) if and only if there exists a function $h \in$ #QP such that, for every x, $f(x) = \sup_{|\phi\rangle} \max_s\{h(x, |\phi\rangle, s)\}$ with $|\phi\rangle$ running over all qustrings of polynomial size and s running over all strings of polynomial length. This shows the robustness of the class Qopt#QP.

In what follows, we explore a relationship between Qopt#QP and #QP. For our purpose, we need the new notations that are used to indicate a certain notion of "closeness" between two functions from Σ^* to $\mathbb{R}$.

Definition 4. *Let $\mathcal{F}$ and $\mathcal{G}$ be any two sets of functions from Σ^* to $\mathbb{R}$. For any function f from Σ^* to $\mathbb{R}$, we write $f \overset{\in}{\sim}{}^e \mathcal{F}$ ($f \overset{\in}{\sim}{}^p \mathcal{F}$, resp.) if, for every polynomial p, there exists a function $g \in \mathcal{F}$ such that, for every x, $|f(x)-g(x)| \leq 2^{-p(|x|)}$ ($|f(x)-g(x)| \leq 1/p(|x|)$, resp.). The notation $\mathcal{F} \Subset^e \mathcal{G}$ ($\mathcal{F} \Subset^p \mathcal{G}$, resp.) means that $f \overset{\in}{\sim}{}^e \mathcal{G}$ ($f \overset{\in}{\sim}{}^p \mathcal{G}$, resp.) for all functions f in $\mathcal{F}$.*

In Lemma 3 below, we show that any function in Qopt#QP can be closely approximated by functions taken from #QP with the help of oracles in QOP, where QOP is defined as follows. The following definition resembles the #QP-characterization of PP sets (given after Lemma 2).

Definition 5. *A set A is in* QOP *("quantum optimization polynomial time") if there exist two functions $f, g \in$ Qopt#QP and a function $h \in$ FP such that, for every x, $x \in A$ exactly when $\lfloor 2^{|h(x)|} f(x)\rfloor > \lfloor 2^{|h(x)|} g(x)\rfloor$. This h is called*

[12] The notation $\|A\|$ denotes the *operator norm* of A defined by $\sup\{\|A|\phi\rangle\|/\||\phi\rangle\|\}$, where sup is taken over all nonzero vectors $|\phi\rangle$. A square matrix A is *contractive* if $\|A\| \leq 1$.

[13] For any two function classes $\mathcal{F}$ and $\mathcal{G}$, let $\mathcal{F} \circ \mathcal{G}$ denote the class of functions $f \circ g$, where $f \in \mathcal{F}$ and $g \in \mathcal{G}$ and $f \circ g$ is the composition defined as $f \circ g(x) = f(g(x))$ for all x in the domain of g.

[14] A *qubit ensemble* is a sequence of qustrings with index set I. A qubit ensemble $\{|\phi_x\rangle\}_{x\in I}$ is called an *ℓ-qubit source* if each $|\phi_x\rangle$ is a qustring of size $\ell(|x|)$ and there exists a polynomial-time, $\tilde{\mathbb{C}}$-amplitude, well-formed, clean QTM that generates $|\phi_x\rangle$ on input x.

a selection function[15]. *Moreover, let* $\widehat{\mathrm{QOP}}$ *denote the subset of* QOP *with the extra condition that* $\lfloor 2^{|h(x)|} f(x) \rfloor \neq \lfloor 2^{|h(x)|} g(x) \rfloor$ *for all* x.

In the following lemma, we view an FPSPACE-function as a function from Σ^* to $\mathbb{D}$. Let $\mathbb{D}^{\Sigma^*}$ denote the class of all *total* functions from Σ^* to $\mathbb{D}$. In the proof, we use the relationship between quantum functions in Qopt#QP and the maximal eigenvalues of Hermitian, positive semi-definite, contractive matrices.

Lemma 3. $\mathrm{Opt\#QP} \subseteq \mathrm{Qopt\#QP} \Subset^e \#\mathrm{QP}^{\mathrm{QOP}} \Subset^e \mathrm{FPSPACE} \cap \mathbb{D}^{\Sigma^*}$.

Proof. The first inclusion follows from $\mathrm{Opt}(\mathrm{Qopt\#QP}) = \mathrm{Qopt\#QP}$.

For the second inclusion, assume that $f \in \mathrm{Qopt\#QP}$. Take a function $g \in \#\mathrm{QP}$ and a polynomial q such that $f(x) = \sup\{g(x, |\phi\rangle) \mid |\phi\rangle \in \Phi_{q(|x|)}\}$ for every x. Define the sets A_+ and A_- as follows: $A_+ = \{(x, y) \mid \lfloor 2^{p(|x|)+1} f(x) \rfloor \geq num(y)\}$ and $A_- = \{(x, y) \mid \lfloor 2^{p(|x|)+1} f(x) \rfloor \leq num(y)\}$, where $num(y)$ is the number n in $\mathbb{N}$ such that y is the nth string in the standard order (the empty string is the 0th string). Let $A = A_+ \oplus A_-$, where $\oplus$ is the disjoint union[16]. It is not difficult to show that $A \in \mathrm{QOP}$. Let p be any polynomial. By a standard binary search algorithm using A, we can find an approximation of $f(x)$ to within $2^{-p(|x|)}$ in polynomial time. Let h^A be the function computed by this algorithm with oracle A. Then, we have $|f(x) - h^A(x)| \leq 2^{-p(|x|)}$.

For the third inclusion, it suffices to show that $\mathrm{Qopt\#QP} \Subset^e \mathrm{FPSPACE} \cap \mathbb{D}^{\Sigma^*}$ since this implies $\mathrm{QOP} \subseteq \mathrm{PSPACE}$ together with the fact that $\#\mathrm{QP} \Subset^e \mathrm{FPSPACE} \cap \mathbb{D}^{\Sigma^*}$ (which follows from results in [21]).

Let f be any function in Qopt#QP. There exist a polynomial q and a sequence $\{V_x\}_{x \in \Sigma^*}$ such that, for every x, (i) V_x is a $2^{q(|x|)} \times 2^{q(|x|)}$ Hermitian, positive semi-definite, contractive matrix and (ii) $f(x) = \|V_x\|$. Note that each (s, t)-entry of V_x is approximated to within 2^{-m} deterministically using space polynomial in m, $|x|$, $|s|$, and $|t|$. Consider the characteristic polynomial $p(z) = \det(zI - V_x)$. This $p(z)$ has the form $p(z) = \sum_{i=0}^{2^{q(|x|)}} a_i z^i$. It is easy to see that each coefficient a_i can be approximated using polynomial space. Thus, we can also approximate each real root of $p(z)$ using polynomial space. Finally, we can find a good approximation of the maximal real root of $p(z)$. Thus, $f \overset{\in}{\sim}{}^p \mathrm{FPSPACE} \cap \mathbb{D}^{\Sigma^*}$. □

Theorem 2. *1. If* $\mathrm{EQP} = \mathrm{QOP}$, *then* $\mathrm{Qopt\#QP} \Subset^e \#\mathrm{QP}$.

2. If $\mathrm{Qopt\#QP} \Subset^e \#\mathrm{QP}$, *then* $\widehat{\mathrm{QOP}} = \mathrm{PP}$ *and* $\mathrm{QOP} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$, *where* $[\cdot]$ *stands for the number of queries on each input.*

Proof. 1) This is an immediate consequence of Lemma 3.

2) Assume that $\mathrm{Qopt\#QP} \Subset^e \#\mathrm{QP}$. Note that $\mathrm{PP} \subseteq \widehat{\mathrm{QOP}} \subseteq \mathrm{QOP}$. We first show that $\widehat{\mathrm{QOP}} \subseteq \mathrm{PP}$. Let A be any set in $\widehat{\mathrm{QOP}}$. There exists a polynomial p and functions f and g in Opt#QP such that, for every x, (i) if $x \in A$ then

[15] We can replace the value $|h(x)|$ by $p(|x|)$ for an appropriate polynomial p. This is seen by taking p that satisfies $|h(x)| \leq p(|x|)$ for all x and by replacing f and g, respectively, with $\hat{f}(x) = 2^{-p(|x|)+|h(x)|} f(x)$ and $\hat{g}(x) = 2^{-p(|x|)+|h(x)|} g(x)$.

[16] For any two sets A and B, $A \oplus B$ is the *disjoint union* of A and B defined by $A \oplus B = \{0x \mid x \in A\} \cup \{1x \mid x \in B\}$.

$\lfloor 2^{p(|x|)} f(x) \rfloor > \lfloor 2^{p(|x|)} g(x) \rfloor$ and (ii) if $x \notin A$ then $\lfloor 2^{p(|x|)} f(x) \rfloor < \lfloor 2^{p(|x|)} g(x) \rfloor$. Since $f \overset{\in}{\sim}{}^{p}\#\mathrm{QP}$ by our assumption, there exists a certain function $\hat{f}$ in #QP such that, for every x, $|f(x) - \hat{f}(x)| \leq 2^{-p(|x|)-1}$, which implies that $\lfloor 2^{p(|x|)} f(x) \rfloor = \lfloor 2^{p(|x|)} \hat{f}(x) \rfloor$. Similarly, we obtain $\hat{g}$ from g. Assume that $x \in A$. Then, we have $\lfloor 2^{p(|x|)} f(x) \rfloor > \lfloor 2^{p(|x|)} g(x) \rfloor$. This clearly implies $\hat{f}(x) > \hat{g}(x)$. Similarly, if $x \notin A$ then $\hat{f}(x) < \hat{g}(x)$. Therefore, $A = \{x \mid \hat{f}(x) > \hat{g}(x)\}$. From the comment after Lemma 2, A belongs to PP.

Next, we show that $\mathrm{QOP} \subseteq \mathrm{P}^{\#\mathrm{P}[1]}$. Assume that $A \in \mathrm{QOP}$. Similar to the previous argument, we can obtain $\hat{f}$ and $\hat{g}$ in #QP. For this $\hat{f}$, by [21], there exist two functions $\tilde{f} \in \mathrm{GapP}$ and $\ell \in \mathrm{FP}$ such that, for every x, $|\hat{f}(x) - \tilde{f}(x)/\ell(1^{|x|})| \leq 2^{-p(|x|)-1}$. Thus, $\lfloor 2^{p(|x|)} \hat{f}(x) \rfloor$ coincides with the first $p(|x|)$ bits (of the binary expansion) of $2^{p(|x|)} \tilde{f}(x)/\ell(1^{|x|})$. Similarly, we obtain $\tilde{g}$ and ℓ'. Without loss of generality, we can assume that $\ell = \ell'$. To determine whether x is in A, we first compute the value $\tilde{f}(x)$ and $\tilde{g}(x)$ and then compare the first $p(|x|)$ bits of the binary expansion of $2^{p(|x|)} \tilde{f}(x)/\ell(1^{|x|})$ and those of $2^{p(|x|)} \tilde{g}(x)/\ell(1^{|x|})$. This is done by making two queries to $\tilde{f}$ and $\tilde{g}$. It is easy to reduce these two queries to one single query to another #P-function. Thus, A belongs to $\mathrm{P}^{\#\mathrm{P}[1]}$. □

For any PSPACE-complete set A, we have $\mathrm{EQP}^A = \mathrm{QOP}^A$ since $\mathrm{P}^A = \mathrm{PSPACE}^A$. Thus, $\mathrm{Qopt\#QP}^A \Subset^p \#\mathrm{QP}^A$ since Theorem 2(1) relativizes. To see the relativized separation between Qopt#QP and #QP, we note that $\mathrm{BQP} \neq \mathrm{QMA}$ implies $\mathrm{Qopt\#QP} \not\Subset^p \#\mathrm{QP}$, where QMA[17] is a quantum version of Babai's MA. This result relativizes. It is also known that $\mathrm{NP}^A \not\subseteq \mathrm{BQP}^A$ for a certain oracle A [3,2] and that $\mathrm{NP}^B \subseteq \mathrm{QMA}^B$ for any oracle B. As a consequence, we obtain the desired result: $\mathrm{Qopt\#QP}^C \not\Subset^p \#\mathrm{QP}^C$ for a certain oracle C.

3 Universal Functions for Opt#QP and Qopt#QP

Based upon his metric reducibility between optimal cost functions for NP optimization problems, Krentel [11] introduced the notion of *complete functions* for the class OptP, which constitute the hardest functions in OptP. It is natural to consider a similar notion among quantum optimization problems. However, elements in Opt#QP and Qopt#QP are functions defined by well-formed $\tilde{\mathbb{C}}$-amplitude QTMs. Since there is no single QTM that *exactly* simulates all the other well-formed QTMs with $\tilde{\mathbb{C}}$-amplitudes, we need to relax the meaning of "completeness".

Following Deutsch's work [6], Bernstein and Vazirani [3] constructed a *universal* QTM that can approximately simulate any well-formed $\tilde{\mathbb{C}}$-amplitude QTM M for t steps with desired accuracy ϵ at the cost of polynomial slowdown. In other words, every QTM can be "approximately" reduced to one single QTM. We can generalize this notion of universality in the following fashion.

[17] A set A is in QMA if there exists a $f \in \mathrm{Qopt\#QP}$ such that, for every x, if $x \in A$ then $f(x) \geq 3/4$ and otherwise $f(x) \leq 1/4$ [9,19].

Definition 6. *1. Let f and g be any functions from Σ^* to $[0,1]$. The function f is* (polynomial-time) approximately reducible to g, *denoted $f \lessapprox^p g$, if there is a function $k \in \mathrm{FP}$ such that, for every x and $m \in \mathbb{N}^+$, $|f(x) - g(k(x01^m))| \leq 1/m$.*

2. Let $\mathcal{F}$ be any class of functions from $\{0,1\}^*$ to $[0,1]$. A function g from Σ^* to $[0,1]$ is universal *for $\mathcal{F}$ ($\mathcal{F}$-*universal*, in short) if (i) g is in $\mathcal{F}$ and (ii) every function $f \in \mathcal{F}$ is approximately reducible to g.*

Using the so-called Solovay-Kitaev theorem, we may replace the term $1/m$ in Definition 6 by 2^{-m}. This relation $\lessapprox^p$ is obviously reflexive and transitive. The importance of a universal function is given as follows. Let $\mathcal{F}$ and $\mathcal{G}$ be any two classes of functions from Σ^* to $[0,1]$. Assume that $\mathcal{G} \circ \mathrm{FP} \subseteq \mathcal{G}$ and let f be any $\mathcal{F}$-universal function. Then, $\mathcal{F} \Subset^p \mathcal{G}$ if and only if $f \overset{\in}{\approx}{}^p \mathcal{G}$. Note that most natural classes $\mathcal{G}$ satisfy this premise $\mathcal{G} \circ \mathrm{FP} \subseteq \mathcal{G}$.

Since many well-known quantum complexity classes are believed to lack complete problems, the notion of universality naturally provides *promise complete* problems for, e.g., BQP by posing appropriate restrictions on the acceptance probabilities of quantum functions in #QP.

Now, we define the Qtm Approximation Problem (QAP) as follows:

Qtm Approximation Problem: QAP
- instance: $\langle M, x, 1^t, 1^m \rangle$, where M is a $\tilde{\mathbb{C}}$-amplitudes well-formed QTM, t is in $\mathbb{N}$, and m is in $\mathbb{N}^+$.
- question: find the acceptance probability of M_U on input $\langle M, x, 1^t, 1^m \rangle$.

As Bernstein and Vazirani [3] demonstrated, every function f outputting the acceptance probability of a certain polynomial-time well-formed QTM with $\tilde{\mathbb{C}}$-amplitudes is approximately reducible to QAP. Thus, we obtain:

Proposition 2. *QAP is* #QP*-universal.*

We return to the quantum optimization class Opt#QP. In Section 2.1, we have shown that the Maximum Qtm Problem (MAXQTM) is in Opt#QP. Due to the universality of QAP for #QP, MAXQTM turns out to be universal for Opt#QP.

Theorem 3. *MAXQTM is* Opt#QP*-universal.*

Next, we recall the Maximum Circuit Eigenvalue Problem (MAXCIR) defined in Section 2.2. Earlier, Yao [22] demonstrated how to simulate a QTM by a family of quantum circuits. Thus, we can exactly simulate M_U on quantum input by a quantum circuit C on the same quantum input. Therefore, we conclude:

Theorem 4. *MAXCIR is* Qopt#QP*-universal.*

4 Complexity of Quantum Optimization Problems

This section briefly discusses relationships between quantum optimization classes and important complexity classes. The quantum optimization class Qopt#QP can "define" many complexity classes. For example, the class QMA is defined in

terms of Qopt#QP. Another example is the class EQMA[18], which is an error-free (or exact) version of QMA. A less trivial example is the class NQP [1].

Proposition 3. *A set A is in* NQP *if and only if there exists a function $f \in$* Qopt#QP *such that, for every x, (i) if $x \in A$ then $f(x) > 0$, and (ii) if $x \notin A$ then $f(x) = 0$.*

The proof of Proposition 3 comes from the following fact (obtained from the closure property (iv) in Section 2.2): for any function h in Qopt#QP, there exists a function $g \in$ #QP and a polynomial p such that, for all strings x and integers $m > 0$, $g(x01^m) \leq h^m(x) \leq 2^{p(|x|)}g(x01^m)$.

The quantum optimization class Qopt#QP also characterizes the counting class PP, where PP is known to contain the aforementioned quantum complexity classes. Moreover, PP is "robust" in the sense that it equals $\mathrm{PQP}_{\tilde{\mathbb{C}}}$ [21], a quantum interpretation of PP. We show that PP can be characterized by Qopt#QP together with GapQP.

Proposition 4. *Let A be any subset of $\{0,1\}^*$. The following statements are equivalent.*

1. *A is in* PP.
2. *For every polynomial q, there exist two functions $f \in$* Qopt#QP *and $g \in$* GapQP *such that, for every string x and integer m ($m \geq |x|$), (i) $g(x01^m) > 0$; (ii) $x \in A$ implies $(1 - 2^{-q(m)})g(x01^m) \leq f(x01^m) \leq g(x01^m)$; and (iii) $x \notin A$ implies $0 \leq f(x01^m) \leq 2^{-q(m)}g(x01^m)$.*

Proof. (1 implies 2.) Let A be any set in PP. The proof uses the following well-known characterization of PP by GapP-functions (see [13,15]): a set A is in PP if and only if, for every polynomial p, there exist two functions $f, g \in$ GapP such that, for every x and every $m \geq |x|$, (i) $g(x01^m) > 0$, (ii) $x \in A$ implies $(1 - 2^{-q(m)})g(x01^m) \leq f(x01^m) \leq g(x01^m)$, and (iii) $x \notin A$ implies $0 \leq f(x01^m) \leq 2^{-q(m)}g(x01^m)$.

Take $\tilde{g}$ and $\tilde{f}$ from GapQP and ℓ from FP such that $f(x) = \tilde{f}(x)\ell(1^{|x|})$ and $g(x) = \tilde{g}(x)\ell(1^{|x|})$ for all x [21]. We thus replace g and f in the above characterization by $\tilde{g}$ and $\tilde{f}$ and then make all the terms squared. Note that $\tilde{f}^2$ and $\tilde{g}^2$ belong to #QP [21] and thus, the both are in Qopt#QP. Since $(1 - 2^{-q(m)})^2 \geq 1 - 2^{-q(m)+1}$ and $(2^{-q(m)})^2 \leq 2^{-q(m)+1}$, we can obtain the desired result.

(2 implies 1.) Set $q(n) = n$ and assume that there exist two functions $f \in$ Qopt#QP and $g \in$ GapQP satisfying Statement 2. Since $g \in$ GapQP implies $g^2 \in$ #QP [21], we can assume without loss of generality that $g \in$ #QP. For each x, let $\hat{x} = x01^{|x|}$. The fact described after Proposition 3 guarantees the existence of a polynomial p and a function $h \in$ #QP satisfying that, for every x and $m > 1$, $h(\hat{x}01^m) \leq f^m(\hat{x}) \leq 2^{p(|x|)}h(\hat{x}01^m)$. For simplicity, assume that $p(n) \geq 2$ for all $n \in \mathbb{N}$. Let $g'(x) = 2^{-2p(|x|)}g^{p(|x|)}(\hat{x})$ and $h'(x) = h(\hat{x}01^{p(|x|)})$ for all x. It is easy to see that $g', h' \in$ #QP since $g, h \in$ #QP. In what follows, we want to show that, for all but finitely-many strings x, $x \in A$ if and only if $g'(x) < h'(x)$. This implies that A is indeed in PP.

[18] A set S is in EQMA if χ_S is in Qopt#QP, where χ_S is the characteristic function for S.

Fix x arbitrarily with $n = |x| \geq 4$. In the case where $x \in A$, we obtain: $2^{-\frac{p(n)}{n-2}} g^{p(n)}(\hat{x}) \leq (1-2^{-n})^{p(n)} g^{p(n)}(\hat{x}) \leq f^{p(n)}(\hat{x}) \leq 2^{p(n)} h(\hat{x}01^{p(n)})$ since $(1-2^{-n})^{p(n)} = ((1-2^{-n})^{n-2})^{\frac{p(n)}{n-2}}$ and $(1-2^{-n})^{n-2} \geq 1 - 2^{-n+(n-2)+1} = \frac{1}{2}$. This yields the inequality $2^{-p(n)(1+1/(n-2))} g^{p(n)}(\hat{x}) \leq h'(x)$, which further implies that $g'(x) < h'(x)$ since $n \geq 4$. Consider the other case where $x \notin A$. In this case, we obtain: $h'(x) = h(\hat{x}01^{p(n)}) \leq f^{p(n)}(\hat{x}) \leq (2^{-n})^{p(n)} g^{p(n)}(\hat{x}) \leq 2^{-np(n)} g^{p(n)}(\hat{x})$, which implies $h'(x) < 2^{-2p(n)} g^{p(n)}(\hat{x}) = g'(x)$ since $n > 2$. □

Consider complexity classes located between QMA and PP. Proposition 4 suggests an introduction of a quantum analogue of Li's APP[19] [13].

Definition 7. *A set A is in* AQMA *("amplified" QMA) if, for every polynomial q, there exist two functions $f \in$ Qopt#QP and $g \in$ GapQP such that, for every string x and integer m ($m \geq |x|$), i) $g(1^m) > 0$; ii) if $x \in A$ then $(1-2^{-q(m)})g(1^m) \leq f(x01^m) \leq g(1^m)$; and iii) if $x \notin A$ then $0 \leq f(x01^m) \leq 2^{-q(m)} g(1^m)$.*

Obviously, APP ∪ QMA ⊆ AQMA. Proposition 4 shows that AQMA ⊆ PP. The inclusion relationships among aforementioned complexity classes are summarized as follows:

$$\mathrm{APP} \cup \mathrm{QMA} \subseteq \mathrm{AQMA} \subseteq \mathrm{PP} \subseteq \widehat{\mathrm{QOP}} \subseteq \mathrm{QOP} \subseteq \mathrm{PSPACE}.$$

At the end of this section, we present lowness results. The sets with low information are known as *low sets*. Let $\mathcal{F}$ be any class (of functions or sets). A set A is *low for* $\mathcal{F}$ ($\mathcal{F}$*-low*, in short) if $\mathcal{F}^A = \mathcal{F}$. Let low-$\mathcal{F}$ denote the collection of all sets that are $\mathcal{F}$-low. An example of such a low set is low-#QP = low-GapQP = EQP [21]. The following proposition shows the complexity of the low sets for Qopt#QP and QOP.

Proposition 5. *1.* EQP ⊆ low-Opt#QP ⊆ low-Qopt#QP ⊆ EQMA.
2. low-Qopt#QP ⊆ low-QOP ⊆ QOP ∩ co-QOP.

The proof of Proposition 5 is straightforward and is left to the reader.

References

1. L. M. Adleman, J. DeMarrais, and M. A. Huang, Quantum computability, *SIAM J. Comput.* **26** (1997), 1524–1540.
2. C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, Strengths and weaknesses of quantum computing, *SIAM J. Comput.* **26**, 1510–1523, 1997.
3. E. Bernstein and U. Vazirani, Quantum complexity theory, *SIAM J. Comput.* **26**, 1411–1473, 1997.
4. H. Buhrman, J. Kadin, and T. Thierauf, Functions computable with nonadaptive queries to NP, *Theory of Computing Systems*, **31** (1998), 77–92.

[19] A set A is in APP ("amplified" PP) if, for every polynomial p, there exist two functions $f, g \in$ GapP such that, for every x and every $m \geq |x|$, (i) $g(1^m) > 0$; (ii) if $x \in A$ then $(1-2^{-q(m)})g(1^m) \leq f(x01^m) \leq g(1^m)$; and (iii) if $x \notin A$ then $0 \leq f(x01^m) \leq 2^{-q(m)} g(1^m)$ [13].

5. Z. Chen and S. Toda, On the complexity of computing optimal solutions, *International Journal of Foundations of Computer Science* **2** (1991), 207–220.
6. D. Deutsch, Quantum theory, the Church-Turing principle, and the universal quantum computer, *Proc. Roy. Soc. London*, A, **400** (1985), 97–117.
7. S. Fenner, L. Fortnow, and S. Kurtz, Gap-definable counting classes, *J. Comput. System Sci.* **48** (1994), 116–148.
8. F. Green, On the power of deterministic reductions to C$_=$P, *Mathematical Systems Theory* **26** (1993), 215–233.
9. A. Kitaev, "Quantum NP", Public Talk at AQIP'99: 2nd Workshop on Algorithms in Quantum Information Processing, DePaul University, 1999.
10. J. Köbler, U. Schöning, and J. Torán, On counting and approximation, *Acta Informatica*, **26** (1989), 363–379.
11. M. W. Krentel, The complexity of optimization problems, *J. Comput. System Sci.*, **36** (1988), 490–509.
12. M. W. Krentel, Generalizations of OptP to the polynomial hierarchy, *Theoretical Computer Science*, **97** (1992), 183–198.
13. L. Li, On the counting functions, Ph.D. thesis, Department of Computer Science, University of Chicago, 1993. Also see technical report TR 93-12.
14. M. Ogiwara and L. Hemachandra, A complexity theory for feasible closure properties, *J. Comput. System Sci.* **46** (1993), 295–325.
15. L. A. Hemaspaandra and M. Ogihara, *The Complexity Theory Companion*, Springer, 2002.
16. M. Ozawa and H. Nishimura, Local transition functions of quantum Turing machines, in *Proceedings of the 4th International Conference on Quantum Communication, Complexity, and Measurement*, 1999.
17. C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, 1998.
18. L. Valiant, The complexity of computing the permanent, *Theor. Comput. Sci.* **8** (1979), 189–201.
19. J. Watrous, Succinct quantum proofs for properties of finite groups, in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pp.537–546, 2000.
20. T. Yamakami, A foundation of programming a multi-tape quantum Turing machine, in Proceedings of the 24th International Symposium on Mathematical Foundation of Computer Science, Lecture Notes in Computer Science, Vol.1672, pp.430-441, 1999.
21. T.Yamakami, Analysis of quantum functions, in Proceedings of the 19th International Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, Vol.1738, pp.407-419, 1999.
22. A. C. Yao, Quantum circuit complexity, in *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pp.352–361, 1993.

Appendix: Quantum Turing Machines

A pure *quantum state* is a vector of unit norm in a Hilbert space (a complex vector space with a standard inner product). A *quantum bit* (qubit, in short) is a quantum state of a two-dimensional Hilbert space. We use the standard basis $\{|0\rangle, |1\rangle\}$ to express qubits. Any quantum state in a 2^n-dimensional Hilbert space is called a *quantum string* (qustring, in short) of size n.

Formally, a k-tape quantum Turing machine (QTM, in short) M is a six-tuple $(Q, \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_k, \Gamma_1 \times \Gamma_2 \times \cdots \times \Gamma_k, q_0, Q_f, \delta)$, where Q is a finite set of internal states including the initial state q_0 and a set Q_f of final states, each Σ_i is an input alphabet of tape i, each Γ_i is a tape alphabet of tape i including a blank symbol and Σ_i, and δ is a quantum transition function from $Q \times \Gamma_1 \times \cdots \times \Gamma_k$ to $\mathbb{C}^{Q \times \Gamma_1 \times \cdots \times \Gamma_k \times \{L,N,R\}^k}$. A multi-tape QTM is equipped with two-way infinite tapes, tape heads, and a finite-control unit, similar to a classical Turing machine. A QTM follows an instruction given by its transition function δ, which dictates the next move of the machine. The *running time* of M on input $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ is the minimal number t (if any) such that, at time t, all computation paths of M on input $\boldsymbol{x}$ reach final configurations (i.e., configurations with final states). We say that M *halts on input* $\boldsymbol{x}$ *in time* t if the running time of M on input $\boldsymbol{x}$ is defined and is exactly t.

The transition function δ is considered as an operator that transforms a superposition of configurations at time t to another superposition of configurations at time $t+1$. We call such an operator a *time-evolution operator* (or *matrix*). A QTM has *K-amplitudes* if all the entries of its time-evolution matrix are drawn from set K. A QTM is called *well-formed* if its time-evolution operator is unitary. A k-tape QTM M is *stationary* if all tape heads move back to the start cells, and M is in *normal form* if, for every $q \in Q_f$, there exists a series of directions $\boldsymbol{d} \in \{L, N, R\}^k$ such that $\delta(q, \boldsymbol{\sigma}) = |q_0\rangle|\boldsymbol{\sigma}\rangle|\boldsymbol{d}\rangle$ for all tape symbols $\boldsymbol{\sigma} \in \Gamma_1 \times \cdots \times \Gamma_k$. A QTM is called *clean* if it is stationary in normal form and all the tapes except the output tape are empty when it halts with one distinguished final state.

We say that a well-formed QTM M *accepts input* $|\phi\rangle$ *with probability* α if M halts in a final configuration in which, when observed, bit 1 is found in the start cell of the output tape with probability α. In this case, we also say that M *rejects input* $|\phi\rangle$ *with probability* $1 - \alpha$.

An *oracle QTM* is a QTM equipped with an extra query tape and two distinguished states, a *pre-query* and *post-query* states. Let A be an oracle. When the machine enters a pre-query state, the string written in the query tape, say $|x\rangle|b\rangle$, where $x \in \Sigma^*$ and $b \in \{0, 1\}$, is changed into $|x\rangle|b \oplus A(x)\rangle$ in a single step and the machine enters a post-query state, where $\oplus$ denotes the exclusive OR.

An Analysis of Absorbing Times of Quantum Walks

Tomohiro Yamasaki[1], Hirotada Kobayashi[1,3], and Hiroshi Imai[2,3]

[1] Department of Information Science,
Graduate School of Science,
The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
{yamasaki, hirotada}@is.s.u-tokyo.ac.jp
[2] Department of Computer Science,
Graduate School of Information Science and Technology,
The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
imai@is.s.u-tokyo.ac.jp
[3] Quantum Computation and Information Project,
Exploratory Research for Advanced Technology,
Japan Science and Technology Corporation,
5-28-3 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

Abstract. Quantum walks are expected to provide useful algorithmic tools for quantum computation. This paper introduces absorbing probability and time of quantum walks and gives both numerical simulation results and theoretical analyses on Hadamard walks on the line and symmetric walks on the hypercube from the viewpoint of absorbing probability and time. Our results may be suggestive in constructing efficient quantum algorithms for combinatorial problems such as SAT.

1 Introduction

Random walks, or often called Markov chains, on graphs have found a number of applications in various fields, not only in natural science such as physical systems and mathematical modeling of life phenomena but also in social science such as financial systems. Also in computer science, random walks have been applied to various problems such as 2-SAT, approximation of the permanent [8,9], and estimation of the volume of convex bodies [5]. Schöning's elegant algorithm for 3-SAT [12] and its improvement [7] are also based on classical random walks. Moreover, one of the most advantageous points of classical random walks as a useful algorithmic tool is that they use only simple local transitions to obtain global properties of the instance.

Thus, it is natural to consider quantum generalization of classical random walks, which may be very useful in constructing efficient quantum algorithms, for which only a few general algorithmic tools have been developed, including Fourier sampling and amplitude amplification. There have been considered two types of quantum walks: one is a discrete-time walk discussed by Watrous [13], Ambainis,

C.S. Calude et al. (Eds.): UMC 2002, LNCS 2509, pp. 315–329, 2002.

Bach, Nayak, Vishwanath, and Watrous [2], Aharonov, Ambainis, Kempe, and Vazirani [1], and Moore and Russell [11], and the other is a continuous-time one by Childs, Farhi, and Gutmann [4] and Moore and Russell [11]. All of these studies demonstrate that the behaviors of quantum walks are quite different from classical ones. In the theory of classical random walks, one of the important measures is the *mixing time*, which is the time necessary to have the probability distribution of the particle be sufficiently close to the stationary distribution. Unfortunately, Aharonov, Ambainis, Kempe, and Vazirani [1] showed a rather negative result that discrete-time quantum walks can be at most polynomially faster than classical ones in view of mixing time. As for continuous-time quantum walks, Childs, Farhi, and Gutmann [4] claim that quantum walks propagate exponentially faster than classical ones. However, it is unclear whether their model of quantum walks is applicable to problems in computer science.

This paper focuses on the discrete-time type of quantum walks and introduces two new criteria for propagation speed of discrete-time quantum walks: *absorbing probability* and *absorbing time*. A number of properties of quantum walks are investigated in view of these criteria. In particular, the behaviors of Hadamard walks on the line and symmetric walks on the hypercube are discussed through the results of computational simulation experiments. In our simulations, quantum walks appear exponentially faster than classical ones in absorbing time under certain conditions. Since any SAT instance can be naturally treated by a symmetric walk on some hypercube, our result may suggest a possibility of non-trivial quantum algorithm for SAT. Several theoretical analyses are also given on classical and quantum symmetric walks on the hypercube.

The remainder of this paper is organized as follows. Section 2 reviews the formal definition of (discrete-time) quantum walks, and introduces new criteria for propagation speed of quantum walks. Section 3 and Section 4 deal with a number of numerical simulations on Hadamard walks on the line and symmetric walks on the hypercube, respectively. Finally, in Section 5, several theoretical analyses are given on symmetric walks on the hypercube.

2 Definitions

In this section, we give a formal definition of quantum walks. In direct analogy to classical random walks, one may try to define quantum walks as follows: at each step, the particle moves in all directions with equal amplitudes. However, such a walk is impossible since the evolution of whole quantum system would not be always unitary. As mentioned in the previous section, there exist two types of quantum generalizations of classical random walks: discrete-time quantum walks and continuous-time quantum walks. Here we only give a definition of discrete-time ones, which this paper treats. We also introduce new criteria of absorbing probability and absorbing time for propagation speed of quantum walks.

2.1 Discrete-Time Walks

We review the definition of discrete-time quantum walks on graphs according to [1].

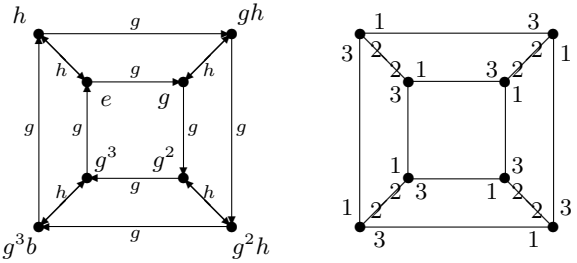


Fig. 1. An example of a Cayley graph and its labeling ($g^4 = e, h^2 = e, g \circ h = h \circ g$).

Let $G = (V, E)$ be a graph where V is a set of vertices and E is a set of edges. For theoretical convenience, we assume that G is d-regular (i.e. each vertex of the graph G is connected to exactly d edges). For each vertex $v \in V$, label each bidirectional edge connected to this vertex with a number between 1 and d such that, for each $a \in \{1, \ldots, d\}$, the directed edges labeled a form a permutation. In other words, for each vertex $v \in V$, not only label each outgoing edge $(v, w) \in E$ with a number between 1 and d but also label each incoming edge $(w, v) \in E$ with a number between 1 and d, where w is a vertex in V and $w \neq v$. In the case that G is a Cayley graph, the labeling of a directed edge simply corresponds to the generator associated with the edge. An example of a Cayley graph and its labeling are in Figure 1.

Let $\mathcal{H}_V$ be the Hilbert space spanned by $\{|v\rangle \mid v \in V\}$, and let $\mathcal{H}_A$ be the auxiliary Hilbert space spanned by $\{|a\rangle \mid 1 \leq a \leq d\}$. We think of this auxiliary Hilbert space as a "coin space".

Definition 1 ([1]). *Let C be a unitary operator on $\mathcal{H}_A$ which we think of as a "coin-tossing operator", and let a shift operator S on $\mathcal{H}_A \otimes \mathcal{H}_V$ be defined as $S|a, v\rangle = |a, w\rangle$ where w is the ath neighbor of vertex v. One step of transition of the quantum walk on the graph G is defined by a unitary operator $W = S(C \otimes I_V)$ where I_V is the identity operator on $\mathcal{H}_V$. We call such a walk a discrete-time quantum walk.*

More generally, we relax the restriction on the exact form of the quantum walk operator W and define general quantum walks such that W respects only the structure of the graph G. In other words, we require that, in the superposition $W|a, v\rangle$ for each $a \in \{1, \ldots, d\}$ and $v \in V$, only basis states $|a', v'\rangle$ for $a' \in \{1, \ldots, d\}, v' \in \text{neighbor}(v) \cup \{v\}$ have non-zero amplitudes, where neighbor(v) is the set of vertices adjacent to v. Thus the graph G does not need to be d-regular and the particle at v in the quantum walk moves to one of the vertices adjacent to v or stays at v in one step.

2.2 Criteria of Propagation Speed

One of the properties of random or quantum walks we investigate is how fast they spread over a graph. In order to evaluate it, a criterion "*mixing time*" has been

considered traditionally. However, Aharonov, Ambainis, Kempe, and Vazirani [1] showed that, in view of mixing time, discrete-time quantum walks can be at most polynomially faster than classical ones. To seek the possibility of quantum walks being advantageous to classical ones, this paper introduces another two new criteria of propagation speed of quantum walks: *absorbing probability* and *absorbing time*. Let us consider the set of *absorbing vertices* such that the particle is absorbed if it reaches a vertex in this set. This is done by considering a measurement that is described by the projection operators over $\mathcal{H}_A \otimes \mathcal{H}_V$: $P = I_A \otimes \sum_{v \in \mathrm{absorb}(V)} |v\rangle\langle v|$ and $P' = I_A \otimes \sum_{v \notin \mathrm{absorb}(V)} |v\rangle\langle v|$, where $\mathrm{absorb}(V) \subset V$ is the set of the absorbing vertices and I_A is the identity operator over $\mathcal{H}_A$.

By using density operators, one step of transition of a discrete-time quantum walk can be described by a completely positive (CP) map Λ as follows:

$$\Lambda : \rho \mapsto \rho' = IP\rho PI + WP'\rho P'W^\dagger,$$

where ρ and ρ' are density operators over the entire system of the quantum walk, W is a unitary operator corresponding to one step of transition of the quantum walk, and I is the identity operator on $\mathcal{H}_A \otimes \mathcal{H}_V$.

Now, we give a formal definition of absorbing probability.

Definition 2. *Absorbing probability of a quantum walk is the probability that the particle which starts at the initial vertex eventually reaches one of the absorbing vertices. More formally, it is defined as*

$$Prob = \sum_{t=0}^{\infty} p(t),$$

where $p(t)$ is the probability that the particle reaches one of the absorbing vertices at time t for the first time.

Let $|\psi_0\rangle$ be the initial state of the system, and let $\rho_0 = |\psi_0\rangle\langle\psi_0|$. Then $p(t) = \mathrm{tr}(\Lambda^t(\rho_0)P) - \mathrm{tr}(\Lambda^{t-1}(\rho_0)P)$ since $\mathrm{tr}(\Lambda^t(\rho_0)P)$ and $\mathrm{tr}(\Lambda^{t-1}(\rho_0)P)$ are the probabilities that the particle is absorbed by the time t and $t-1$, respectively. Here we allowed a little abuse of a notation Λ such that $\Lambda^2(\rho) = \Lambda(\Lambda(\rho))$, $\Lambda^3(\rho) = \Lambda(\Lambda^2(\rho))$, $\ldots$, $\Lambda^t(\rho) = \Lambda(\Lambda^{t-1}(\rho))$, and so on.

Next, we give a formal definition of *nominal absorbing time*.

Definition 3. *Nominal absorbing time of a quantum walk is the expected time to have the particle which starts at the initial vertex eventually reach one of the absorbing vertices. More formally, it is defined as*

$$Time_{\mathrm{nom}} = \sum_{t=0}^{\infty} tp(t),$$

where $p(t)$ is the probability that the particle reaches one of the absorbing vertices at time t for the first time.

Even though nominal absorbing time is bounded polynomial in the size of the graph instance, the quantum walk may not be efficient from a viewpoint of computational complexity if absorbing probability is exponentially small, because the smaller absorbing probability is, the more times quantum walks must

be repeated so that the particle is absorbed. Thus we cannot regard nominal absorbing time as a criterion of how fast a quantum walk spreads over a graph. Instead, we use the following *real absorbing time* as a criterion of the propagation speed of quantum walks.

Definition 4. *Real absorbing time of a quantum walk is defined as*

$$Time_{\text{real}} = \frac{Time_{\text{nom}}}{Prob} = \frac{\sum_{t=0}^{\infty} tp(t)}{\sum_{t=0}^{\infty} p(t)},$$

where $p(t)$ is the probability that the particle reaches one of the absorbing vertices at time t for the first time.

In what follows, we may refer to this real absorbing time as absorbing time in short, when it is not confusing.

3 Hadamard Walks on the Line

3.1 The Model

We start with the model of discrete-time quantum walks on the line, which was introduced and discussed by Ambainis, Bach, Nayak, Vishwanath, and Watrous [2]. Note that the line can be regarded as a Cayley graph of the Abelian group $\mathbb{Z}$ with generators $+1$ and -1, denoted by "R" and "L", respectively. Thus, the shift operator S is defined as

$$\begin{aligned} S|\mathrm{R}, v\rangle &= |\mathrm{R}, v+1\rangle, \\ S|\mathrm{L}, v\rangle &= |\mathrm{L}, v-1\rangle, \end{aligned}$$

where $v \in \mathbb{Z}$ is the vertex at which the particle is located.

Recall that the particle doing a classical random walk determines a direction to move randomly at each step. In the quantum case, therefore, it is quite natural to set a coin-tossing operator C as the Hadamard operator

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

We call this quantum walk an *Hadamard walk on the line*. A number of properties were shown in [2] on this model, including several different aspects of Hadamard walks from their classical counterparts.

This paper considers the following process for some fixed m [14], which is slightly different from the model above.

1. Initialize the system in the state $|\mathrm{R}, 0\rangle$, that is, let the particle be located at vertex 0 at time 0, with the chirality being R.
2. a) Apply $W = S(H \otimes I_{\mathbb{Z}})$.
   b) Measure the system according to $\{\Pi_m, \Pi'_m\}$ such that $\Pi_m = I_2 \otimes |m\rangle\langle m|$ and $\Pi'_m = I_2 \otimes (I_{\mathbb{Z}} - |m\rangle\langle m|)$, where I_2 is the two-dimensional identity operator (i.e. measure the system to observe whether the particle is located at the vertex m or not).
3. If the particle is observed to be located at the vertex m after the measurement, the process terminates, otherwise it repeats Step 2.

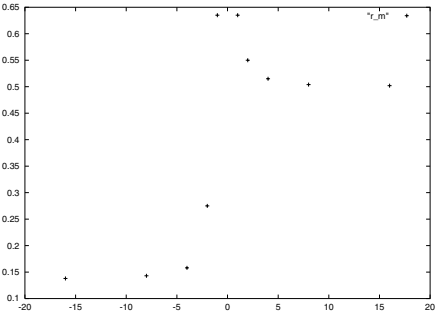


Fig. 2. Absorbing probability r_m for Hadamard walks.

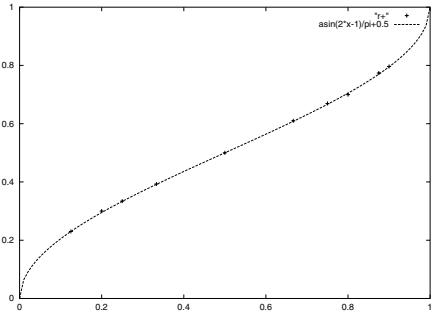


Fig. 3. Relation between p and $\lim_{m\to+\infty} r_m$ for generalized Hadamard walks.

3.2 Numerical Simulations

Let r_m be absorbing probability that the particle is eventually absorbed by the boundary at the vertex m. Figure 2 illustrates the relation between m (in x-axis) and r_m (in y-axis). From Figure 2, we conjecture that $\lim_{m\to\infty} r_m = 1/2$.

Next, let us consider a more general model of Hadamard walks whose coin-tossing operator is defined as the matrix

$$H_p = \begin{pmatrix} \sqrt{p} & \sqrt{1-p} \\ \sqrt{1-p} & -\sqrt{p} \end{pmatrix},$$

instead of H. Figure 3 illustrates the relation between p (in x-axis) and $\lim_{n\to\infty} r_m$ (in y-axis) for these generalized Hadamard walks. Although we have not yet derived a closed form for r_m, we conjecture the following.

Conjecture 1. For a generalized Hadamard walk with a coin-tossing operator H_p,

$$\lim_{m\to\infty} r_m = \frac{\sin^{-1}(2p-1)}{\pi} + \frac{1}{2}.$$

Remark. This conjecture was proved very recently by Bach, Coppersmith, Goldschen, Joynt, and Watrous [3].

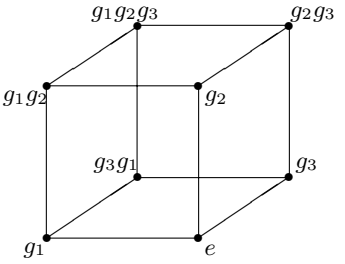


Fig. 4. The 3-dimensional hypercube. Each vertex can be regarded as a binary representation of length 3 (e.g.$g_1 \simeq (100)_2$ or $g_2g_3 \simeq (011)_2$).

4 Symmetric Walks on the n-Dimensional Hypercube

4.1 The Model

Next we consider the discrete-time quantum walks on the graph G of the n-dimensional hypercube, which was introduced and discussed by Moore and Russell [11]. Note that the n-dimensional hypercube can be regarded as a Cayley graph of the Abelian group $\mathbb{Z}_2^n$ with generators $g_1, g_2, \ldots, g_n$ such that $g_1^2 = g_2^2 = \cdots = g_n^2 = e$. The shift operator S is defined as $S|a, v\rangle = |a, v \circ g_a\rangle$, where $a \in \{1, \ldots, n\}$ is a label and $v \in \mathbb{Z}_2^n$ is a vertex at which the particle is located.

A coin-tossing operator C is set as the Grover's diffusion operator [6]:

$$D = \begin{pmatrix} \frac{2}{n} - 1 & \frac{2}{n} & \cdots & \frac{2}{n} \\ \frac{2}{n} & \frac{2}{n} - 1 & \cdots & \frac{2}{n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{n} & \frac{2}{n} & \cdots & \frac{2}{n} - 1 \end{pmatrix}.$$

Notice that coin-tossing operators in this model should obey the permutation symmetry of the hypercube. Among such operators, the Grover's diffusion operator is the one farthest away from the identity operator. We call this quantum walk a *symmetric walk on the n-dimensional hypercube*. A number of properties were shown in [11] on the mixing time of this model in comparison with the classical case.

The process of this quantum walk is defined as follows:

1. Initialize the system in the state $|1, 0\rangle$. That is, let the particle be located at the vertex 0 at time 0, with the labeling being 1.
2. For every chosen number t of steps, apply W^t to the system, where $W = S(D \otimes I_{\mathbb{Z}_2^n})$, and then observe the location of the particle.

4.2 Numerical Simulations

Figure 5 illustrates the relation between the dimension n of the hypercube (in x-axis) and absorbing time of quantum walks (in y-axis). One can see that, if the

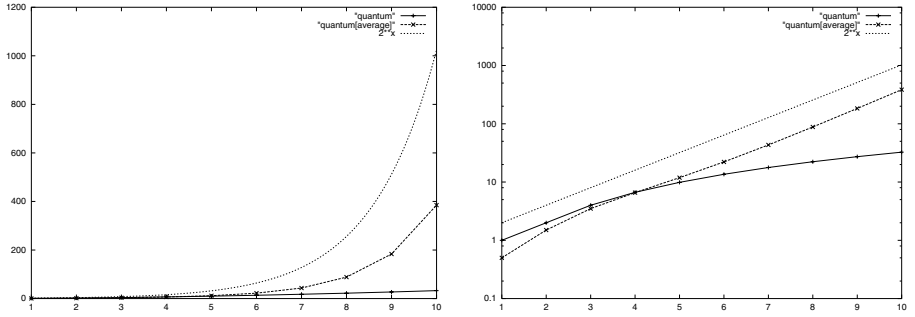


Fig. 5. Absorbing time of quantum symmetric walks on the n-dimensional hypercube. The solid line "quantum" is the case that absorbing vertex is located at $(1, 1, \ldots, 1)$. The dashed line "quantum[average]" is the case that absorbing vertex is located at random. The dotted line represents the function 2^n.

Table 1. Absorbing time and absorbing probability of a quantum symmetric walk on the 8-dimensional hypercube.

Hamming distance	Absorbing time (real)		Absorbing time (nominal)		Absorbing probability
0	0.0000	=	0.0000	/	1.0000
1	29.0000	=	29.0000	/	1.0000
2	59.0000	=	16.8571	/	0.2857
3	97.2444	=	13.8921	/	0.1429
4	115.5175	=	13.2020	/	0.1143
5	95.7844	=	13.6835	/	0.1429
6	56.3111	=	16.0889	/	0.2857
7	26.5603	=	26.5603	/	1.0000
8	22.3137	=	22.3137	/	1.0000

absorbing vertex is located at random, absorbing time averaged over all choices of the absorbing vertex increases exponentially with respect to n. This is similar to the classical case for which the absorbing time is theoretically analyzed in the next section. However, if the absorbing vertex is located at $(1, 1, \ldots, 1)$, absorbing time seems to increase polynomially (quadratically, more precisely) with respect to n. This may suggest a striking contrast between quantum and classical symmetric walks on the n-dimensional hypercube that propagation between a particular pair of vertices is *exponentially* faster in the quantum case.

Table 1 shows the relation between (nominal and real) absorbing time of the quantum walk on the 8-dimensional hypercube and Hamming distance between the initial vertex and the absorbing vertex. One can see the following:

– absorbing probability is large if the absorbing vertex is located near the initial vertex or is located far from the initial vertex, while it is small if the absorbing vertex is located in the "middle" of the hypercube relative to the initial vertex,

- nominal absorbing time has values of almost the same order except for the trivial case that Hamming distance is 0,
- nominal absorbing time is small, but real absorbing time is large, in the case of small absorbing probability.

In the classical case, the shorter Hamming distance between the initial vertex and the absorbing vertex is, the sooner the particle reaches the absorbing vertex, which meets our intuition. In the quantum case, however, our simulation result above is quite counter-intuitive.

From our simulation results illustrated by Figure 5 and Table 1, we conjecture the following for the quantum case.

Conjecture 2. Absorbing probability of quantum walks on the n-dimensional hypercube is $\min(1, \frac{n}{{}_nC_i})$, where i represents Hamming distance between the initial vertex and the absorbing vertex.

Conjecture 3. Nominal absorbing time of quantum walks on the n-dimensional hypercube is $O(n^2)$ independent of the location of the absorbing vertex, except for the trivial case that the initial vertex is the absorbing vertex.

Conjecture 4. Real absorbing time of quantum walks on the n-dimensional hypercube is $\frac{n^2-n+2}{2}$ if Hamming distance between the initial vertex and the absorbing vertex is 1, $O(n^2)$ if Hamming distance is n, and $\Theta(2^n)$ if Hamming distance is close to $\frac{n}{2}$.

Remark. Kempe [10] recently showed that real absorbing time of quantum walks on the n-dimensional hypercube is bounded polynomial in n if Hamming distance between the initial vertex and the absorbing vertex is n.

5 Theoretical Analyses of Symmetric Walks

5.1 Classical Symmetric Walks

First we state a property on the absorbing time of classical symmetric walks on the n-dimensional hypercube.

Proposition 1. *The absorbing time of classical symmetric walks on the n-dimensional hypercube is $\Theta(2^n)$, independent of the location of the absorbing vertex, except for the trivial case that the initial vertex is the absorbing vertex.*

Proof. Recall that the particle doing a classical random walk determines a direction to move randomly at each step. Thus this walk obeys the permutation symmetry of the hypercube and the vertices of the hypercube can be grouped in sets indexed by $i \in \{0, \ldots, n\}$, each of which is a set of vertices whose Hamming distance from the initial vertex is i.

Let s_i be absorbing time for the case that Hamming distance between the initial vertex and the absorbing vertex is i. Then, the following equations are satisfied:

$$\begin{array}{rl} s_0 &= 0, \\ s_1 &= \frac{1}{n}s_0 + \frac{n-1}{n}s_2 + 1, \\ s_2 &= \frac{2}{n}s_1 + \frac{n-2}{n}s_3 + 1, \\ &\vdots \\ s_{n-1} &= \frac{n-1}{n}s_{n-2} + \frac{1}{n}s_n + 1, \\ s_n &= s_{n-1} + 1, \end{array} \iff A \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{n-1} \\ s_n \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix},$$

where

$$A = (a_{ij}) = \begin{pmatrix} 1 & 0 & & & & & \\ -\frac{1}{n} & 1 & -\frac{n-1}{n} & & & & \\ & -\frac{2}{n} & 1 & -\frac{n-2}{n} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -\frac{n-1}{n} & 1 & -\frac{1}{n} \\ & & & & & -1 & 1 \end{pmatrix}.$$

Let a matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} 1 & \text{if } j = 1, \\ 0 & \text{if } i = 1, j \geq 2, \\ {}_nC_{j-1} \sum_{l=0}^{\min(i,j)-2} \frac{1}{{}_{n-1}C_l} & \text{if } i, j \geq 2, \end{cases}$$

and let $c_{ik} = \sum_{j=1}^{n+1} a_{ij} b_{jk}$. After some calculations, we have the following.

- In the case $i = 1$, $c_{ik} = b_{1k} = 1$ if $k = 1$, otherwise $c_{ik} = 0$.
- In the case $i = n+1$, $c_{ik} = -b_{nk} + b_{n+1,k} = 1$ if $k = n+1$, otherwise $c_{ik} = 0$.
- In the case $2 \leq i \leq n$, $c_{ik} = -\frac{i-1}{n} b_{i-1,k} + b_{ik} - \frac{n-i+1}{n} b_{i+1,k} = 1$ if $k = i$, otherwise $c_{ik} = 0$.

Therefore, AB is the identity matrix, and thus $B = A^{-1}$. It follows that

$$s_{i-1} = \sum_{j=2}^{n+1} b_{ij} = \begin{cases} 0 & \text{if } i = 1, \\ \sum_{j=2}^{n+1} \left({}_nC_{j-1} \sum_{l=0}^{\min(i,j)-2} \frac{1}{{}_{n-1}C_l} \right) & \text{if } i \geq 2. \end{cases}$$

From this, it is obvious that s_i increases monotonously with respect to i. For $i \geq 1$, we have

$$2^n - 1 = \sum_{j=2}^{n+1} {}_nC_{j-1} \leq s_i < 3 \sum_{j=2}^{n+1} {}_nC_{j-1} = 3(2^n - 1),$$

since $1 \leq \sum_{l=0}^{\min(i,j)-2} \frac{1}{{}_{n-1}C_l} < 3$. Thus we have the assertion. □

5.2 Quantum Symmetric Walks

In our simulation results in the previous section, quantum symmetric walks on the hypercube behave in a quite different manner from classical ones. In particular, it is remarkable that in the quantum case propagation between a particular pair of vertices seems exponentially faster than in the classical case. Here we try to describe absorbing time of quantum symmetric walks on the n-dimensional hypercube as a function of n.

The following lemma states that we do not need to keep track of the amplitudes of all basis states for symmetric quantum walks on the hypercube.

Lemma 1. *For a symmetric quantum walk on the n-dimensional hypercube with the initial vertex o, define sets $\{F_i\}$ and $\{B_i\}$ as follows:*

$$F_i = \big\{(a,v) \mid \mathrm{Ham}(v,o) = i,\ \mathrm{Ham}(v \circ g_a, o) = i+1\big\},$$
$$B_i = \big\{(a,v) \mid \mathrm{Ham}(v,o) = i,\ \mathrm{Ham}(v \circ g_a, o) = i-1\big\},$$

where $\mathrm{Ham}(x,y)$ denotes the Hamming distance between x and y. Let two unit vectors $|f_i\rangle$ and $|b_i\rangle$ in $\mathcal{H}_A \otimes \mathcal{H}_V$ be defined by

$$|f_i\rangle = \frac{1}{\sqrt{|F_i|}} \sum_{(a,v)\in F_i} |a,v\rangle \quad \text{for } i = 0,\dots,n-1,$$
$$|b_i\rangle = \frac{1}{\sqrt{|B_i|}} \sum_{(a,v)\in B_i} |a,v\rangle \quad \text{for } i = 1,\dots,n.$$

Then, a transition step of symmetric quantum walks is given by the following $2n \times 2n$ unitary matrix:

$$U_n = \begin{pmatrix}
0 & -\frac{n-2}{n} & \frac{\sqrt{4n-4}}{n} & & & & & & & \\
1 & 0 & 0 & & & & & & & \\
0 & 0 & 0 & -\frac{n-4}{n} & \frac{\sqrt{8n-16}}{n} & & & & & \\
 & \frac{\sqrt{4n-4}}{n} & \frac{n-2}{n} & 0 & 0 & & & & & \\
 & & & 0 & 0 & -\frac{n-6}{n} & \ddots & & & \\
 & & & \frac{\sqrt{8n-16}}{n} & \frac{n-4}{n} & \ddots & \ddots & & & \\
 & & & & & \ddots & \ddots & \frac{n-2}{n} & \frac{\sqrt{4n-4}}{n} & \\
 & & & & & \ddots & -\frac{n-4}{n} & 0 & 0 & 0 \\
 & & & & & & & 0 & 0 & 1 \\
 & & & & & & & \frac{\sqrt{4n-4}}{n} & -\frac{n-2}{n} & 0
\end{pmatrix}$$

with the order of bases $|f_0\rangle, |b_1\rangle, |f_1\rangle, \dots, |b_{n-1}\rangle, |f_{n-1}\rangle, |b_n\rangle$.

Proof. First, we calculate $|F_i|$ and $|B_i|$. The number of vertices v satisfying $\mathrm{Ham}(v,o) = i$ is ${}_nC_i$, and for such v, the number of labels a satisfying $\mathrm{Ham}(v \circ g_a, o) = i+1$ is $n-i$. Therefore, the number of basis vectors in F_i is $(n-i){}_nC_i$. Similarly, the number of basis vectors in B_i is $i\,{}_nC_i$.

Now, after applying a coin-tossing operator (which is the Grover's diffusion operator) to $|f_i\rangle$, we have

$$\begin{aligned}(C \otimes I)|f_i\rangle &= \frac{1}{\sqrt{|F_i|}} \sum_{(a,v)\in F_i} \left(\frac{2}{n} \sum_{1\le b\le n} |b,v\rangle - |a,v\rangle \right) \\ &= \frac{1}{\sqrt{|F_i|}} \left(\frac{2(n-i)}{n} \sum_{(a,v)\in B_i} |a,v\rangle + \frac{n-2i}{n} \sum_{(a,v)\in F_i} |a,v\rangle \right).\end{aligned}$$

Then, the shift operator is applied to this vector to have

$$\begin{aligned}S(C \otimes I)|f_i\rangle &= \frac{1}{\sqrt{|F_i|}} \left(\frac{2(n-i)}{n} \sum_{(a,v)\in F_{i-1}} |a,v\rangle + \frac{n-2i}{n} \sum_{(a,v)\in B_{i+1}} |a,v\rangle \right) \\ &= \sqrt{\frac{|F_{i-1}|}{|F_i|}} \cdot \frac{2(n-i)}{n} |f_{i-1}\rangle + \sqrt{\frac{|B_{i+1}|}{|F_i|}} \cdot \frac{n-2i}{n} |b_{i+1}\rangle \\ &= \frac{\sqrt{4ni-4i^2}}{n} |f_{i-1}\rangle + \frac{n-2i}{n} |b_{i+1}\rangle. \qquad (1)\end{aligned}$$

Similarly, the coin-tossing operator and shift operator are applied in sequence to the state $|b_i\rangle$ to have

$$S(C \otimes I)|b_i\rangle = -\frac{n-2i}{n} |f_{i-1}\rangle + \frac{\sqrt{4ni-4i^2}}{n} |b_{i+1}\rangle. \qquad (2)$$

The lemma holds immediately from (1) and (2). □

The following is immediate from Lemma 1.

Corollary 1. *One step of transition of symmetric quantum walks on the n-dimensional hypercube with the absorbing vertex located at $(1,1,\ldots,1)$ is described by a CP map $\Lambda : \rho_t \mapsto \rho_{t+1} = U_n P'_n \rho_t P'_n U_n^\dagger + I_n P_n \rho_t P_n I_n$, where ρ_t and ρ_{t+1} are density operators of the system at time t and $t+1$, respectively, I_n is the $2n \times 2n$ identity matrix, P_n is a $2n \times 2n$ projection matrix whose $(2n, 2n)$-element is 1 and all the others are 0, and $P'_n = I_n - P_n$.*

By this corollary, we can do numerical simulations for much larger n, say $n = 500$. Figure 6 illustrates the relation between the dimension n of the hypercube (in x-axis) and absorbing time of quantum walks (in y-axis) for larger n. One can see that absorbing time is close to $1.25n^{1.5}$.

In the following, we focus on the case where the absorbing vertex is at $(1,1,\ldots,1)$.

Since both absorbing probability and nominal absorbing time are defined as power series, and time evolution of the quantum walk is described by a CP map (hence by a linear operator), it is significant to investigate the properties of the matrix corresponding to time evolution operator and its eigenvalues in order to study behaviors of absorbing time with respect to the dimension n. First, we prove the following lemma.

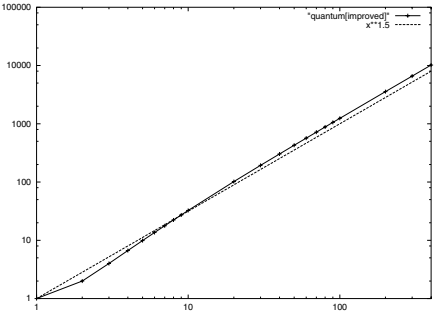


Fig. 6. Absorbing time of quantum walks on higher dimensional hypercubes with the absorbing vertex located at $(1, 1, \ldots, 1)$. The dotted line represents the function $n^{1.5}$.

Lemma 2. *Every eigenvalue of* $U_n P'_n$ *has its absolute value of less then* 1*, where* $P'_n = I_n - P_n$*.*

Proof. It is obvious that every eigenvalue of $U_n P'_n$ has its absolute value of at most 1.

Suppose that there exists an eigenvalue λ of $U_n P'_n$ such that $|\lambda| = 1$. Let $\boldsymbol{e}$ be the eigenvector of $U_n P'_n$ corresponding to λ. Then the $2n$th element of $\boldsymbol{e}$ is 0 and $P'_n \boldsymbol{e} = \boldsymbol{e}$, since applying the projection operator P'_n changes the $2n$th element of $\boldsymbol{e}$ into 0 and $|\lambda| = 1$. Thus we have $U_n \boldsymbol{e} = \lambda \boldsymbol{e}$, which implies that $\boldsymbol{e}$ is an eigenvector of U_n. However, it is easy to check by calculation that, in every eigenvector of U_n, the $2n$th element of it is not 0, which is a contradiction. □

By using Lemma 2, exact values of absorbing probability and absorbing time can be described for quantum walks on the n-dimensional hypercube with the absorbing vertex located at $(1, 1, \ldots, 1)$.

Proposition 2. *Let* X_n *be the* $2n \times 2n$ *symmetric matrix satisfying* $X_n - U_n P'_n X_n P'_n U_n^\dagger = \rho_0$*, where* $P'_n = I_n - P_n$ *and* ρ_0 *is the initial density operator of a* $2n \times 2n$ *matrix whose* $(1, 1)$*-element is* 1 *and all the other elements are* 0*. Then the absorbing time of quantum walks on the* n*-dimensional hypercube with the absorbing vertex located at* $(1, 1, \ldots, 1)$ *is* $\mathrm{tr} X_n - 1$*.*

Proof. Let $p(t)$ denote the probability that the particle reaches the absorbing vertex $(1, 1, \ldots, 1)$ at time t for the first time. Then we have

$$\begin{aligned} p(t) &= \mathrm{tr}\big((U_n P'_n)^t \rho_0 (P'_n U_n^\dagger)^t P_n\big) \\ &= \mathrm{tr}\big((U_n P'_n)^t \rho_0 (P'_n U_n^\dagger)^t - (U_n P'_n)^{t+1} \rho_0 (P'_n U_n^\dagger)^{t+1}\big). \end{aligned}$$

From Lemma 2, both absorbing probability and nominal absorbing time are convergent series. Thus we have the following.

$$\begin{aligned} Prob &= \sum_{t=0}^{\infty} p(t) = \sum_{t=0}^{\infty} \mathrm{tr}\big((U_n P'_n)^t \rho_0 (P'_n U_n^\dagger)^t - (U_n P'_n)^{t+1} \rho_0 (P'_n U_n^\dagger)^{t+1}\big) \\ &= \mathrm{tr}\rho_0 = 1, \end{aligned}$$

$$\begin{aligned} Time_{\mathrm{nom}} &= \sum_{t=0}^{\infty} tp(t) = \sum_{t=0}^{\infty} t\mathrm{tr}\big((U_n P'_n)^t \rho_0 (P'_n U_n^\dagger)^t - (U_n P'_n)^{t+1} \rho_0 (P'_n U_n^\dagger)^{t+1}\big) \\ &= \mathrm{tr}\left(\sum_{t=0}^{\infty} (U_n P'_n)^{t+1} \rho_0 (P'_n U_n^\dagger)^{t+1}\right) = \mathrm{tr}(U_n P'_n X_n P'_n U_n^\dagger) = \mathrm{tr}X_n - 1. \end{aligned}$$

It follows that real absorbing time is $\mathrm{tr}X_n - 1$. □

Another characterization of values of the absorbing time is given by the following proposition.

Proposition 3. *Let $f_n(x)$ be a power series of x defined as*

$$f_n(x) = \sum_{t=0}^{\infty} a_t x^t = \frac{2^{n-1}(n-1)!}{n^{n-1}} \cdot \frac{x^{n+1}}{\det(xI_n - U_n P'_n)}.$$

Then the absorbing time of quantum walks on the n-dimensional hypercube with the absorbing vertex located at $(1, 1, \ldots, 1)$ is $\sum_{t=0}^{\infty} t|a_t|^2$.

Proof. Let $\langle b_n| = (0 \ \cdots \ 0 \ 1)$ and $|f_0\rangle = (1 \ 0 \ \cdots \ 0)^T$. Then $\langle b_n|(U_n P'_n)^t|f_0\rangle$ gives the amplitude that the particle reaches the absorbing vertex $(1, 1, \ldots, 1)$ at time t for the first time. Consider a power series of x, $g_n(x) = \sum_{t=0}^{\infty} \langle b_n|(U_n P'_n)^t|f_0\rangle x^t$. We prove that this $g_n(x)$ is equivalent to $f_n(x)$.

Note that $g_n(x) = (0 \ \cdots \ 0 \ 1)(\sum_{t=0}^{\infty} (xU_n P'_n)^t)(1 \ 0 \ \cdots \ 0)^T$, which is equal to the $(2n, 1)$-element of $(I_n - xU_n P_n)^{-1}$. Thus we have for $x \neq 0$,

$$g_n(x) = \frac{x^{2n}}{\det(xI_n - U_n P'_n)} (-1)^{2n+1} \Delta_{1,2n},$$

where $\Delta_{1,2n}$ is a minor of $xI_n - U_n P'_n$ with respect to $(1, 2n)$.

It is straightforward to show that $\Delta_{1,2n} = -\frac{2^{n-1}(n-1)!}{n^{n-1}} \cdot \frac{1}{x^{n-1}}$, and thus

$$g_n(x) = \frac{2^{n-1}(n-1)!}{n^{n-1}} \cdot \frac{x^{n+1}}{\det(xI_n - U_n P'_n)} = f_n(x).$$

By the definition of $g_n(x)$, each coefficient a_t of x^t in the power series $f_n(x)$ corresponds to the amplitude that the particle reaches the absorbing vertex $(1, 1, \ldots, 1)$ at time t for the first time. Since the absorbing probability is 1 from the proof of Proposition 2, the real absorbing time is $Time_{\mathrm{real}} = \sum_{t=0}^{\infty} t|a_t|^2$. □

References

1. D. Aharonov, A. Ambainis, J. Kempe, and U. V. Vazirani. Quantum walks on graphs. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 50–59, 2001.
2. A. Ambainis, E. Bach, A. Nayak, A. Vishwanath, and J. Watrous. One-dimensional quantum walks. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 37–49, 2001.
3. E. Bach, S. Coppersmith, M. P. Goldschen, R. Joynt, and J. Watrous. One-dimensional quantum walks with absorbing boundaries. Los Alamos e-print archive, quant-ph/0207008, 2002.
4. A. M. Childs, E. Farhi, and S. Gutmann. An example of the difference between quantum and classical random walks. Los Alamos e-print archive, quant-ph/0103020, 2001.
5. M. E. Dyer, A. M. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *Journal of the ACM*, 38(1):1–17, 1991.
6. L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.
7. T. Hofmeister, U. Schöning, R. Schuler, and O. Watanabe. A probabilistic 3-SAT algorithm further improved. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2285 of *Lecture Notes in Computer Science*, pages 192–202, 2002.
8. M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.
9. M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 712–721, 2001.
10. J. Kempe. Quantum random walks hit exponentially faster. Los Alamos e-print archive, quant-ph/0205083, 2002.
11. C. Moore and A. Russell. Quantum walks on the hypercube. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques in Computer Science*, *Lecture Notes in Computer Science*, 2002. To appear.
12. U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 410–414, 1999.
13. J. Watrous. Quantum simulations of classical random walks and undirected graph connectivity. *Journal of Computer and System Sciences*, 62(2):376–391, 2001.
14. T. Yamasaki. Quantum walks and two-way quantum one-counter automata. Master's thesis, Department of Information Science, The University of Tokyo, March 2002.

Author Index