

**Encyclopedia of  
Complexity and Systems Science Series**  
*Editor-in-Chief:* Robert A. Meyers

SPRINGER  
REFERENCE

Andrew Adamatzky *Editor*

# Unconventional Computing

A Volume in the Encyclopedia of  
Complexity and Systems Science,  
Second Edition

---

# Encyclopedia of Complexity and Systems Science Series

**Editor-in-Chief**

Robert A. Meyers

The Encyclopedia of Complexity and Systems Science Series of topical volumes provides an authoritative source for understanding and applying the concepts of complexity theory together with the tools and measures for analyzing complex systems in all fields of science and engineering. Many phenomena at all scales in science and engineering have the characteristics of complex systems and can be fully understood only through the transdisciplinary perspectives, theories, and tools of self-organization, synergetics, dynamical systems, turbulence, catastrophes, instabilities, nonlinearity, stochastic processes, chaos, neural networks, cellular automata, adaptive systems, genetic algorithms, and so on. Examples of near-term problems and major unknowns that can be approached through complexity and systems science include: the structure, history, and future of the universe; the biological basis of consciousness; the integration of genomics, proteomics, and bioinformatics as systems biology; human longevity limits; the limits of computing; sustainability of human societies and life on earth; predictability, dynamics, and extent of earthquakes, hurricanes, tsunamis, and other natural disasters; the dynamics of turbulent flows; lasers or fluids in physics; microprocessor design; macromolecular assembly in chemistry and biophysics; brain functions in cognitive neuroscience; climate change; ecosystem management; traffic management; and business cycles. All these seemingly diverse kinds of phenomena and structure formation have a number of important features and underlying structures in common. These deep structural similarities can be exploited to transfer analytical methods and understanding from one field to another. This unique work will extend the influence of complexity and system science to a much wider audience than has been possible to date.

More information about this series at <https://link.springer.com/bookseries/15581>

---

Andrew Adamatzky  
Editor

# Unconventional Computing

A Volume in the Encyclopedia of  
Complexity and Systems Science,  
Second Edition

With 363 Figures and 23 Tables



*Editor*

Andrew Adamatzky  
Unconventional Computing Centre  
University of the West of England  
Bristol, UK

ISBN 978-1-4939-6882-4      ISBN 978-1-4939-6883-1 (eBook)  
ISBN 978-1-4939-6884-8 (print and electronic bundle)  
<https://doi.org/10.1007/978-1-4939-6883-1>

Library of Congress Control Number: 2018947854

© Springer Science+Business Media, LLC, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

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

Printed on acid-free paper

This Springer imprint is published by the registered company Springer Science+Business Media, LLC part of Springer Nature.  
The registered company address is: 233 Spring Street, New York, NY 10013, U.S.A.

---

## Series Preface

The Encyclopedia of Complexity and System Science Series is a multivolume authoritative source for understanding and applying the basic tenets of complexity and systems theory as well as the tools and measures for analyzing complex systems in science, engineering, and many areas of social, financial, and business interactions. It is written for an audience of advanced university undergraduate and graduate students, professors, and professionals in a wide range of fields who must manage complexity on scales ranging from the atomic and molecular to the societal and global.

Complex systems are systems that comprise many interacting parts with the ability to generate a new quality of collective behavior through self-organization, e.g., the spontaneous formation of temporal, spatial, or functional structures. They are therefore adaptive as they evolve and may contain self-driving feedback loops. Thus, complex systems are much more than a sum of their parts. Complex systems are often characterized as having extreme sensitivity to initial conditions as well as emergent behavior that are not readily predictable or even completely deterministic. The conclusion is that a reductionist (bottom-up) approach is often an incomplete description of a phenomenon. This recognition that the collective behavior of the whole system cannot be simply inferred from the understanding of the behavior of the individual components has led to many new concepts and sophisticated mathematical and modeling tools for application to many scientific, engineering, and societal issues that can be adequately described only in terms of complexity and complex systems.

Examples of Grand Scientific Challenges which can be approached through complexity and systems science include: the structure, history, and future of the universe; the biological basis of consciousness; the true complexity of the genetic makeup and molecular functioning of humans (genetics and epigenetics) and other life forms; human longevity limits; unification of the laws of physics; the dynamics and extent of climate change and the effects of climate change; extending the boundaries of and understanding the theoretical limits of computing; sustainability of life on the earth; workings of the interior of the earth; predictability, dynamics, and extent of earthquakes, tsunamis, and other natural disasters; dynamics of turbulent flows and the motion of granular materials; the structure of atoms as expressed in the Standard Model and the formulation of the Standard Model and gravity into a Unified Theory; the structure of water; control of global infectious diseases; and also evolution and quantification of (ultimately) human cooperative behavior in politics,

economics, business systems, and social interactions. In fact, most of these issues have identified nonlinearities and are beginning to be addressed with nonlinear techniques, e.g., human longevity limits, the Standard Model, climate change, earthquake prediction, workings of the earth's interior, natural disaster prediction, etc.

The individual complex systems mathematical and modeling tools and scientific and engineering applications that comprised the *Encyclopedia of Complexity and Systems Science* are being completely updated and the majority will be published as individual books edited by experts in each field who are eminent university faculty members.

The topics are as follows:

Agent Based Modeling and Simulation  
Applications of Physics and Mathematics to Social Science  
Cellular Automata, Mathematical Basis of  
Chaos and Complexity in Astrophysics  
Climate Modeling, Global Warming, and Weather Prediction  
Complex Networks and Graph Theory  
Complexity and Nonlinearity in Autonomous Robotics  
Complexity in Computational Chemistry  
Complexity in Earthquakes, Tsunamis, and Volcanoes, and Forecasting and Early Warning of Their Hazards  
Computational and Theoretical Nanoscience  
Control and Dynamical Systems  
Data Mining and Knowledge Discovery  
Ecological Complexity  
Ergodic Theory  
Finance and Econometrics  
Fractals and Multifractals  
Game Theory  
Granular Computing  
Intelligent Systems  
Nonlinear Ordinary Differential Equations and Dynamical Systems  
Nonlinear Partial Differential Equations  
Percolation  
Perturbation Theory  
Probability and Statistics in Complex Systems  
Quantum Information Science  
Social Network Analysis  
Soft Computing  
Solitons  
Statistical and Nonlinear Physics  
Synergetics  
System Dynamics  
Systems Biology

Each entry in each of the Series books was selected and peer reviewed organized by one of our university-based book Editors with advice and

consultation provided by our eminent Board Members and the Editor-in-Chief. This level of coordination assures that the reader can have a level of confidence in the relevance and accuracy of the information far exceeding than that generally found on the World Wide Web. Accessibility is also a priority and for this reason each entry includes a glossary of important terms and a concise definition of the subject. In addition, we are pleased that the mathematical portions of our Encyclopedia have been selected by Math Reviews for indexing in MathSciNet. Also, ACM, the world's largest educational and scientific computing society, recognized our *Computational Complexity: Theory, Techniques, and Applications* book, which contains content taken exclusively from the *Encyclopedia of Complexity and Systems Science*, with an award as one of the notable Computer Science publications. Clearly, we have achieved prominence at a level beyond our expectations, but consistent with the high quality of the content!

Palm Desert, CA, USA  
July 2018

Robert A. Meyers  
Editor-in-Chief

---

## Volume Preface

### UNCONVENTIONAL COMPUTING. Other ways to compute.

Unconventional computing is a field without definition.

There are two kinds of unconventional computing.

First one is about implementation of computing systems in non-silicon substrates, e.g., chemical systems, plants, electricity, and slime mold. Here, everything is clear and straight. You can touch novel computing devices and feel how they compute.

The second kind deals with dissident thinking, by challenging current stereotypes and dogmas. You might say that everything going on in our minds is conventional because it happened before in *other* peoples' minds. You are right: we can call then this second kind of unconventional computing as "uncommon thinking about computing." Whatever you name it, it remains entertaining and mind-opening. Thus, the purpose of unconventional computing is not just in discovering mechanisms of information processing in chemical, physical, and biological systems and using these mechanisms to develop novel, and sometimes even efficient, architectures and prototypes of future computers.

The purpose of unconventional computing is also in uncovering other ways to compute. These other ways are alike alternative state of minds. They are fascinating; they change your understanding of life but they are short living.

What is unconventional computing? Few people know the answer. Authors of the book believe they do. They crafted extraordinary good chapters about physics, chemistry, biology, and theoretical foundations of "*other ways to compute.*" All authors of the book are world leading authorities in interdisciplinary spanning mathematics, computer science, physics, chemistry, biology, and electronic engineering.

Bristol, UK

July 2018

Andrew Adamatzky

Volume Editor

---

# Unconventional Computing Editorial

---

## Andrew Adamatzky, Editor

Unconventional computing is a science in a flux, where norms are dynamics and truth is relative, or, less poetically and by Cambridge dictionary, this is different from what is usual or from the way most people do things. The field of unconventional computing is highly interdisciplinary and open to wild ideas. We grouped chapters in two parts – “Practical, experimental laboratory computing” and “Theorizing about computing and nature-inspired algorithms.” First part includes not only chapters discussing results of laboratory experiments with novel computing substrates but also computer models, which imitate novel computing substrates with near physical accuracy. Results presented in the second part deal mainly with novel theories of computation and software implementation of algorithms inspired by nature.

---

## Part I: Practical, Experimental Laboratory Computing

In 1876, Lord Kelvin envisaged a computation by directly exploiting law of Nature and invented differential analyzer. Ideas of analog computation emerged, flourished, almost deceased by 1980s, and then were resurrected in 1990s in ever-growing field of computing based on continuous representations by means of continuous processes. Chapter ► “[Analog Computation,](#)” by Bruce J. MacLennan, discusses history of analog computing from Kirchhoff and Thompson devices to electronic analog computers in 1940s. A reader will enjoy concise introduction to analog, very large scale integration circuits, field programmable analog arrays, fundamentals of analog computing, continuous state space, precision and scaling of analog computation, overview of analog computation in nature and exotic devices, such as Rubel’s extended analog computer and transcending Turing computability. The theme of analogous computing is augment with details of mechanical devices in chapter ► “[Mechanical Computing: The Computational Complexity of Physical Devices,](#)” by John H. Reif. The chapter makes an excursion into the history of mechanical computation: harmonic analyzers and synthesizers, Napier’s bones, Pascal’s wheeled calculator, and advances into self-assembling devices and molecular machines. Topics discussed include a computational complexity of motion planning and simulation of mechanical devices, hardness results for mechanical devices with a constant number of degrees of freedom, analog

mechanical devices for arithmetic operations, mechanical and electro-optical devices for integer factorization.

The electro-optical devices discussed recall us about optical computing, where information is transmitted and processed with photons. The chapter ► “[Optical Computing](#),” by Thomas J. Naughton and Damien Woods, introduces optical pattern recognition, analog optical computing, digital optical computing, hardware of optical computers, continuous state machines, and complexity of computing with photons. Photons are key players of photonic integrated circuits, discussed in the chapter ► “[Principles of Neuromorphic Photonics](#),” by Bhavin J. Shastri, Alexander N. Tait, Thomas Ferreira de Lima, Mitchell A. Nahmias, Hsuan-Tung Peng, and Paul R. Prucnal, where lasers, modulators, filters, and detectors are used instead of conventional electronic elements. The chapters give an excellent introduction to the elementary base of photonic neurons, including photonic spiking processes, excitable/spiking laser, and photonic analogies with leaky integrate-and-fire model. While thinking about photons we enter a field of quantum mechanics. The logic of quantum mechanics is used in quantum computing. The chapter ► “[Quantum Computing](#),” by Viv Kendon, is a very assessable introduction to quantum computing, discussing concepts and implementations of qubits, quantum gates, error correction for quantum computers, programming quantum computers, implementation of Shor’s algorithm and quantum walks, and estimation of power of quantum computing. Concepts of continuous variable quantum computing, hybrid quantum computing architectures, adiabatic quantum computing, and quantum annealing are presented in a lively manner.

Waves and other propagating and traveling processes and artifacts are ubiquitous in nature. Devices computing with waves are described in next chapters of the book. The chapter ► “[Computing with Solitons](#),” by Darren Rand and Ken Steiglitz, gives us a unique insight on logical circuits implemented in collisions between solitary waves. The chapter is theoretical; however, the modeled prototypes can be implemented in experimental laboratory conditions. The soliton-based devices compute by colliding self-preserved solitary waves, which change their state or trajectories in result of the collision and thus are capable of implementing basic logical gates. Exact topics considered include scalar and vector solitons, computation with Manakov solitons and implementation of FANOUT, NOT, and ONE gates, output/input converters, NAND gate, and time gating in the solitons-based circuits.

Waves propagating in unconstrained, or free-space, chemical systems are also proved to be capable for implementation of logical circuits, and robot control and computational geometry, as shown in the chapter ► “[Reaction-Diffusion Computing](#),” by Andrew Adamatzky and Benjamin De Lacy Costello. In reaction-diffusion processors, both the data and the results of the computation are encoded as concentration profiles of the reagents. The computation per se is performed via the spreading and interaction of wave fronts. The chapter demonstrates experimental laboratory prototypes of precipitating chemical processors for computing of Voronoi diagram and skeleton of a planar shape, collision-free path calculation, control of robotic hand, and implementation of logical gates and arithmetic circuits in Belousov-

Zhabotinsky excitable chemical medium. Belousov-Zhabotinsky medium confined to geometrical restrictions of channel in junctions is further explored for computation in the chapter ► “[Computing in Geometrical Constrained Excitable Chemical Systems](#),” by Jerzy Gorecki and Joanna Natalia, where designs of diodes, frequency transformers, logical gates, and chemical sensors are presented. Overview of the semi-conductor implementation of reaction-diffusion computers in large-scale integrated circuits is presented in chapter ► “[Novel Hardware for Unconventional Computing](#),” by Tetsuya Asai. This includes digital complementary metal–oxide–semiconductor chips, reaction-diffusion circuits based on cellular automaton processing emulating the Belousov-Zhabotinsky reaction, silicon implementation of a chemical reaction-diffusion processor for computation of Voronoi diagram, reaction-diffusion computing devices based on minority-carrier transport in semiconductors, and collision-based reaction-diffusion computers. Our excursions to the field of physical and chemical computing substrates concludes with the chapter ► “[Thermodynamics of Computation](#),” by H. John Caulfield and Lei Qian, which analyses relationship between thermodynamics and computing. Particular attention in the context is paid to the energetics and temporal properties of classical digital computers, conservative computing, thermodynamics of analog and digital computers, and quantum and optical circuits. From chemical computers we move to biochemical and molecular computers. The chapter ► “[Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications](#),” by Evgeny Katz, presents a detailed overview of computing devices implemented with enzymatic reactions and networks: enzyme-based logic gates and their interfacing with conventional electronics. The methods used to analyze output signals generated by enzyme-based computing circuits include optical and electrochemical techniques, impedance spectroscopy, and conductivity measurements. These are augmented by transduction of chemical output signals produced by enzyme-based logic systems using semiconductor devices and implementation of digital biosensors based on enzyme logic gates.

Almost a quarter century ago, solution of a few points Hamiltonian path problem with DNA has been demonstrated, yet DNA based computing remains boisterous and right now show signs of further explosive growth. Despite quite a respectable age, the field can still be classed as unconventional computing, because one is using “DNA computers” in a regular life. The chapter ► “[DNA Computing](#),” by Martyn Amos, gives an authoritative introduction to the concepts and implementations of the DNA computers and shows how DNA ensembles solve Hamiltonian path problem, simulate implementation of chess problem, and execute binary arithmetic. Nucleic acids are also employed in molecular automata described in chapter ► “[Molecular Automata](#),” by Joanne Macdonald, Darko Stefanovic, and Milan Stojanovic. Via phenomena of binding, dissociation, and catalytic actions the molecular automata implement sensing and computation. The chapter overviews molecular automata as language recognizers, algorithmic DNA self-assembly, molecular automata as transducers and controllers, and deoxyribozyme logic gates. Experimental laboratory prototypes of the molecular automata include therapeutic and diagnostic automata, arithmetical circuits, and game solvers.

The molecular automata are nanocomputers. Other types of nanocomputers and particulars of their implementation are discussed in the chapter ► “[Nano-computers,](#)” by Ferdinand Peper. The chapter introduces requirements for nano-electronic devices and considers their scalability by several orders of magnitude beyond complementary metal–oxide–semiconductor chips. High signals processing capacity is analyzed in a context of high switching speeds and a high degree of parallelism. Practical issues of nano-fabrication, heat dissipation, and fault tolerance of nanocomputers shed lights onto future implementations. Cellular automata, crossbar array, and neural network based architectures are analyzed. The chapter ► “[Cellular Computing,](#)” by Christof Teuscher, overviews cellular computing hardware, as initially conserved in bio-inspired embryonic electronics project aimed to design highly robust integrated circuits features properties of living cells – self-repair and self-replication.

A concept of a wire, called mnemotrix, which decreases its resistance when traversed by an electrical current was proposed in 1980s, and experimentally implemented with organic conductive polymer polyaniline early 2000s. The chapter ► “[Neuromorphic Computing Based on Organic Memristive Systems,](#)” by Victor Erokhin, introduces computing architectures and properties based on polyaniline polymers and shows how to experimentally realize oscillators, synapses, logical gates, perceptron, and Pavlovian learning with the conductive polymers.

Slime mold *Physarum polycephalum* is a single cell visible by unaided eye. During its foraging behavior, the plasmodium spans scattered sources of nutrients with a network of protoplasmic tubes. The plasmodium optimizes its protoplasmic network to cover all sources of nutrients, stay away from repellents, and minimize transportation of metabolites inside its body. The plasmodium’s ability to optimize its shape attracted the attention of biologists and later computer scientists. The chapter ► “[Slime Mold Computing,](#)” by Andrew Adamatzky, overviews a spectrum of computing devices experimentally prototyped with the living slime mold. These include shortest path and maze solvers, approximation of spanning trees, Voronoi diagram, concave hull, attraction-based, ballistic, opto-electronic, frequency-based and microfluidic logicals gates, and implementation of Kolmogorov-Uspensky machine.

The chapter ► “[Evolution in Materio,](#)” by Simon Harding and Julian F. Miller, is about evolving computing abilities of unconventional computing substrates by reconfiguring their physical structure. The evolutionary algorithms are used to find values of input variables that should be applied to a substrate so that the substrate carries out useful computation. Examples include evolving liquid crystals, conductive and electro-activated polymers, voltage controlled colloids, Langmuir-Blodgett films, and Kirchhoff-Lukasiewicz machines. Another chapter ► “[Reservoir Computing,](#)” by Zoran Konkoli, relates to the “Evolution in Materio.”

The chapter ► “[Reversible Computing,](#)” by Kenichi Morita, does not offer any experimental laboratory results; however, some theoretical designs, e.g., a billiard ball gate, presented there have been experimentally prototyped; therefore, we brought the chapter here. Every computational configuration, global state, has exactly one previous configuration. The chapter introduces designs

of reversible logic elements and circuits, including reversible logical gates, reversible elements with memory, rotary element, billiard-ball model, reversible Turin machines, and garbage-less computation.

---

## Part II: Theory of Computation and Nature-Inspired Algorithms

Chapters in this part are very diverse in their contents, relationship with reality, and (dis) affinity to any mainstream sciences. We start with the chapter ► “[Bacterial Computing](#),” by Martyn Amos, offering a discourse in a conceptual subset of synthetic biology. The chapter discusses fundamental operations of the engineered biological systems, synthetic chemistry, optimization of biological systems, and developing engineering principles to the design and construction of biological systems. The chapter is followed by the chapter ► “[Immune Computing](#),” by Jon Timmis, which focuses on the learning and memory mechanisms of the immune system inherent in clonal selection and immune networks, including artificial cloning selection, alternative immunological theories as danger theory, and cognitive immunology.

Membrane computing is inspired by architectures and functioning of living cells, their ensembles, tissues, and organs. Two chapters of the book deal with the membrane computing. First chapter ► “[Membrane Computing: Power and Complexity](#),” by Marian Gheorghe, Andrei Păun, Sergey Verlan, and Gexiang Zhang, introduces cell-like, tissue-like, neural-like P-systems, gives an introduction to numerical P systems and spiking neural P systems, and touches up the topic of computational efficiency of membrane computing. Second chapter ► “[Applications of P Systems](#),” by Marian Gheorghe, Andrei Păun, Sergey Verlan, and Gexiang Zhang, exemplifies applications of membrane computing in synthetic biology, real-life complex problems, including fault-diagnosis, robot control, sorting algorithms, synchronization, and broadcasting. Bacterial colonies, clones of immune cells, cells in tissues, discussed in previous chapters, are in fact social systems. The computation in social systems is advanced in the chapter ► “[Social Algorithms](#),” by Xin-She Yang. There multiple agents and the “social” interactions are used to design algorithms that mimic some useful traits of the social/biological systems. This comprehensive introduction to social algorithms analyzes algorithms inspired by bees, bats, fireflies, and cuckoos and ant colony optimization. More about ants can be found in the chapter ► “[Cellular Ants Computing](#),” by Georgios Sirakoulis, where ant-colony optimization algorithms are implemented in cellular automata, and topics of cellular ants and data clustering, and swarm robotics are well treated.

Two next chapters are here rather for historical reasons because the topics discussed were very popular when the field of unconventional computing was established. Chemical and molecular media and substrates are now widely explored as prototypes of future nonstandard computing devices. The chapter ► “[Artificial Chemistry](#),” by Peter Dittrich, discusses models based interpretations of chemical reactions as computing processes, including autocatalytic polymer chemistries, chemistries inspired by Turing machines, and lattice

molecular systems. The ideas of pools of interacting processes are echoed in the chapter ► “[Amorphous Computing](#),” by Hal Abelson, Jacob Bean, and Gerald Jay Sussman. The amorphous computers are built of a collection of computational particles, with no a priori knowledge of their positions or orientations, dispersed irregularly on a surface or throughout a volume, a cloud of computing particles computing. Another approach to compute with pools of particles or in fact any chemical, physical, and biological substrate is offered in the chapter ► “[Reservoir Computing](#),” by Zoran Konkoli. The chapter introduces a reservoir computer consisting of two components: a dynamical system that accepts an input signal and a trainable readout layer which is used to produce the output – any computation can be achieved by training the readout layer for a necessary functionality. The chapter introduces the models of Liquid State Machines and Echo State Networks.

The remaining chapters of the book are provocative because being provocative is one of the key features of the unconventional computing. An unconventional computational problem requires more than one algorithmic step per time unit and therefore cannot be solved on a computer with a finite and fixed number of algorithmic steps per time unit. The chapter ► “[Unconventional Computational Problems](#),” by Selim G. Akl, considers the unconventional problems in the framework of computation obeying mathematical constraints, time-varying and rank-varying computational complexities, time-varying and interacting variables, and uncertain time constraints. While it is unclear whether the human mind has Turing super- or sub-universal capabilities, the chapter ► “[Algorithmic Cognition and the Computational Nature of the Mind](#),” by Hector Zenil and Nicolas Gauvrit, argues that the algorithmic approach to cognition provides a strong formal connection between cognition and computation by means of recursion. An unconventional approach to computing the uncomputable is offered in the chapter ► “[Approximations to Algorithmic Probability](#),” by Hector Zenil. The chapter positions computability and algorithmic complexity at the center of causality in science and discusses how new methods have been advanced to estimate universal measures of complexity. The chapter ► “[Grossone Infinity Computing](#),” by Yaroslav D. Sergeyev, proposes a numeral system that uses the same numerals for several different purposes for dealing with infinities and infinitesimals. Examples include, measuring infinite sets, indicating positions of elements in ordered infinite sequences, working with functions and their derivatives that can assume different infinite, describing Turing machines. Natural processes never halt. Even death is not the end of life. Inductive Turing machine, first super-recursive class of abstract automata, overviewed in the chapter ► “[Inductive Turing Machines](#),” by Mark Burgin, is a machine that never stops. The inductive Turing machines could be applied in algorithmic information theory and complexity studies, software testing, evolutionary information theory, logic, machine learning and artificial intelligence, software engineering, and computer networks.

---

## Contents

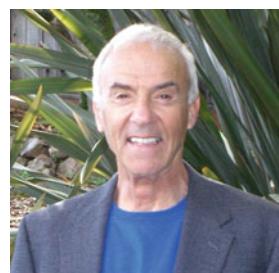
<b>Part I Practical, Experimental Laboratory Computing .....</b>	<b>1</b>
<b>Analog Computation .....</b>	<b>3</b>
Bruce J. MacLennan	
<b>Mechanical Computing: The Computational Complexity of Physical Devices .....</b>	<b>35</b>
John H. Reif	
<b>Optical Computing .....</b>	<b>57</b>
Thomas J. Naughton and Damien Woods	
<b>Principles of Neuromorphic Photonics .....</b>	<b>83</b>
Bhavin J. Shastri, Alexander N. Tait, Thomas Ferreira de Lima, Mitchell A. Nahmias, Hsuan-Tung Peng and Paul R. Prucnal	
<b>Quantum Computing .....</b>	<b>119</b>
Viv Kendon	
<b>Computing with Solitons .....</b>	<b>147</b>
Darren Rand and Ken Steiglitz	
<b>Reaction-Diffusion Computing .....</b>	<b>171</b>
Andrew Adamatzky and Benjamin De Lacy Costello	
<b>Computing in Geometrical Constrained Excitable Chemical Systems .....</b>	<b>195</b>
Jerzy Gorecki and Joanna Natalia Gorecka	
<b>Novel Hardware for Unconventional Computing .....</b>	<b>225</b>
Tetsuya Asai	
<b>Thermodynamics of Computation .....</b>	<b>251</b>
H. John Caulfield and Lei Qian	
<b>Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications .....</b>	<b>265</b>
Evgeny Katz	

<b>DNA Computing</b>	307
Martyn Amos	
<b>Molecular Automata</b>	327
Joanne Macdonald, Darko Stefanovic and Milan Stojanovic	
<b>Nanocomputers</b>	355
Ferdinand Peper	
<b>Cellular Computing</b>	393
Christof Teuscher	
<b>Neuromorphic Computing Based on Organic Memristive Systems</b>	411
Victor Erokhin	
<b>Slime Mold Computing</b>	431
Andrew Adamatzky	
<b>Evolution in Materio</b>	447
Simon Harding and Julian F. Miller	
<b>Reversible Computing</b>	463
Kenichi Morita	
<b>Part II Theory of Computation and Nature-Inspired Algorithms</b>	489
<b>Bacterial Computing</b>	491
Martyn Amos	
<b>Immune Computing</b>	503
Jon Timmis	
<b>Membrane Computing: Power and Complexity</b>	519
Marian Gheorghe, Andrei Păun, Sergey Verlan and Gexiang Zhang	
<b>Applications of P Systems</b>	535
Marian Gheorghe, Andrei Păun, Sergey Verlan and Gexiang Zhang	
<b>Social Algorithms</b>	549
Xin-She Yang	
<b>Cellular Ants Computing</b>	565
Konstantinos Ioannidis and Georgios Ch. Sirakoulis	
<b>Artificial Chemistry</b>	577
Peter Dittrich	
<b>Amorphous Computing</b>	601
Hal Abelson, Jacob Beal and Gerald Jay Sussman	
<b>Reservoir Computing</b>	619
Zoran Konkoli	

<b>Unconventional Computational Problems</b>	.....	631
Selim G. Akl		
<b>Algorithmic Cognition and the Computational Nature of the Mind</b>	.....	641
Hector Zenil and Nicolas Gauvrit		
<b>Approximations to Algorithmic Probability</b>	.....	651
Hector Zenil		
<b>Grossone Infinity Computing</b>	.....	663
Yaroslav D. Sergeyev		
<b>Inductive Turing Machines</b>	.....	675
Mark Burgin		
<b>Index</b>	.....	689

---

## About the Editor-in-Chief



### **Dr. Robert A. Meyers**

President: RAMTECH Limited

Manger, Chemical Process Technology, TRW Inc.

Post doctoral Fellow: California Institute of Technology

Ph.D. Chemistry, University of California at Los Angeles

B.A. Chemistry, California State University, San Diego

---

## **Biography**

Dr. Meyers has worked with more than 20 Nobel laureates during his career and is the originator and serves as Editor-in-Chief of both the Springer Nature *Encyclopedia of Sustainability Science and Technology* and the related and supportive Springer Nature *Encyclopedia of Complexity and Systems Science*.

---

## **Education**

Postdoctoral Fellow: California Institute of Technology

Ph.D. in Organic Chemistry, University of California at Los Angeles

B.A. Chemistry with minor in Mathematics, California State University, San Diego

Dr. Meyers holds more than 20 patents and is the author or Editor-in-Chief of 12 technical books including the *Handbook of Chemical Production*

*Processes, Handbook of Synfuels Technology, and Handbook of Petroleum Refining Processes* now in 4th Edition, and the *Handbook of Petrochemical Production Processes*, now in its second edition, (McGraw-Hill) and the *Handbook of Energy Technology and Economics*, published by John Wiley & Sons; *Coal Structure*, published by Academic Press; and *Coal Desulfurization* as well as the *Coal Handbook* published by Marcel Dekker. He served as Chairman of the Advisory Board for *A Guide to Nuclear Power Technology*, published by John Wiley & Sons, which won the Association of American Publishers Award as the best book in technology and engineering.

---

## About the Volume Editor



Andrew Adamatzky is Professor in Unconventional Computing at the Department of Computer Science and Director of the Unconventional Computing Laboratory, University of the West of England, Bristol. He does research in theoretical models of computation, cellular automata theory and applications, molecular computing, reaction-diffusion computing, collision-based computing, slime mold computing, massive parallel computation, applied mathematics, complexity, nature-inspired optimization, collective intelligence, bionics, computational psychology, nonlinear science, novel hardware, and future and emergent computation. He invented and developed new fields of computing – reaction-diffusion computing and slime mold computing – which are now listed as key topics of all major conferences in computer science and future and emerging technologies. His first authored book was *Identification of Cellular Automata* (Taylor & Francis, 1994). He authored seven books, most notable are *Reaction-Diffusion Computing* (Elsevier, 2005), *Dynamics of Crow Minds* (World Scientific, 2005), *Physarum Machines* (World Scientific, 2010), and *Reaction-Diffusion Automata* (Springer, 2013) and edited 22 books in computing, most notable are *Collision Based Computing* (Springer, 2002), *Game of Life Cellular Automata* (Springer, 2010), and *Memristor Networks* (Springer, 2014); he also produced a series of influential artworks published in the atlas *Silence of Slime Mould* (Luniver Press, 2014). He is founding editor-in-chief of *Journal of Cellular Automata* and *Journal of Unconventional Computing* (both published by OCP Science, USA) and editor-in-chief of *Parallel, Emergent, and Distributed Systems* (Taylor & Francis) and *Parallel Processing Letters* (World Scientific). He is co-founder of Springer Series *Emergence, Complexity and Computation*, which publishes elected topics in the fields of complexity, computation, and emergence, including all aspects of reality-based computation approaches from an interdisciplinary point of view

especially from applied sciences, biology, physics, or chemistry. Adamatzky is famous for his unorthodox ideas, which attracted substantial funding from UK and EU, including computing with liquid marbles, living architectures, growing computers with slime mold, learning and computation in memristor networks, artificial wet neural networks, biologically inspired transportation, collision-based computing, dynamical logical circuits in sub-exitable media, particle dynamics in cellular automata, and amorphous biological intelligence.

---

## Contributors

**Hal Abelson** Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Andrew Adamatzky** Unconventional Computing Centre, University of the West of England, Bristol, UK

**Selim G. Akl** School of Computing and Department of Mathematics and Statistics, Queen's University, Kingston, ON, Canada

**Martyn Amos** Department of Computing and Mathematics, Manchester Metropolitan University, Manchester, UK

**Tetsuya Asai** Hokkaido University, Sapporo, Japan

**Jacob Beal** Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Mark Burgin** Department of Mathematics, UCLA, Los Angeles, CA, USA

**H. John Caulfield** Fisk University, Nashville, TN, USA

**Benjamin De Lacy Costello** Centre for Analytical Chemistry and Smart Materials, University of the West of England, Bristol, UK

**Peter Dittrich** Department of Mathematics and Computer Science, Friedrich Schiller University Jena, Jena, Germany

**Victor Erokhin** Institute of Materials for Electronics and Magnetism, Italian National Council of Research (CNR-IMEM), Parma, Italy

**Thomas Ferreira de Lima** Department of Electrical Engineering, Princeton University, Princeton, NJ, USA

**Nicolas Gauvrit** Human and Artificial Cognition Lab, EPHE, Paris, France

**Marian Gheorghe** School of Electrical Engineering and Computer Science, University of Bradford, Bradford, West Yorkshire, UK

**Joanna Natalia Gorecka** Institute of Physics, Polish Academy of Science, Warsaw, Poland

**Jerzy Gorecki** Institute of Physical Chemistry, Polish Academy of Science, Warsaw, Poland

**Simon Harding** Department of Computer Science, Memorial University, St. John's, Canada

**Konstantinos Ioannidis** School of Engineering, Department of Electrical and Computer Engineering, Democritus University of Thrace (DUTH), Xanthi, Greece

**Evgeny Katz** Department of Chemistry and Biomolecular Science, Clarkson University, Potsdam, NY, USA

**Viv Kendon** JQC and Atmol, Department of Physics, Durham University, Durham, UK

**Zoran Konkoli** Department of Microtechnology and Nanoscience - MC2, Chalmers University of Technology, Gothenburg, Sweden

**Joanne Macdonald** Division of Experimental Therapeutics, Department of Medicine, Columbia University, NY, USA

Genecology Research Centre, Inflammation and Healing Research Cluster, School of Science and Engineering, University of the Sunshine Coast, Queensland, Australia

**Bruce J. MacLennan** Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA

**Julian F. Miller** Department of Electronics, University of York, Heslington, UK

**Kenichi Morita** Hiroshima University, Higashi-Hiroshima, Japan

**Mitchell A. Nahmias** Department of Electrical Engineering, Princeton University, Princeton, NJ, USA

**Thomas J. Naughton** Department of Computer Science, National University of Ireland, Maynooth County Kildare, Ireland

**Andrei Păun** Department of Computer Science, University of Bucharest, Bucharest, Romania

**Hsuan-Tung Peng** Department of Electrical Engineering, Princeton University, Princeton, NJ, USA

**Ferdinand Peper** National Institute of Information and Communications Technology, Kobe, Japan

**Paul R. Prucnal** Department of Electrical Engineering, Princeton University, Princeton, NJ, USA

**Lei Qian** Fisk University, Nashville, TN, USA

**Darren Rand** Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, USA

**John H. Reif** Department of Computer Science, Duke University, Durham, NC, USA

**Yaroslav D. Sergeyev** University of Calabria, Rende, Italy

Lobachevsky State University, Nizhni Novgorod, Russia

Institute of High Performance Computing and Networking, C.N.R., Rome, Italy

**Bhavin J. Shastri** Department of Electrical Engineering, Princeton University, Princeton, NJ, USA

**Georgios Ch. Sirakoulis** School of Engineering, Department of Electrical and Computer Engineering, Democritus University of Thrace (DUTH), Xanthi, Greece

**Darko Stefanovic** Department of Computer Science and Center for Biomedical Engineering, University of New Mexico, Albuquerque, NM, USA

**Ken Steiglitz** Computer Science Department, Princeton University, Princeton, NJ, USA

**Milan Stojanovic** Division of Experimental Therapeutics, Department of Medicine, Columbia University, NY, USA

**Gerald Jay Sussman** Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

**Alexander N. Tait** Department of Electrical Engineering, Princeton University, Princeton, NJ, USA

**Christof Teuscher** Portland State University, Portland, OR, USA

**Jon Timmis** Department of Electronics, Department of Computer Science, University of York, York, UK

**Sergey Verlan** LACL, Université Paris Est Créteil, Créteil, France

**Damien Woods** Computer Science, California Institute of Technology, Pasadena, CA, USA

**Xin-She Yang** School of Science and Technology, Middlesex University, London, UK

Department of Engineering, University of Cambridge, Cambridge, UK

**Hector Zenil** Algorithmic Dynamics Lab, Unit of Computational Medicine and SciLifeLab, Center for Molecular Medicine, Department of Medicine Solna, Karolinska Institutet, Stockholm, Sweden

**Gexiang Zhang** Robotics Research Center, Xihua University, Chengdu, Sichuan, China

Key Laboratory of Fluid and Power Machinery, Xihua University, Ministry of Education, Chengdu, Sichuan, China

School of Electrical Engineering, Southwest Jiaotong University, Chengdu, Sichuan, China

---

## **Part I**

### **Practical, Experimental Laboratory Computing**



---

## Analog Computation

Bruce J. MacLennan

Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, USA

### Article Outline

Glossary  
Introduction  
History  
Article Road Map  
Fundamentals of Analog Computing  
Computational Process  
Analog Computer Programs  
Characteristics of Analog Computation  
Analog Computation in Nature  
Is Everything a Computer?  
General-Purpose Analog Computation  
Analog Computation and the Turing Limit  
A Sampling of Theoretical Results  
Real-Valued Inputs, Outputs, and Constants  
The Issue of Simulation by Turing Machines and Digital Computers  
The Problem of Models of Computation  
Relevant Issues for Analog Computation  
Transcending Turing Computability  
Analog Thinking  
Future Directions  
Bibliography

### Glossary

**Accuracy** The closeness of a computation to the corresponding primary system  
**BSS** The theory of computation over the real numbers defined by Blum, Shub, and Smale  
**Church–Turing (CT) computation** The model of computation based on the Turing machine and other equivalent abstract computing

machines commonly accepted as defining the limits of digital computation

**EAC** Extended analog computer defined by Rubel

**GPAC** General-purpose analog computer

**Nomograph** A device for the graphical solution of equations by means of a family of curves and a straightedge

**ODE** Ordinary differential equation

**PDE** Partial differential equation

**Potentiometer** A variable resistance adjustable by the computer operator, used in electronic analog computing as an attenuator for setting constants and parameters in a computation

**Precision** The quality of an analog representation or computation which depends on both resolution and stability

**Primary system** The system being simulated, modeled, analyzed, or controlled by an analog computer, also called the *target system*

**Scaling** The adjustment by constant multiplication of variables in the primary system (including time) so that the corresponding variables in the analog systems are in an appropriate range

**TM** Turing machine

### Introduction

#### Definition of the Subject

Although analog computation was eclipsed by digital computation in the second half of the twentieth century, it has returned as an important alternative computing technology. Indeed, as explained in this article, theoretical results imply that analog computation can escape the limitations of digital computation. Furthermore, analog computation has emerged as an important theoretical framework for discussing computation in the brain and other natural systems.

Analog computation gets its name from an analogy, or systematic relationship, between the physical processes in the computer and those in

the system it is intended to model or simulate (the primary system). For example, the electrical quantities voltage, current, and conductance might be used as analogs of the fluid pressure, flow rate, and pipe diameter. More specifically, in traditional analog computation, physical quantities in the computation obey the same mathematical laws as physical quantities in the primary system. Thus, the computational quantities are proportional to the modeled quantities. This is in contrast to digital computation, in which quantities are represented by strings of symbols (e.g., binary digits) that have no direct physical relationship to the modeled quantities. According to the Oxford English Dictionary (2nd ed., s.vv. *analogue*, *digital*), these usages emerged in the 1940s.

However, in a fundamental sense, all computing is based on an analogy, that is, on a systematic relationship between the states and processes in the computer and those in the primary system. In a digital computer, the relationship is more abstract and complex than simple proportionality, but even so simple an analog computer as a slide rule goes beyond strict proportion (i.e., distance on the rule is proportional to the logarithm of the number). In both analog and digital computation – indeed in all computation – the relevant abstract mathematical structure of the problem is realized in the physical states and processes of the computer, but the realization may be more or less direct (MacLennan 1994a, b, 2004).

Therefore, despite the etymologies of the terms “digital” and “analog,” in modern usage the principal distinction between digital and analog computation is that the former operates on discrete representations in discrete steps, while the latter operates on continuous representations by means of continuous processes (e.g., MacLennan 2004; Siegelmann 1999, p. 147; Small 2001, p. 30; Weyrick 1969, p. 3).

That is, the primary distinction resides in the topologies of the states and processes, and it would be more accurate to refer to discrete and continuous computation (Goldstine 1972, p. 39). (Consider so-called analog and digital clocks. The principal difference resides in the continuity or discreteness of the representation of time; the motion of the two (or three) hands of an “analog”

clock does not mimic the motion of the rotating earth or the position of the sun relative to it.)

## History

### Preelectronic Analog Computation

Just like digital calculation, analog computation was originally performed by hand. Thus, we find several analog computational procedures in the “constructions” of Euclidean geometry (Euclid, fl. 300 BCE), which derive from techniques used in ancient surveying and architecture. For example, Problem II.51 is “to divide a given straight line into two parts, so that the rectangle contained by the whole and one of the parts shall be equal to the square of the other part.” Also, Problem VI.13 is “to find a mean proportional between two given straight lines,” and VI.30 is “to cut a given straight line in extreme and mean ratio.” These procedures do not make use of measurements in terms of any fixed unit or of digital calculation; the lengths and other continuous quantities are manipulated directly (via compass and straightedge). On the other hand, the techniques involve discrete, precise operational steps, and so they can be considered algorithms, but over continuous magnitudes rather than discrete numbers.

It is interesting to note that the ancient Greeks distinguished continuous magnitudes (Grk., *megethoi*), which have physical dimensions (e.g., length, area, rate), from discrete numbers (Grk., *arithmoi*), which do not (Maziarz and Greenwood 1968). Euclid axiomatizes them separately (magnitudes in Book V, numbers in Book VII), and a mathematical system comprising both discrete and continuous quantities was not achieved until the nineteenth century in the work of Weierstrass and Dedekind.

The earliest known mechanical analog computer is the “Antikythera mechanism,” which was found in 1900 in a shipwreck under the sea near the Greek island of Antikythera (between Kythera and Crete). It dates to the second century BCE and performs astronomical calculations. The device is sophisticated (at least 70 gears) and well engineered, suggesting that it was not the first of its type and therefore that other analog computing

devices may have been used in the ancient Mediterranean world (Freeth et al. 2006). Indeed, according to Cicero (Rep. 22) and other authors, Archimedes (c. 287-c. 212 BCE) and other ancient scientists also built analog computers, such as armillary spheres, for astronomical simulation and computation. Other antique mechanical analog computers include the astrolabe, which is used for determination of longitude and a variety of other astronomical purposes, and the torquetum, which converts astronomical measurements between equatorial, ecliptic, and horizontal coordinates.

A class of special-purpose analog computer, which is simple in conception but may be used for a wide range of purposes, is the nomograph (also, nomogram, alignment chart). In its most common form, it permits the solution of quite arbitrary equations in three real variables,  $f(u, v, w) = 0$ . The nomograph is a chart or graph with scales for each of the variables; typically, these scales are curved and have nonuniform numerical markings. Given values for any two of the variables, a straightedge is laid across their positions on their scales, and the value of the third variable is read off where the straightedge crosses the third scale. Nomographs were used to solve many problems in engineering and applied mathematics. They improve intuitive understanding by allowing the relationships among the variables to be visualized and facilitate exploring their variation by moving the straightedge. Lipka (1918) is an example of a course in graphical and mechanical methods of analog computation, including nomographs and slide rules.

Until the introduction of portable electronic calculators in the early 1970s, the slide rule was the most familiar analog computing device. Slide rules use logarithms for multiplication and division, and they were invented in the early seventeenth century shortly after John Napier's description of logarithms.

The mid-nineteenth century saw the development of the field analogy method by G. Kirchhoff (1824–1887) and others (Kirchhoff 1845). In this approach an electrical field in an electrolytic tank or conductive paper was used to solve two-dimensional boundary problems for temperature distributions and magnetic fields

(Small 2001, p. 34). It is an early example of analog field computation.

In the nineteenth century, a number of mechanical analog computers were developed for integration and differentiation (e.g., Lipka 1918, pp. 246–256; Clymer 1993). For example, the planimeter measures the area under a curve or within a closed boundary. While the operator moves a pointer along the curve, a rotating wheel accumulates the area. Similarly, the integrator is able to draw the integral of a given function as its shape is traced. Other mechanical devices can draw the derivative of a curve or compute a tangent line at a given point.

In the late nineteenth century, William Thomson, Lord Kelvin, constructed several analog computers, including a “tide predictor” and a “harmonic analyzer,” which computed the Fourier coefficients of a tidal curve (Thomson 1878, 1938). In 1876 he described how the mechanical integrators invented by his brother could be connected together in a feedback loop in order to solve second- and higher-order differential equations (Small 2001, pp. 34–35, 42; Thomson 1876). He was unable to construct this differential analyzer, which had to await the invention of the torque amplifier in 1927.

The torque amplifier and other technical advancements permitted Vannevar Bush at MIT to construct the first practical differential analyzer in 1930 (Small 2001, pp. 42–45). It had six integrators and could also do addition, subtraction, multiplication, and division. Input data were entered in the form of continuous curves, and the machine automatically plotted the output curves continuously as the equations were integrated. Similar differential analyzers were constructed at other laboratories in the USA and the UK.

Setting up a problem on the MIT differential analyzer took a long time; gears and rods had to be arranged to define the required dependencies among the variables. Bush later designed a much more sophisticated machine, the Rockefeller Differential Analyzer, which became operational in 1947. With 18 integrators (out of a planned 30), it provided programmatic control of machine setup and permitted several jobs to be run simultaneously. Mechanical differential analyzers were

rapidly supplanted by electronic analog computers in the mid-1950s, and most were disassembled in the 1960s (Bowles 1996; Owens 1986; Small 2001, pp. 50–45).

During World War II, and even later wars, an important application of optical and mechanical analog computation was in “gun directors” and “bomb sights,” which performed ballistic computations to accurately target artillery and dropped ordnance.

### **Electronic Analog Computation in the Twentieth Century**

It is commonly supposed that electronic analog computers were superior to mechanical analog computers, and they were in many respects, including speed, cost, ease of construction, size, and portability (Small 2001, pp. 54–56). On the other hand, mechanical integrators produced higher precision results (0.1% vs. 1% for early electronic devices) and had greater mathematical flexibility (they were able to integrate with respect to any variable, not just time). However, many important applications did not require high precision and focused on dynamic systems for which time integration was sufficient.

Analog computers (nonelectronic as well as electronic) can be divided into active-element and passive-element computers; the former involve some kind of amplification, the latter do not (Truitt and Rogers 1960, pp. 2-1–4). Passive-element computers included the network analyzers, which were developed in the 1920s to analyze electric power distribution networks and which continued in use through the 1950s (Small 2001, pp. 35–40). They were also applied to problems in thermodynamics, aircraft design, and mechanical engineering. In these systems networks or grids of resistive elements or reactive elements (i.e., involving capacitance and inductance as well as resistance) were used to model the spatial distribution of physical quantities such as voltage, current, and power (in electric distribution networks), electrical potential in space, stress in solid materials, temperature (in heat diffusion problems), pressure, fluid flow rate, and wave amplitude (Truitt and Rogers 1960, p. 2-2). That is, network analyzers dealt with partial differential

equations (PDEs), whereas active-element computers, such as the differential analyzer and its electronic successors, were restricted to ordinary differential equations (ODEs) in which time was the independent variable. Large network analyzers are early examples of analog field computers.

Electronic analog computers became feasible after the invention of the DC operational amplifier (“op amp”) c. 1940 (Small 2001, pp. 64, 67–72). Already in the 1930s scientists at Bell Telephone Laboratories (BTL) had developed the DC-coupled feedback-stabilized amplifier, which is the basis of the op amp. In 1940, as the USA prepared to enter World War II, DL Parkinson at BTL had a dream in which he saw DC amplifiers being used to control an antiaircraft gun. As a consequence, with his colleagues CA Lovell and BT Weber, he wrote a series of papers on “electrical mathematics,” which described electrical circuits to “operationalize” addition, subtraction, integration, differentiation, etc. The project to produce an electronic gun director led to the development and refinement of DC op amps suitable for analog computation.

The wartime work at BTL was focused primarily on control applications of analog devices, such as the gun director. Other researchers, such as Lakatos at BTL, were more interested in applying them to general-purpose analog computation for science and engineering, which resulted in the design of the general-purpose analog computer (GPAC), called “Gypsy” and completed in 1949 (Small 2001, pp. 69–71). Building on the BTL op amp design, fundamental work on electronic analog computation was conducted at Columbia University in the 1940s. In particular, this research showed how analog computation could be applied to the simulation of dynamic systems and to the solution of nonlinear equations.

Commercial general-purpose analog computers (GPACs) emerged in the late 1940s and early 1950s (Small 2001, pp. 72–73). Typically, they provided several dozen integrators, but several GPACs could be connected together to solve larger problems. Later, large-scale GPACs might have up to 500 amplifiers and compute with

0.01–0.1% precision (Truitt and Rogers 1960, pp. 2–33).

Besides integrators, typical GPACs provided adders, subtracters, multipliers, fixed function generators (e.g., logarithms, exponentials, trigonometric functions), and variable function generators (for user-defined functions) (Truitt and Rogers 1960, Chaps. 1.3 and 2.4). A GPAC was programmed by connecting these components together, often by means of a patch panel. In addition, parameters could be controlled by adjusting potentiometers (attenuators), and arbitrary functions could be entered in the form of graphs (Truitt and Rogers 1960, pp. 1-72–81, 2-154–156). Output devices plotted data continuously or displayed it numerically (Truitt and Rogers 1960, pp. 3-1–30).

The most basic way of using a GPAC was in single-shot mode (Weyrick 1969, pp. 168–170). First, parameters and initial values were entered into the potentiometers. Next, putting a master switch in “reset” mode controlled relays to apply the initial values to the integrators. Turning the switch to “operate” or “compute” mode allowed the computation to take place (i.e., the integrators to integrate). Finally, placing the switch in “hold” mode stopped the computation and stabilized the values, allowing them to be read from the computer (e.g., on voltmeters). Although single-shot operation was also called “slow operation” (in comparison to “repetitive operation,” discussed next), it was in practice quite fast. Because all of the devices computed in parallel and at electronic speeds, analog computers usually solved problems in real time but often much faster (Truitt and Rogers 1960, pp. 1-30–32; Small 2001, p. 72).

One common application of GPACs was to explore the effect of one or more parameters on the behavior of a system. To facilitate this exploration of the parameter space, some GPACs provided a repetitive operation mode, which worked as follows (Weyrick 1969, p. 170; Small 2001, p. 72). An electronic clock switched the computer between reset and compute modes at an adjustable rate (e.g., 10–1,000 cycles per second) (Ashley 1963). In effect the simulation was rerun at the clock rate, but if any parameters were adjusted, the simulation results would vary along with them. Therefore, within a few seconds, an entire

family of related simulations could be run. More importantly, the operator could acquire an intuitive understanding of the system’s dependence on its parameters.

### The Eclipse of Analog Computing

It is commonly supposed that electronic analog computers were a primitive predecessor of the digital computer and that their use was just a historical episode, or even a digression, in the inevitable triumph of digital technology. It is supposed that the current digital hegemony is a simple matter of technological superiority. However, the history is much more complicated and involves a number of social, economic, historical, pedagogical, and also technical factors, which are outside the scope of this article (see Small 1993, 2001, especially Chap. 8, for more information). In any case, beginning after World War II and continuing for 25 years, there was lively debate about the relative merits of analog and digital computation.

Speed was an oft-cited advantage of analog computers (Small 2001, Chap. 8). While early digital computers were much faster than mechanical differential analyzers, they were slower (often by several orders of magnitude) than electronic analog computers. Furthermore, although digital computers could perform individual arithmetic operations rapidly, complete problems were solved sequentially, one operation at a time, whereas analog computers operated in parallel. Thus, it was argued that increasingly large problems required more time to solve on a digital computer, whereas on an analog computer, they might require more hardware but not more time. Even as digital computing speed was improved, analog computing retained its advantage for several decades, but this advantage eroded steadily.

Another important issue was the comparative precision of digital and analog computation (Small 2001, Chap. 8). Analog computers typically computed with three or four digits of precision, and it was very expensive to do much better, due to the difficulty of manufacturing the parts and other factors. In contrast, digital computers could perform arithmetic operations with many digits of precision, and the hardware cost was

approximately proportional to the number of digits. Against this, analog computing advocates argued that many problems did not require such high precision because the measurements were known to only a few significant figures and the mathematical models were approximations. Further, they distinguished between precision and accuracy, which refers to the conformity of the computation to physical reality, and they argued that digital computation was often less accurate than analog due to numerical limitations (e.g., truncation, cumulative error in numerical integration). Nevertheless, some important applications, such as the calculation of missile trajectories, required greater precision, and for these, digital computation had the advantage. Indeed, to some extent precision was viewed as inherently desirable, even in applications where it was unimportant, and it was easily mistaken for accuracy. (See section “[Precision](#)” for more on precision and accuracy.)

There was even a social factor involved, in that the written programs, precision, and exactness of digital computation were associated with mathematics and science, but the hands-on operation, parameter variation, and approximate solutions of analog computation were associated with engineering, and so analog computing inherited “the lower status of engineering vis-à-vis science” (Small 2001, p. 251). Thus, the status of digital computing was further enhanced as engineering became more mathematical and scientific after World War II (Small 2001, pp. 247–251).

Already by the mid-1950s, the competition between analog and digital had evolved into the idea that they were complementary technologies. This resulted in the development of a variety of hybrid analog/digital computing systems (Small 2001, pp. 251–253, 263–266). In some cases, this involved using a digital computer to control an analog computer by using digital logic to connect the analog computing elements, to set parameters, and to gather data. This improved the accessibility and usability of analog computers but had the disadvantage of distancing the user from the physical analog system. The intercontinental ballistic missile program in the USA stimulated the further development of hybrid computers in the late

1950s and 1960s (Small 1993). These applications required the speed of analog computation to simulate the closed-loop control systems and the precision of digital computation for accurate computation of trajectories. However, by the early 1970s hybrids were being displaced by all digital systems. Certainly, part of the reason was the steady improvement in digital technology, driven by a vibrant digital computer industry, but contemporaries also pointed to an inaccurate perception that analog computing was obsolete and to a lack of education about the advantages and techniques of analog computing.

Another argument made in favor of digital computers was that they were general purpose, since they could be used in business data processing and other application domains, whereas analog computers were essentially special purpose, since they were limited to scientific computation (Small 2001, pp. 248–250). Against this, it was argued that all computing is essentially computing by analogy, and therefore, analog computation was general purpose because the class of analog computers included digital computers! (See also section “[Definition of the Subject](#)” on computing by analogy.) Be that as it may, analog computation, as normally understood, is restricted to continuous variables, and so it was not immediately applicable to discrete data, such as that manipulated in business computing and other nonscientific applications. Therefore, business (and eventually consumer) applications motivated the computer industry’s investment in digital computer technology at the expense of analog technology.

Although it is commonly believed that analog computers quickly disappeared after digital computers became available, this is inaccurate, for both general-purpose and special-purpose analog computers have continued to be used in specialized applications to the present time. For example, a general-purpose electrical (vs. electronic) analog computer, the Anacom, was still in use in 1991. This is not technological atavism, for “there is no doubt considerable truth in the fact that Anacom continued to be used because it effectively met a need in a historically neglected but nevertheless important computer application area” (Aspray 1993). As mentioned, the reasons

for the eclipse of analog computing were not simply the technological superiority of digital computation; the conditions were much more complex. Therefore, a change in conditions has necessitated a reevaluation of analog technology.

### Analog VLSI

In the mid-1980s, Carver Mead, who already had made important contributions to digital VLSI technology, began to advocate for the development of analog VLSI (Mead 1987, 1989). His motivation was that “the nervous system of even a very simple animal contains computing paradigms that are orders of magnitude more effective than are those found in systems made by humans” and that they “can be realized in our most commonly available technology – silicon integrated circuits” (Mead 1989, pp. xi). However, he argued, since these natural computation systems are analog and highly nonlinear, progress would require understanding neural information processing in animals and applying it in a new analog VLSI technology.

Because analog computation is closer to the physical laws by which all computation is realized (which are continuous), analog circuits often use fewer devices than corresponding digital circuits. For example, a four-quadrant adder (capable of adding two signed numbers) can be fabricated from four transistors (Mead 1989, pp. 87–88), and a four-quadrant multiplier from 9 to 17, depending on the required range of operation (Mead 1989, pp. 90–96). Intuitions derived from digital logic about what is simple or complex to compute are often misleading when applied to analog computation. For example, two transistors are sufficient to compute the logarithm or exponential, five for the hyperbolic tangent (which is very useful in neural computation), and three for the square root (Mead 1989, pp. 70–71, 97–99). Thus, analog VLSI is an attractive approach to “post-Moore’s Law computing” (see section “Future Directions” below). Mead and his colleagues demonstrated a number of analog VLSI devices inspired by the nervous system, including a “silicon retina” and an “electronic cochlea” (Mead 1989, Chaps. 15 and 16), research that has lead to a renaissance of interest in electronic analog computing.

### Field Programmable Analog Arrays

Field programmable analog arrays (FPAs) permit the programming of analog VLSI systems analogously to the use of field programmable gate arrays (FPGAs) for digital systems (Basu et al. 2010). An FPA comprises a number of identical computational analog blocks (CABs), each of which contains a small number of analog computing elements. Programmable switching matrices control the interconnections among the elements of a CAB and the interconnections between the CABs. Contemporary FPAs make use of floating-gate transistors, in which the gate has no DC connection to other circuit elements and thus is able to hold a charge indefinitely. Therefore, the floating gate can be used to store a continuous value that governs the impedance of the transistor by several orders of magnitude. The gate charge can be changed by processes such as electron tunneling, which increases the charge, and hot-electron injection, which decreases it. Digital decoders allow individual floating-gate transistors in the switching matrices to be addressed and programmed. At the extremes of zero and infinite impedance, the transistors operate as perfect switches, connecting or disconnecting circuit elements.

Programming the connections to these extreme values is time consuming, however, and so in practice some trade-off is made between programming time and switch impedance. Each CAB contains several operational transconductance amplifiers (OTAs), which are op amps whose gain is controlled by a bias current. They are the principal analog computing elements, since they can be used for operations such as integration, differentiation, and gain amplification. Other computing elements may include tunable bandpass filters, which can be used for Fourier signal processing, and small matrix–vector multipliers, which can be used to implement linear operators. Current FPAs can compute with a resolution of ten bits (precision of  $10^{-3}$ ).

### Nonelectronic Analog Computation

As will be explained in the body of this article, analog computation suggests many opportunities for future computing technologies. Many physical

phenomena are potential media for analog computation provided they have useful mathematical structure (i.e., the mathematical laws describing them are mathematical functions useful for general- or special-purpose computation), and they are sufficiently controllable for practical use.

## Article Road Map

The remainder of this article will begin by summarizing the fundamentals of analog computing, starting with the continuous state space and the various processes by which analog computation can be organized in time. Next, it will discuss analog computation in nature, which provides models and inspiration for many contemporary uses of analog computation, such as neural networks. Then, we consider general-purpose analog computing, both from a theoretical perspective and in terms of practical general-purpose analog computers (GPACs). This leads to a discussion of the theoretical power of analog computation and in particular to the issue of whether analog computing is in some sense more powerful than digital computing. We briefly consider the cognitive aspects of analog computing, and whether it leads to a different approach to computation than does digital computing. Finally, we conclude with some observations on the role of analog computation in “post-Moore’s Law computing.”

## Fundamentals of Analog Computing

### Continuous State Space

As discussed in section “[Introduction](#),” the fundamental characteristic that distinguishes analog from digital computation is that the state space is continuous in analog computation and discrete in digital computation. Therefore, it might be more accurate to call analog and digital computation continuous and discrete computation, respectively. Furthermore, since the earliest days, there have been hybrid computers that combine continuous and discrete state spaces and processes. Thus, there are several respects in which the state space may be continuous.

In the simplest case, the state space comprises a finite (generally modest) number of variables, each holding a continuous quantity (e.g., voltage, current, charge). In a traditional GPAC, they correspond to the variables in the ODEs defining the computational process, each typically having some independent meaning in the analysis of the problem. Mathematically, the variables are taken to contain bounded real numbers, although complex-valued variables are also possible (e.g., in AC electronic analog computers). In a practical sense, however, their precision is limited by noise, stability, device tolerance, and other factors (discussed below, section “[Characteristics of Analog Computation](#)”).

In typical analog neural networks, the state space is larger in dimension but more structured than in the former case. The artificial neurons are organized into one or more layers, each composed of a (possibly large) number of artificial neurons. Commonly, each layer of neurons is densely connected to the next layer. In general, the layers each have some meaning in the problem domain, but the individual neurons constituting them do not (and so, in mathematical descriptions, the neurons are typically numbered rather than named).

The individual artificial neurons usually perform a simple computation such as this:

$$y = \sigma(s), \text{ where } s = b + \sum_{i=1}^n w_i x_i,$$

where  $y$  is the activity of the neuron,  $x_1, \dots, x_n$  are the activities of the neurons that provide its inputs,  $b$  is a bias term, and  $w_1, \dots, w_n$  are the weights or strengths of the connections. Often, the activation function  $\sigma$  is a real-valued sigmoid (“S-shaped”) function, such as the logistic sigmoid,

$$\sigma(s) = \frac{1}{1 + e^{-s}},$$

in which case the neuron activity  $y$  is a real number, but some applications use a discontinuous threshold function, such as the Heaviside function,

$$U(s) = \begin{cases} +1 & \text{if } s \geq 0 \\ 0 & \text{if } s < 0 \end{cases}$$

in which case the activity is a discrete quantity. The saturated-linear or piecewise-linear sigmoid is also used occasionally:

$$\sigma(s) = \begin{cases} +1 & \text{if } s > 1 \\ s & \text{if } 0 \leq s \leq 1 \\ 0 & \text{if } s < 0. \end{cases}$$

Regardless of whether the activation function is continuous or discrete, the bias  $b$  and connection weights  $w_1, \dots, w_n$  are real numbers, as is the “net input”  $s = \sum_i w_i x_i$  to the activation function. Analog computation may be used to evaluate the linear combination  $s$  and the activation function  $\sigma(s)$ , if it is real valued. The biases and weights are normally determined by a learning algorithm (e.g., back-propagation), which is also a good candidate for analog implementation.

In summary, the continuous state space of a neural network includes the bias values and net inputs of the neurons and the interconnection strengths between the neurons. It also includes the activity values of the neurons, if the activation function is a real-valued sigmoid function, as is often the case. Often, large groups (“layers”) of neurons (and the connections between these groups) have some intuitive meaning in the problem domain, but typically, the individual neuron activities, bias values, and interconnection weights do not.

If we extrapolate the number of neurons in a layer to the continuum limit, we get a field, which may be defined as a continuous distribution of continuous quantity. Treating a group of artificial or biological neurons as a continuous mass is a reasonable mathematical approximation if their number is sufficiently large and if their spatial arrangement is significant (as it generally is in the brain). Fields are especially useful in modeling cortical maps, in which information is represented by the pattern of activity over a region of neural cortex.

In field computation, the state space is continuous in two ways: it is continuous in variation but

also in space. Therefore, field computation is especially applicable to solving PDEs and to processing spatially extended information such as visual images. Some early analog computing devices were capable of field computation (Truitt and Rogers 1960, pp. 1-14–1-17, 2-2–2-16). For example, as previously mentioned (section “[Introduction](#)”), large resistor and capacitor networks could be used for solving PDEs such as diffusion problems. In these cases, a discrete ensemble of resistors and capacitors was used to approximate a continuous field, while in other cases, the computing medium was spatially continuous. The latter made use of conductive sheets (for two-dimensional fields) or electrolytic tanks (for two- or three-dimensional fields). When they were applied to steady-state spatial problems, these analog computers were called field plotters or potential analyzers.

The ability to fabricate very large arrays of analog computing devices, combined with the need to exploit massive parallelism in real-time computation and control applications, creates new opportunities for field computation (MacLennan 1987, 1990, 1999). There is also renewed interest in using physical fields in analog computation. For example, Rubel (1993) defined an abstract extended analog computer (EAC), which augments Shannon’s (1993) general-purpose analog computer with (unspecified) facilities for field computation, such as PDE solvers (see sections “[Shannon’s Analysis](#)” and “[Rubel’s Extended Analog Computer](#)” below). JW Mills has explored the practical application of these ideas in his artificial neural field networks and VLSI EACs, which use the diffusion of electrons in bulk silicon or conductive gels and plastics for 2D and 3D field computation (Mills 1996; Mills et al. 2006).

## Computational Process

We have considered the continuous state space, which is the basis for analog computing, but there are a variety of ways in which analog computers can operate on the state. In particular, the state can change continuously in time or be updated at distinct instants (as in digital computation).

## Continuous Time

Since the laws of physics on which analog computing is based are differential equations, many analog computations proceed in continuous real time. Also, as we have seen, an important application of analog computers in the late nineteenth and early twentieth centuries was the integration of ODEs in which time is the independent variable. A common technique in analog simulation of physical systems is time scaling, in which the differential equations are altered systematically so the simulation proceeds either more slowly or more quickly than the primary system (see section “[Characteristics of Analog Computation](#)” for more on time scaling). On the other hand, because analog computations are close to the physical processes that realize them, analog computing is rapid, which makes it very suitable for real-time control applications.

In principle, any mathematically describable physical process operating on time-varying physical quantities can be used for analog computation. In practice, however, analog computers typically provide familiar operations that scientists and engineers use in differential equations (Rogers and Connolly 1960; Truitt and Rogers 1960). These include basic arithmetic operations, such as algebraic sum and difference ( $u(t) = v(t) \pm w(t)$ ), constant multiplication or scaling ( $u(t) = cv(t)$ ), variable multiplication and division ( $u(t) = v(t)w(t)$ ,  $u(t) = v(t)/w(t)$ ), and inversion ( $u(t) = -v(t)$ ). Transcendental functions may be provided, such as the exponential ( $u(t) = \exp v(t)$ ), logarithm ( $u(t) = \ln v(t)$ ), trigonometric functions ( $u(t) = \sin v(t)$ , etc.), and resolvers for converting between polar and rectangular coordinates. Most important, of course, is definite integration ( $u(t) = v_0 + \int_0^t v(\tau)d\tau$ ), but differentiation may also be provided ( $u(t) = \dot{v}(t)$ ). Generally, however, direct differentiation is avoided, since noise tends to have a higher frequency than the signal, and therefore, differentiation amplifies noise; typically, problems are reformulated to avoid direct differentiation (Weyrick 1969, pp. 26–27). As previously mentioned, many GPACs include (arbitrary) function generators, which allow the use of functions defined only by a graph and for which no mathematical definition might be available; in this way empirically defined functions can be used (Rogers and Connolly 1960,

pp. 32–42). Thus, given a graph  $(x, f(x))$ , or a sufficient set of samples,  $(x_k, f(x_k))$ , the function generator approximates  $u(t) = f(v(t))$ . Rather less common are generators for arbitrary functions of two variables,  $u(t) = f(v(t), w(t))$ , in which the function may be defined by a surface,  $(x, y, f(x, y))$ , or by sufficient samples from it.

Although analog computing is primarily continuous, there are situations in which discontinuous behavior is required. Therefore, some analog computers provide comparators, which produce a discontinuous result depending on the relative value of two input values. For example,

$$u = \begin{cases} k & \text{if } v \geq w, \\ 0 & \text{if } v < w. \end{cases}$$

Typically, this would be implemented as a Heaviside (unit step) function applied to the difference of the inputs,  $u = k U(v - w)$ . In addition to allowing the definition of discontinuous functions, comparators provide a primitive decision-making ability and may be used, for example, to terminate a computation (switching the computer from “operate” to “hold” mode).

Other operations that have proved useful in analog computation are time delays and noise generators (Howe 1961, Chap. 7). The function of a time delay is simply to retard the signal by an adjustable delay  $T > 0$ , that is,  $u(t+T) = v(t)$ . One common application is to model delays in the primary system (e.g., human response time).

Typically, a noise generator produces time-invariant Gaussian-distributed noise with zero mean and a flat power spectrum (over a band compatible with the analog computing process). The standard deviation can be adjusted by scaling, the mean can be shifted by addition, and the spectrum can be altered by filtering, as required by the application. Historically, noise generators were used to model noise and other random effects in the primary system, to determine, for example, its sensitivity to effects such as turbulence. However, noise can make a positive contribution in some analog computing algorithms (e.g., for symmetry breaking and in simulated annealing, weight perturbation learning, and stochastic resonance).

As already mentioned, some analog computing devices for the direct solution of PDEs have been developed. In general, a PDE solver depends on an analogous physical process, that is, on a process obeying the same class of PDEs that it is intended to solve. For example, in Mill's EAC, diffusion of electrons in conductive sheets or solids is used to solve diffusion equations (Mills 1996; Mills et al. 2006). Historically, PDEs were solved on electronic GPACs by discretizing all but one of the independent variables, thus replacing the differential equations by difference equations (Rogers and Connolly 1960, pp. 173–193). That is, computation over a field was approximated by computation over a finite real array.

Reaction-diffusion computation is an important example of continuous-time analog computing (► “Reaction-Diffusion Computing”). The state is represented by a set of time-varying chemical concentration fields,  $c_1, \dots, c_n$ . These fields are distributed across a one-, two-, or three-dimensional space  $\Omega$  so that, for  $x \in \Omega$ ,  $c_k(x, t)$  represents the concentration of chemical  $k$  at location  $x$  and time  $t$ . Computation proceeds in continuous time according to reaction-diffusion equations, which have the form:

$$\partial \mathbf{c} / \partial t = D \nabla^2 \mathbf{c} + \mathbf{F}(\mathbf{c}),$$

where  $\mathbf{c} = (c_1, \dots, c_n)^T$  is the vector of concentrations,  $D = \text{diag}(d_1, \dots, d_n)$  is a diagonal matrix of positive diffusion rates, and  $\mathbf{F}$  is a nonlinear vector function that describes how the chemical reactions affect the concentrations.

Some neural net models operate in continuous time and thus are examples of continuous-time analog computation. For example, Grossberg (1967, 1973, 1976) defines the activity of a neuron by differential equations such as this:

$$\begin{aligned} \dot{x}_i &= -a_i x_i + \sum_{j=1}^n b_{ij} w_{ij}^{(+)} f_j(x_j) \\ &\quad - \sum_{j=1}^n c_{ij} w_{ij}^{(-)} g_j(x_j) + I_i. \end{aligned}$$

This describes the continuous change in the activity of neuron  $i$  resulting from passive decay

(first term), positive feedback from other neurons (second term), negative feedback (third term), and input (last term). The  $f_j$  and  $g_j$  are nonlinear activation functions, and the  $w_{ij}^{(+)}$  and  $w_{ij}^{(-)}$  are adaptable excitatory and inhibitory connection strengths, respectively.

The continuous Hopfield network is another example of continuous-time analog computation (Hopfield 1984). The output  $y_i$  of a neuron is a nonlinear function of its internal state  $x_i$ ,  $y_i = \sigma(x_i)$ , where the hyperbolic tangent is usually used as the activation function,  $\sigma(x) = \tanh x$ , because its range is  $[-1, 1]$ . The internal state is defined by a differential equation:

$$\tau_i \dot{x}_i = -a_i x_i + b_i + \sum_{j=1}^n w_{ij} y_j,$$

where  $\tau_i$  is a time constant,  $a_i$  is the decay rate,  $b_i$  is the bias, and  $w_{ij}$  is the connection weight to neuron  $i$  from neuron  $j$ . In a Hopfield network, every neuron is symmetrically connected to every other ( $w_{ij} = w_{ji}$ ) but not to itself ( $w_{ii} = 0$ ).

Of course analog VLSI implementations of neural networks also operate in continuous time (e.g., Fakhraie and Smith 1997; Mead 1989). Concurrent with the resurgence of interest in analog computation have been innovative reconceptualizations of continuous-time computation. For example, Brockett (1988) has shown that dynamical systems can solve certain problems normally considered to be intrinsically sequential. For example, a certain system of ODEs (a nonperiodic finite Toda lattice) can sort a list of numbers by continuous-time analog computation. The system is started with the vector  $\mathbf{x}$  equal to the values to be sorted and a vector  $\mathbf{y}$  initialized to small nonzero values; the  $\mathbf{y}$  vector converges to a sorted permutation of  $\mathbf{x}$ .

Analog computation can be used for solving discrete constraint satisfaction problems, such as Boolean satisfiability, a well-known NP-complete problem. Given a Boolean expression with  $N$  variables, the problem is to determine an assignment of Boolean values to the variables that will make the expression true (if such an assignment exists). Without loss of generality, the expression

is a conjunction of  $M$  clauses, each of which is a disjunction of  $k > 2$  literals. A literal is either a plain variable or a negated variable. Ercsey-Ravasz and her colleagues have designed continuous time dynamical systems for which the only attractors are solutions to Boolean satisfiability problems (Ercsey-Ravasz and Toroczkai 2011; Molnár and Ercsey-Ravasz 2013). A particular instance to be solved is defined by an  $M \times N$  matrix  $\mathbf{C}$  in which  $C_{mi}$  is  $+1$  if variable  $i$  is in clause  $m$ ,  $-1$  if it is negated in clause  $m$ , and  $0$  if it does not occur in clause  $m$ .  $N$  continuous variables  $s_1, \dots, s_N \in [-1, +1]$  represent the Boolean variables, which will approach  $+1$  for a true value, and  $-1$  for a false value. In addition, there are  $M$  continuous auxiliary variables  $a_1, \dots, a_M \in [0, 1]$ , one for each clause, which will represent the “urgency” of satisfying an individual clause. The dynamics of the  $s_i$  variables are given by

$$\dot{s}_i(t) = -s_i(t) + Af[s_i(t)] + \sum_{m=1}^M C_{mi}g[a_m(t)],$$

where  $A$  is a constant, and  $f$  and  $g$  are linear squashing functions. The dynamics of the auxiliary variables is defined

$$\dot{a}_m(t) = -a_m(t) + Bg[a_m(t)] - \sum_{i=1}^N c_{mi}f[s_i(t)] + 1 - k,$$

where  $B$  is a constant. These differential equations are implemented by the analog program shown in Fig. 1 with  $M + N$  integrators for the  $a_m$  and  $s_i$  variables. Analog computations such as these are suitable for implementation in analog electronics (Basford et al. 2016; Yin et al. 2016).

### Sequential Time

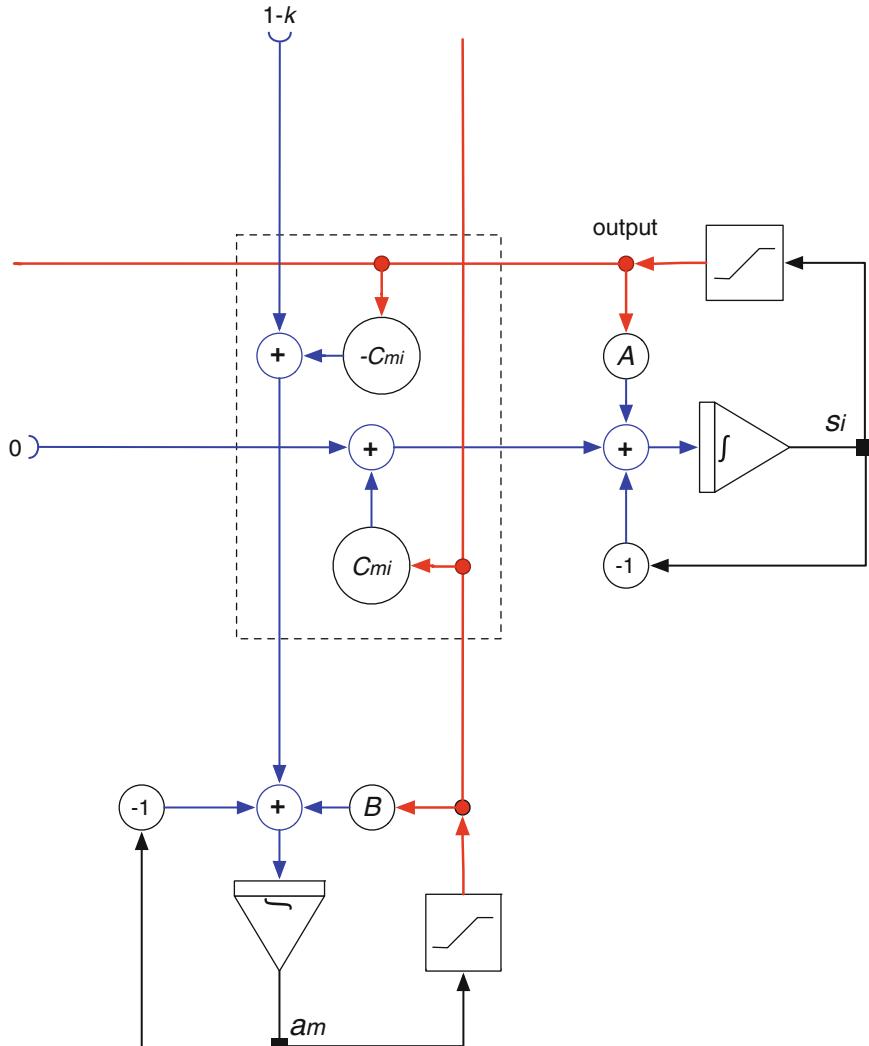
Sequential-time computation refers to computation in which discrete computational operations take place in succession but at no definite interval (van Gelder 1997). Ordinary digital computer programs take place in sequential time, for the operations occur one after another, but the individual operations are not required to have any specific duration, so long as they take finite time.

One of the oldest examples of sequential analog computation is provided by the compass-and-straightedge constructions of traditional Euclidean geometry (section “Introduction”). These computations proceed by a sequence of discrete operations, but the individual operations involve continuous representations (e.g., compass settings, straightedge positions) and operate on a continuous state (the figure under construction). Slide rule calculation might seem to be an example of sequential analog computation, but if we look at it, we see that although the operations are performed by an analog device, the intermediate results are recorded digitally (and so this part of the state space is discrete). Thus, it is a kind of hybrid computation.

The familiar digital computer automates sequential digital computations that once were performed manually by human “computers.” Sequential analog computation can be similarly automated. That is, just as the control unit of an ordinary digital computer sequences digital computations, so a digital control unit can sequence analog computations. In addition to the analog computation devices (adders, multipliers, etc.), such a computer must provide variables and registers capable of holding continuous quantities between the sequential steps of the computation (see also section “Discrete Time” below).

The primitive operations of sequential-time analog computation are typically similar to those in continuous-time computation (e.g., addition, multiplication, transcendental functions), but integration and differentiation with respect to sequential time do not make sense. However, continuous-time integration within a single step, and space-domain integration, as in PDE solvers or field computation devices, is compatible with sequential analog computation.

In general, any model of digital computation can be converted to a similar model of sequential analog computation by changing the discrete state space to a continuum and making appropriate changes to the rest of the model. For example, we can make an analog Turing machine by allowing it to write a bounded real number (rather than a symbol from a finite alphabet) onto a tape cell. The Turing machine’s finite control can be altered to test for tape markings in some specified range.



**Analog Computation, Fig. 1** Analog program for Boolean satisfiability

Similarly, in a series of publications, Blum, Shub, and Smale developed a theory of computation over the reals, which is an abstract model of sequential-time analog computation (Blum et al. 1988, 1998). In this “BSS model,” programs are represented as flowcharts, but they are able to operate on real-valued variables. Using this model, they were able to prove a number of theorems about the complexity of sequential analog algorithms.

The BSS model, and some other sequential analog computation models, assumes that it is possible to make exact comparisons between real

numbers (analogous to exact comparisons between integers or discrete symbols in digital computation) and to use the result of the comparison to control the path of execution. Comparisons of this kind are problematic because they imply infinite precision in the comparator (which may be defensible in a mathematical model but is impossible in physical analog devices) and because they make the execution path a discontinuous function of the state (whereas analog computation is usually continuous). Indeed, it has been argued that this is not “true” analog computation (Siegelmann 1999, p. 148).

Many artificial neural network models are examples of sequential-time analog computation. In a simple feed-forward neural network, an input vector is processed by the layers in order, as in a pipeline. That is, the output of layer  $n$  becomes the input of layer  $n + 1$ . Since the model does not make any assumptions about the amount of time it takes a vector to be processed by each layer and to propagate to the next, execution takes place in sequential time. Most recurrent neural networks, which have feedback, also operate in sequential time, since the activities of all the neurons are updated synchronously (that is, signals propagate through the layers, or back to earlier layers, in lockstep).

Many artificial neural net learning algorithms are also sequential-time analog computations. For example, the back-propagation algorithm updates a network's weights, moving sequentially backward through the layers.

In summary, the correctness of sequential-time computation (analog or digital) depends on the order of operations, not on their duration, and similarly, the efficiency of sequential computations is evaluated in terms of the number of operations, not on their total duration.

### Discrete Time

Discrete-time analog computation has similarities to both continuous-time and sequential analog computation. Like the latter, it proceeds by a sequence of discrete (analog) computation steps; like the former, these steps occur at a constant rate in real time (e.g., some “sample rate”). If the real-time rate is sufficient for the application, then discrete-time computation can approximate continuous-time computation (including integration and differentiation).

Some electronic GPACs implemented discrete-time analog computation by a modification of repetitive operation mode, called iterative analog computation (Ashley 1963, Chap. 9). Recall (section “[Electronic Analog Computation in the Twentieth Century](#)”) that in repetitive operation mode, a clock rapidly switched the computer between reset and compute modes, thus repeating the same analog computation, but with different parameters (set by the operator). However, each

repetition was independent of the others. Iterative operation was different in that analog values computed by one iteration could be used as initial values in the next. This was accomplished by means of an analog memory circuit (based on an op amp) that sampled an analog value at the end of one compute cycle (effectively during hold mode) and used it to initialize an integrator during the following reset cycle. (A modified version of the memory circuit could be used to retain a value over several iterations.) Iterative computation was used for problems such as determining, by iterative search or refinement, the initial conditions that would lead to a desired state at a future time. Since the analog computations were iterated at a fixed clock rate, iterative operation is an example of discrete-time analog computation. However, the clock rate is not directly relevant in some applications (such as the iterative solution of boundary-value problems), in which case iterative operation is better characterized as sequential analog computation.

The principal contemporary examples of discrete-time analog computing are in neural network applications to time-series analysis and (discrete-time) control. In each of these cases, the input to the neural net is a sequence of discrete-time samples, which propagate through the net and generate discrete-time output signals. Many of these neural nets are recurrent, that is, values from later layers are fed back into earlier layers, which allows the net to remember information from one sample to the next.

### Analog Computer Programs

The concept of a program is central to digital computing, both practically, for it is the means for programming general-purpose digital computers, and theoretically, for it defines the limits of what can be computed by a universal machine, such as a universal Turing machine. Therefore, it is important to discuss means for describing or specifying analog computations.

Traditionally, analog computers were used to solve ODEs (and sometimes PDEs), and so in one sense, a mathematical differential equation is one

way to represent an analog computation. However, since the equations were usually not suitable for direct solution on an analog computer, the process of programming involved the translation of the equations into a schematic diagram showing how the analog computing devices (integrators, etc.) should be connected to solve the problem. These diagrams are the closest analogies to digital computer programs and may be compared to flowcharts, which were once popular in digital computer programming. It is worth noting, however, that flowcharts (and ordinary computer programs) represent sequences among operations, whereas analog computing diagrams represent functional relationships among variables and therefore a kind of parallel data flow.

Differential equations and schematic diagrams are suitable for continuous-time computation, but for sequential analog computation, something more akin to a conventional digital program can be used. Thus, as previously discussed (section “[Sequential Time](#)”), the BSS system uses flowcharts to describe sequential computations over the reals. Similarly, Moore (1996) defines recursive functions over the reals by means of a notation similar to a programming language.

In principle any sort of analog computation might involve constants that are arbitrary real numbers, which therefore might not be expressible in finite form (e.g., as a finite string of digits). Although this is of theoretical interest (see section “[Real-Valued Inputs, Outputs, and Constants](#)” below), from a practical standpoint, these constants could be set with about at most four digits of precision (Rogers and Connolly 1960, p. 11). Indeed, automatic potentiometer-setting devices were constructed that read a series of decimal numerals from punched paper tape and used them to set the potentiometers for the constants (Truitt and Rogers 1960, pp. 3-58–3-60). Nevertheless, it is worth observing that analog computers do allow continuous inputs that need not be expressed in digital notation, for example, when the parameters of a simulation are continuously varied by the operator. In principle, therefore, an analog program can incorporate constants that are represented by a real-valued physical quantity (e.g., an angle or a distance), which

need not be expressed digitally. Further, as we have seen (section “[Electronic Analog Computation in the Twentieth Century](#)”), some electronic analog computers could compute a function by means of an arbitrarily drawn curve, that is, not represented by an equation or a finite set of digitized points. Therefore, in the context of analog computing, it is natural to expand the concept of a program beyond discrete symbols to include continuous representations (scalar magnitudes, vectors, curves, shapes, surfaces, etc.).

Typically, such continuous representations would be used as adjuncts to conventional discrete representations of the analog computational process, such as equations or diagrams. However, in some cases the most natural static representation of the process is itself continuous, in which case it is more like a “guiding image” than a textual prescription (MacLennan 1995). A simple example is a potential surface, which defines a continuum of trajectories from initial states (possible inputs) to fixed-point attractors (the results of the computations). Such a “program” may define a deterministic computation (e.g., if the computation proceeds by gradient descent), or it may constrain a nondeterministic computation (e.g., if the computation may proceed by any potential-decreasing trajectory). Thus, analog computation suggests a broadened notion of programs and programming.

## Characteristics of Analog Computation

### Precision

Analog computation is evaluated in terms of both accuracy and precision, but the two must be distinguished carefully (Ashley 1963, pp. 25–28; Weyrick 1969, pp. 12–13; Small 2001, pp. 257–261). Accuracy refers primarily to the relationship between a simulation and the primary system it is simulating or, more generally, to the relationship between the results of a computation and the mathematically correct result. Accuracy is a result of many factors, including the mathematical model chosen, the way it is set up on a computer, and the precision of the analog computing devices. Precision, therefore, is a narrower

notion, which refers to the quality of a representation or computing device. In analog computing, precision depends on resolution (fineness of operation) and stability (absence of drift) and may be measured as a fraction of the represented value. Thus, a precision of 0.01% means that the representation will stay within 0.01% of the represented value for a reasonable period of time. For purposes of comparing analog devices, the precision is usually expressed as a fraction of full-scale variation (i.e., the difference between the maximum and minimum representable values).

It is apparent that the precision of analog computing devices depends on many factors. One is the choice of physical process and the way it is utilized in the device. For example, a linear mathematical operation can be realized by using a linear region of a nonlinear physical process, but the realization will be approximate and have some inherent imprecision. Also, associated, unavoidable physical effects (e.g., loading, and leakage and other losses) may prevent precise implementation of an intended mathematical function. Further, there are fundamental physical limitations to resolution (e.g., quantum effects, thermal noise, diffraction). Noise is inevitable, both intrinsic (e.g., thermal noise) and extrinsic (e.g., ambient radiation). Changes in ambient physical conditions, such as temperature, can affect the physical processes and decrease precision. At slower time scales, materials and components age and their physical characteristics change. In addition, there are always technical and economic limits to the control of components, materials, and processes in analog device fabrication.

The precision of analog and digital computing devices depends on very different factors. The precision of a (binary) digital device depends on the number of bits, which influences the amount of hardware, but not its quality. For example, a 64-bit adder is about twice the size of a 32-bit adder, but can be made out of the same components. At worst, the size of a digital device might increase with the square of the number of bits of precision. This is because binary digital devices only need to represent two states, and therefore, they can operate in saturation. The fabrication standards sufficient for the first bit of precision

are also sufficient for the 64th bit. Analog devices, in contrast, need to be able to represent a continuum of states precisely. Therefore, the fabrication of high-precision analog devices is much more expensive than low-precision devices, since the quality of components, materials, and processes must be much more carefully controlled. Doubling the precision of an analog device may be expensive, whereas the cost of each additional bit of digital precision is incremental; that is, the cost is proportional to the logarithm of the precision expressed as a fraction of full range.

The forgoing considerations might seem to be a convincing argument for the superiority of digital to analog technology, and indeed, they were an important factor in the competition between analog and digital computers in the middle of the twentieth century (Small 2001, pp. 257–261). However, as was argued at that time, many computer applications do not require high precision. Indeed, in many engineering applications, the input data are known to only a few digits, and the equations may be approximate or derived from experiments. In these cases, the very high precision of digital computation is unnecessary and may in fact be misleading (e.g., if one displays all 14 digits of a result that is accurate to only three). Furthermore, many applications in image processing and control do not require high precision. More recently, research in artificial neural networks (ANNs) has shown that low-precision analog computation is sufficient for almost all ANN applications. Indeed, neural information processing in the brain seems to operate with very low precision (perhaps less than 10% McClelland 1986, p. 378), for which it compensates with massive parallelism. For example, by coarse coding, a population of low-precision devices can represent information with relatively high precision (Rumelhart et al. 1986, pp. 91–96; Sanger 1996).

## Scaling

An important aspect of analog computing is scaling, which is used to adjust a problem to an analog computer. First is time scaling, which adjusts a problem to the characteristic time scale at which a computer operates, which is a consequence of its design and the physical processes by which it is

realized (Peterson 1967, pp. 37–44; Rogers and Connolly 1960, pp. 262–263; Weyrick 1969, pp. 241–243). For example, we might want a simulation to proceed on a very different time scale from the primary system. Thus, a weather or economic simulation should proceed faster than real time in order to get useful predictions. Conversely, we might want to slow down a simulation of protein folding so that we can observe the stages in the process. Also, for accurate results, it is necessary to avoid exceeding the maximum response rate of the analog devices, which might dictate a slower simulation speed. On the other hand, too slow a computation might be inaccurate as a consequence of instability (e.g., drift and leakage in the integrators).

Time scaling affects only time-dependent operations such as integration. For example, suppose  $t$ , time in the primary system or “problem time,” is related to  $\tau$ , time in the computer, by  $\tau = \beta t$ . Therefore, an integration  $u(t) = \int_0^t v(t')dt'$  in the primary system is replaced by the integration  $u(\tau) = \beta^{-1} \int_0^\tau v(\tau')d\tau'$  on the computer. Thus, time scaling may be accomplished simply by decreasing the input gain to the integrator by a factor of  $\beta$ .

Fundamental to analog computation is the representation of a continuous quantity in the primary system by a continuous quantity in the computer. For example, a displacement  $x$  in meters might be represented by a potential  $V$  in volts. The two are related by an amplitude or magnitude scale factor,  $V = \alpha x$  (with unit volts/meter), chosen to meet two criteria (Ashley 1963; Pour-El 1974; Rogers and Connolly 1960; Weyrick 1969). On the one hand,  $\alpha$  must be sufficiently small so that the range of the problem variable is accommodated within the range of values supported by the computing device. Exceeding the device’s intended operating range may lead to inaccurate results (e.g., forcing a linear device into nonlinear behavior). On the other hand, the scale factor should not be too small, or relevant variation in the problem variable will be less than the resolution of the device, also leading to inaccuracy. (Recall that precision is specified as a fraction of full-range variation.)

In addition to the explicit variables of the primary system, there are implicit variables, such as

the time derivatives of the explicit variables, and scale factors must be chosen for them too. For example, in addition to displacement  $x$ , a problem might include velocity  $\dot{x}$  and acceleration  $\ddot{x}$ . Therefore, scale factors  $\alpha$ ,  $\alpha'$ , and  $\alpha''$  must be chosen so that  $\alpha x$ ,  $\alpha' \dot{x}$ , and  $\alpha'' \ddot{x}$  have an appropriate range of variation (neither too large nor too small). Once a scale factor has been chosen, the primary system equations are adjusted to obtain the analog computing equations. For example, if we have scaled  $u = \alpha x$  and  $v = \alpha' \dot{x}$ , then the integration  $x(t) = \int_0^t \dot{x}(t')dt'$  would be computed by scaled equation:

$$u(t) = \frac{\alpha}{\alpha'} \int_0^t v(t')dt'.$$

This is accomplished by simply setting the input gain of the integrator to  $\alpha/\alpha'$ .

In practice, time scaling and magnitude scaling are not independent (Rogers and Connolly 1960, p. 262). For example, if the derivatives of a variable can be large, then the variable can change rapidly, and so it may be necessary to slow down the computation to avoid exceeding the high-frequency response of the computer. Conversely, small derivatives might require the computation to be run faster to avoid integrator leakage, drift, and other sources of imprecision. Appropriate scale factors are determined by considering both the physics and the mathematics of the problem (Peterson 1967, pp. 40–44). That is, first, the physics of the primary system may limit the ranges of the variables and their derivatives. Second, analysis of the mathematical equations describing the system can give additional information on the ranges of the variables. For example, in some cases the natural frequency of a system can be estimated from the coefficients of the differential equations; the maximum of the  $n$ th derivative is then estimated as the  $n$ th power of this frequency (Peterson 1967, p. 42; Weyrick 1969, pp. 238–240). In any case, it is not necessary to have accurate values for the ranges; rough estimates giving orders of magnitude are adequate.

It is tempting to think of magnitude scaling as a problem unique to analog computing, but before the invention of floating-point numbers, it was

also necessary in digital computer programming. In any case it is an essential aspect of analog computing, in which physical processes are more directly used for computation than they are in digital computing. Although the necessity of scaling has been a source of criticism, advocates for analog computing have argued that it is a blessing in disguise, because it leads to improved understanding of the primary system, which was often the goal of the computation in the first place (Bissell 2004; Small 2001, Chap. 8). Practitioners of analog computing are more likely to have an intuitive understanding of both the primary system and its mathematical description (see section “[Analog Thinking](#)”).

## Analog Computation in Nature

Computational processes – that is to say, information processing and control – occur in many living systems, most obviously in nervous systems, but also in the self-organized behavior of groups of organisms. In most cases natural computation is analog, either because it makes use of continuous natural processes or because it makes use of discrete but stochastic processes. Several examples will be considered briefly.

### Neural Computation

In the past neurons were thought of binary computing devices, something like digital logic gates. This was a consequence of the “all or nothing” response of a neuron, which refers to the fact that it does or does not generate an action potential (voltage spike) depending, respectively, on whether its total input exceeds a threshold or not (more accurately, it generates an action potential if the membrane depolarization at the axon hillock exceeds the threshold and the neuron is not in its refractory period). Certainly, some neurons (e.g., so-called command neurons) do act something like logic gates. However, most neurons are analyzed better as analog devices because the rate of impulse generation represents significant information. In particular, an amplitude code, the membrane potential near the axon hillock (which is a summation of the electrical influences on the

neuron), is translated into a rate code for more reliable long-distance transmission along the axons. Nevertheless, the code is low precision (about one digit), since information theory shows that it takes at least  $N$  milliseconds (and probably more like  $5 N$  ms) to discriminate  $N$  values (MacLennan 1991). The rate code is translated back to an amplitude code by the synapses, since successive impulses release neurotransmitter from the axon terminal, which diffuses across the synaptic cleft to receptors. Thus, a synapse acts as a leaky integrator to time-average the impulses.

As previously discussed (section “[Continuous State Space](#)”), many artificial neural net models have real-valued neural activities, which correspond to rate-encoded axonal signals of biological neurons. On the other hand, these models typically treat the input connections as simple real-valued weights, which ignores the analog signal processing that takes place in the dendritic trees of biological neurons. The dendritic trees of many neurons are complex structures, which often have tens of thousands of synaptic inputs. The binding of neurotransmitters to receptors causes minute voltage fluctuations, which propagate along the membrane, and ultimately cause voltage fluctuations at the axon hillock, which influence the impulse rate. Since the dendrites have both resistance and capacitance, to a first approximation, the signal propagation is described by the “cable equations,” which describe passive signal propagation in cables of specified diameter, capacitance, and resistance (Anderson 1995, Chap. 1). Therefore, to a first approximation, a neuron’s dendritic net operates as an adaptive linear analog filter with thousands of inputs, and so it is capable of quite complex signal processing. More accurately, however, it must be treated as a nonlinear analog filter, since voltage-gated ion channels introduce nonlinear effects. The extent of analog signal processing in dendritic trees is still poorly understood.

In most cases, then, neural information processing is treated best as low-precision analog computation. Although individual neurons have quite broadly tuned responses, accuracy in perception and sensorimotor control is achieved through coarse coding, as already discussed (section

“Characteristics of Analog Computation”). Further, one widely used neural representation is the cortical map, in which neurons are systematically arranged in accord with one or more dimensions of their stimulus space, so that stimuli are represented by patterns of activity over the map. (Examples are tonotopic maps, in which pitch is mapped to cortical location, and retinotopic maps, in which cortical location represents retinal location.) Since neural density in the cortex is at least 125,000 neurons per square millimeter (Collins et al. 2010), even relatively small cortical maps can be treated as fields and information processing in them as analog field computation. Overall, the brain demonstrates what can be accomplished by massively parallel analog computation, even if the individual devices are comparatively slow and of low precision.

### Adaptive Self-Organization in Social Insects

Another example of analog computation in nature is provided by the self-organizing behavior of social insects, microorganisms, and other populations (Camazine et al. 2001). Often, such organisms respond to concentrations, or gradients in the concentrations, of chemicals produced by other members of the population. These chemicals may be deposited and diffuse through the environment. In other cases, insects and other organisms communicate by contact, but may maintain estimates of the relative proportions of different kinds of contacts. Because the quantities are effectively continuous, all these are examples of analog control and computation.

Self-organizing populations provide many informative examples of the use of natural processes for analog information processing and control. For example, diffusion of pheromones is a common means of self-organization in insect colonies, facilitating the creation of paths to resources, the construction of nests, and many other functions (Camazine et al. 2001). Real diffusion (as opposed to sequential simulations of it) executes, in effect, a massively parallel search of paths from the chemical’s source to its recipients and allows the identification of near-optimal paths. Furthermore, if the chemical degrades, as

is generally the case, then the system will be adaptive, in effect continually searching out the shortest paths, so long as the source continues to function (Camazine et al. 2001). Simulated diffusion has been applied to robot path planning (Khatib 1986; Rimon and Koditschek 1989).

### Genetic Circuits

Another example of natural analog computing is provided by the genetic regulatory networks that control the behavior of cells, in multicellular organisms as well as single-celled ones (Davidson 2006). These networks are defined by the mutually interdependent regulatory genes, promoters, and repressors that control the internal and external behavior of a cell. The interdependencies are mediated by proteins, the synthesis of which is governed by genes, and which in turn regulate the synthesis of other gene products (or themselves). Since it is the quantities of these substances that are relevant, many of the regulatory motifs can be described in computational terms as adders, subtracters, integrators, etc. Thus, the genetic regulatory network implements an analog control system for the cell (Reiner 1968).

It might be argued that the number of intracellular molecules of a particular protein is a (relatively small) discrete number, and therefore that it is inaccurate to treat it as a continuous quantity. However, the molecular processes in the cell are stochastic, and so the relevant quantity is the probability that a regulatory protein will bind to a regulatory site. Further, the processes take place in continuous real time, and so the rates are generally the significant quantities. Finally, although in some cases gene activity is either on or off (more accurately: very low), in other cases it varies continuously between these extremes (Hartl 1994, pp. 388–390). Indeed, electronic analog circuit design is proving to be a valuable model for synthetic biology circuits at the DNA, RNA, protein, and small molecule levels (Teo et al. 2015). Knowledge and tools for the design, simulation, analysis, and implementation of electronic analog circuits can be applied in synthetic biology to the design of molecular circuits that are efficient and robust in spite of stochastic effects.

Embryological development combines the analog control of individual cells with the sort of self-organization of populations seen in social insects and other colonial organisms. Locomotion of the cells and the expression of specific genes are controlled by chemical signals, among other mechanisms (Davidson 2006; Davies 2005). Thus, PDEs have proved useful in explaining some aspects of development; for example, reaction–diffusion equations have been used to describe the formation of hair-coat patterns and related phenomena (Camazine et al. 2001; Maini and Othmer 2001; Murray 1977; see ► “[Reaction-Diffusion Computing](#)”). Therefore, the developmental process is governed by naturally occurring analog computation.

## Is Everything a Computer?

It might seem that any continuous physical process could be viewed as analog computation, which would make the term almost meaningless. As the question has been put, is it meaningful (or useful) to say that the solar system is computing Kepler’s laws? In fact, it is possible and worthwhile to make a distinction between computation and other physical processes that happen to be described by mathematical laws (MacLennan 1994a, b, 2001, 2004).

If we recall the original meaning of analog computation (section “[Definition of the Subject](#)”), we see that the computational system is used to solve some mathematical problem with respect to a primary system. What makes this possible is that the computational system and the primary system have the same, or systematically related, abstract (mathematical) structures. Thus, the computational system can inform us about the primary system, or be used to control it, etc. Although from a practical standpoint some analogs are better than others, in principle any physical system can be used that obeys the same equations as the primary system.

Based on these considerations, we may define computation as a physical process, the purpose of which is the abstract manipulation of abstract objects (i.e., information processing); this

definition applies to analog, digital, and hybrid computation (MacLennan 1994a, b, 2001, 2004). Therefore, to determine if a natural system is computational, we need to look to its purpose or function within the context of the living system of which it is a part. One test of whether its function is the abstract manipulation of abstract objects is to ask whether it could still fulfill its function if realized by different physical processes, a property called multiple realizability. (Similarly, in artificial systems, a simulation of the economy might be realized equally accurately by a hydraulic analog computer or an electronic analog computer (Bissell 2004)). By this standard, the majority of the nervous system is purely computational; in principle it could be replaced by electronic devices obeying the same differential equations. In the other cases we have considered (self-organization of living populations, genetic circuits) there are instances of both pure computation and computation mixed with other functions (for example, where the specific substances used have other – e.g., metabolic – roles in the living system).

## General-Purpose Analog Computation

### The Importance of General-Purpose Computers

Although special-purpose analog and digital computers have been developed, and continue to be developed, for many purposes, the importance of general-purpose computers, which can be configured easily for a wide variety of purposes, has been recognized since at least the nineteenth century. Babbage’s plans for a general-purpose digital computer, his analytical engine (1835), are well known, but a general-purpose differential analyzer was advocated by Kelvin (1876). Practical general-purpose analog and digital computers were first developed at about the same time: from the early 1930s through the war years. General-purpose computers of both kinds permit the prototyping of special-purpose computers and, more importantly, permit the flexible reuse of computer hardware for different or evolving purposes.

The concept of a general-purpose computer is useful also for determining the limits of a computing paradigm. If one can design – theoretically or practically – a universal computer, that is, a general-purpose computer capable of simulating any computer in a relevant class, then anything uncomputable by the universal computer will also be uncomputable by any computer in that class. This is, of course, the approach used to show that certain functions are uncomputable by any Turing machine because they are uncomputable by a universal Turing machine. For the same reason, the concept of general-purpose analog computers, and in particular of universal analog computers, is theoretically important for establishing limits to analog computation.

### General-Purpose Electronic Analog Computers

Before taking up these theoretical issues, it is worth recalling that a typical electronic GPAC would include linear elements, such as adders, subtracters, constant multipliers, integrators, and differentiators; nonlinear elements, such as variable multipliers and function generators; and other computational elements, such as comparators, noise generators, and delay elements (section “[Electronic Analog Computation in the Twentieth Century](#)”). These are, of course, in addition to input/output devices, which would not affect its computational abilities.

### Shannon’s Analysis

Claude Shannon did an important analysis of the computational capabilities of the differential analyzer, which applies to many GPACs (Shannon 1941, 1993). He considered an abstract differential analyzer equipped with an unlimited number of integrators, adders, constant multipliers, and function generators (for functions with only a finite number of finite discontinuities), with at most one source of drive (which limits possible interconnections between units). This was based on prior work that had shown that almost all the generally used elementary functions could be generated with addition and integration. We will summarize informally a few of Shannon’s results; for details, please consult the original paper.

First, Shannon offers proofs that, by setting up the correct ODEs, a GPAC with the mentioned facilities can generate a function if and only if it is not hypertranscendental (Theorem II); thus, the GPAC can generate any function that is algebraic transcendental (a very large class) but not, for example, Euler’s gamma function and Riemann’s zeta function. He also shows that the GPAC can generate functions derived from generable functions, such as the integrals, derivatives, inverses, and compositions of generable functions (Theorems III, IV). These results can be generalized to functions of any number of variables and to their compositions, partial derivatives, and inverses with respect to any one variable (Theorems VI, VII, IX, X).

Next, Shannon shows that a function of any number of variables that is continuous over a closed region of space can be approximated arbitrarily closely over that region with a finite number of adders and integrators (Theorems V, VIII).

Shannon then turns from the generation of functions to the solution of ODEs and shows that the GPAC can solve any system of ODEs defined in terms of non-hypertranscendental functions (Theorem XI).

Finally, Shannon addresses a question that might seem of limited interest, but turns out to be relevant to the computational power of analog computers (see section “[Analog Computation and the Turing Limit](#)” below). To understand it, we must recall that he was investigating the differential analyzer – a mechanical analog computer – but similar issues arise in other analog computing technologies. The question is whether it is possible to perform an arbitrary constant multiplication,  $u = kv$ , by means of gear ratios. He showed that if we have just two gear ratios  $a$  and  $b$  ( $a, b \neq 0, 1$ ) such that  $b$  is not a rational power of  $a$ , then by combinations of these gears, we can approximate  $k$  arbitrarily closely (Theorem XII). That is, to approximate multiplication by arbitrary real numbers, it is sufficient to be able to multiply by  $a, b$ , and their inverses, provided  $a$  and  $b$  are not related by a rational power.

Shannon mentions an alternative method of constant multiplication, which uses integration,  $kv = \int_0^v kdv$ , but this requires setting the integrand

to the constant function  $k$ . Therefore, multiplying by an arbitrary real number requires the ability to input an arbitrary real as the integrand. The issue of real-valued inputs and outputs to analog computers is relevant both to their theoretical power and to practical matters of their application (see section “[Real-Valued Inputs, Outputs, and Constants](#)”).

Shannon’s proofs, which were incomplete, were eventually refined by Pour-El (1974) and finally corrected by Lipshitz and Rubel (1987). Rubel (1988) proved that Shannon’s GPAC cannot solve the Dirichlet problem for Laplace’s equation on the disk; indeed, it is limited to initial-value problems for algebraic ODEs. Specifically, the Shannon–Pour-El Thesis is that the outputs of the GPAC are exactly the solutions of the algebraic differential equations, that is, equations of the form

$$P[x, y(x), y'(x), y''(x), \dots, y^{(n)}(x)] = 0,$$

where  $P$  is a polynomial that is not identically vanishing in any of its variables (these are the differentially algebraic functions) (Rubel 1985). (For details please consult the cited papers.) The limitations of Shannon’s GPAC motivated Rubel’s definition of the extended analog computer.

### Rubel’s Extended Analog Computer

The combination of Rubel’s (1985) conviction that the brain is an analog computer together with the limitations of Shannon’s GPAC led him to propose the extended analog computer (EAC) (Rubel 1993).

Like Shannon’s GPAC (and the Turing machine), the EAC is a conceptual computer intended to facilitate theoretical investigation of the limits of a class of computers. The EAC extends the GPAC in a number of respects. For example, whereas the GPAC solves equations defined over a single variable (time), the EAC can generate functions over any finite number of real variables. Further, whereas the GPAC is restricted to initial-value problems for ODEs, the EAC solves both initial- and boundary-value problems for a variety of PDEs.

The EAC is structured into a series of levels, each more powerful than the ones below it, from which it accepts inputs. The inputs to the lowest level are a finite number of real variables (“settings”). At this level it operates on real polynomials, from which it is able to generate the differentially algebraic functions. The computation on each level is accomplished by conceptual analog devices, which include constant real-number generators, adders, multipliers, differentiators, “substitutors” (for function composition), devices for analytic continuation, and inverters, which solve systems of equations defined over functions generated by the lower levels. Most characteristic of the EAC is the “boundary-value-problem box,” which solves systems of PDEs and ODEs subject to boundary conditions and other constraints. The PDEs are defined in terms of functions generated by the lower levels. Such PDE solvers may seem implausible, and so it is important to recall field-computing devices for this purpose were implemented in some practical analog computers (see section “[History](#)”) and more recently in Mills’ EAC (Mills et al. 2006). As Rubel observed, PDE solvers could be implemented by physical processes that obey the same PDEs (heat equation, wave equation, etc.). (See also section “[Future Directions](#)” below.)

Finally, the EAC is required to be “extremely well posed,” which means that each level is relatively insensitive to perturbations in its inputs; thus, “all the outputs depend in a strongly deterministic and stable way on the initial settings of the machine” (Rubel 1993).

Rubel (1993) proves that the EAC can compute everything that the GPAC can compute, but also such functions as the gamma and zeta functions, and that it can solve the Dirichlet problem for Laplace’s equation on the disk, all of which are beyond the GPAC’s capabilities. Further, whereas the GPAC can compute differentially algebraic functions of time, the EAC can compute differentially algebraic functions of any finite number of real variables. In fact, Rubel did not find any real-analytic ( $C^\infty$ ) function that is not computable on the EAC, but he observes that if the EAC can indeed generate every real-analytic function, it would be too broad to be useful as a model of analog computation.

## Analog Computation and the Turing Limit

### Introduction

The Church–Turing Thesis asserts that anything that is effectively computable is computable by a Turing machine, but Turing machines (and equivalent models, such as the lambda calculus) are models of discrete computation, and so it is natural to wonder how analog computing compares in power, and in particular whether it can compute beyond the “Turing limit.” Superficial answers are easy to obtain, but the issue is subtle because it depends upon choices among definitions, none of which is obviously correct, because it involves the foundations of mathematics and its philosophy, and because it raises epistemological issues about the role of models in scientific theories. This is an active research area, but many of the results are apparently inconsistent due to the differing assumptions on which they are based. Therefore, this section will be limited to a mention of a few of the interesting results, but without attempting a comprehensive, systematic, or detailed survey; Siegelmam (1999) can serve as an introduction to the literature.

## A Sampling of Theoretical Results

### Continuous-Time Models

Orponen’s (1997) survey of continuous-time computation theory is a good introduction to the literature as of that time; here, we give a sample of these and more recent results.

There are several results showing that – under various assumptions – analog computers have at least the power of Turing machines (TMs). For example, Branicky (1994) showed that a TM could be simulated by ODEs, but he used non-differentiable functions; Bournez et al. (2006) provide an alternative construction using only analytic functions. They also prove that GPAC computability coincides with (Turing-) computable analysis, which is surprising, since the gamma function is Turing computable but, as we have seen, the GPAC cannot generate it. The paradox is resolved by a distinction between

generating a function and computing it, with the latter, broader notion permitting convergent computation of the function (that is, as  $t \rightarrow \infty$ ). However, the computational power of general ODEs has not been determined in general (Siegelmann 1999, p. 149). Pour-El and Richards exhibit a Turing-computable ODE that does not have a Turing-computable solution (Pour-El and Richards 1979, 1982). Stannett (1990) also defined a continuous-time analog computer that can solve the halting problem.

Moore (1996) defines a class of continuous-time recursive functions over the reals, which includes a zero-finding operator  $\mu$ . Functions can be classified into a hierarchy depending on the number of uses of  $\mu$ , with the lowest level (no  $\mu$ s) corresponding approximately to Shannon’s GPAC. Higher levels can compute non-Turing-computable functions, such as the decision procedure for the halting problem, but he questions whether this result is relevant in the physical world, which is constrained by “noise, quantum effects, finite accuracy, and limited resources.” Bournez and Cosnard (1996) have extended these results and shown that many dynamical systems have super-Turing power.

Omohundro (1984) showed that a system of ten coupled nonlinear PDEs could simulate an arbitrary cellular automaton, which implies that PDEs have at least Turing power. Further, Wolpert and MacLennan (Wolpert 1991; Wolpert and MacLennan 1993) showed that any TM can be simulated by a field computer with linear dynamics, but the construction uses Dirac delta functions. Pour-El and Richards exhibit a wave equation in three-dimensional space with Turing-computable initial conditions, but for which the unique solution is Turing uncomputable (Pour-EL and Richards 1981, 1982).

### Sequential-Time Models

We will mention a few of the results that have been obtained concerning the power of sequential-time analog computation.

Although the BSS model has been investigated extensively, its power has not been completely determined (Blum et al. 1988, 1998). It is known

to depend on whether just rational numbers or arbitrary real numbers are allowed in its programs (Siegelmann 1999, p. 148).

A coupled map lattice (CML) is a cellular automaton with real-valued states; it is a sequential-time analog computer, which can be considered a discrete-space approximation to a simple sequential-time field computer. Orponen and Matamala (1996) showed that a finite CML can simulate a universal Turing machine. However, since a CML can simulate a BSS program or a recurrent neural network (see section “[Recurrent Neural Networks](#)” below), it actually has super-Turing power (Siegelmann 1999, p. 149).

Recurrent neural networks are some of the most important examples of sequential analog computers, and so the following section is devoted to them.

### [Recurrent Neural Networks](#)

With the renewed interest in neural networks in the mid-1980s, many investigators wondered if recurrent neural nets have super-Turing power. Garzon and Franklin showed that a sequential-time net with a countable infinity of neurons could exceed Turing power (Franklin and Garzon 1990; Garzon and Franklin 1989, 1990). Indeed, Siegelmann and Sontag (1994) showed that finite neural nets with real-valued weights have super-Turing power, but Maass and Sontag (1999) showed that recurrent nets with Gaussian or similar noise had sub-Turing power, illustrating again the dependence on these results on assumptions about what is a reasonable mathematical model of analog computing.

For recent results on recurrent neural networks, we will restrict our attention of the work of Siegelmann (1999), who addresses the computational power of these networks in terms of the classes of languages they can recognize. Without loss of generality, the languages are restricted to sets of binary strings. A string to be tested is fed to the network one bit at a time, along with an input that indicates when the end of the input string has been reached. The network is said to decide whether the string is in the language if it correctly

indicates whether it is in the set or not after some finite number of sequential steps since input began.

Siegelmann shows that, if exponential time is allowed for recognition, finite recurrent neural networks with real-valued weights (and saturated-linear activation functions) can recognize all languages, and thus, they are more powerful than Turing machines. Similarly, stochastic networks with rational weights also have super-Turing power, although less power than the deterministic nets with real weights. (Specifically, they compute P/POLY and BPP/log\*, respectively; see Siegelmann (1999, Chaps. 4 and 9) for details.) She further argues that these neural networks serve as a “standard model” of (sequential) analog computation (comparable to Turing machines in Church–Turing computation) and therefore that the limits and capabilities of these nets apply to sequential analog computation generally.

Siegelmann (1999, p. 156) observes that the super-Turing power of recurrent neural networks is a consequence of their use of irrational real-valued weights. In effect, a real number can contain an infinite number of bits of information. This raises the question of how the irrational weights of a network can ever be set, since it is not possible to control a physical quantity with infinite precision. However, although irrational weights may not be able to be set from outside the network, they can be computed within the network by learning algorithms, which are analog computations. Thus, Siegelmann suggests, the fundamental distinction may be between static computational models, such as the Turing machine and its equivalents, and dynamically evolving computational models, which can tune continuously variable parameters and thereby achieve super-Turing power.

### **Dissipative Models**

Beyond the issue of the power of analog computing relative to the Turing limit, there are also questions of its relative efficiency. For example, could analog computing solve NP-hard problems in polynomial or even linear time? In traditional computational complexity theory, efficiency issues are addressed in terms of the asymptotic number of computation steps to compute a

function as the size of the function's input increases. One way to address corresponding issues in an analog context is by treating an analog computation as a dissipative system, which in this context means a system that decreases some quantity (analogous to energy) so that the system state converges to a point attractor. From this perspective, the initial state of the system incorporates the input to the computation, and the attractor represents its output. Therefore, Sieglemann, Fishman, and Ben-Hur have developed a complexity theory for dissipative systems, in both sequential and continuous time, which addresses the rate of convergence in terms of the underlying rates of the system (Ben-Hur et al. 2002; Siegelmann et al. 1999). The relation between dissipative complexity classes (e.g.,  $P_d$ ,  $NP_d$ ) and corresponding classical complexity classes ( $P$ ,  $NP$ ) remains unclear (Siegelmann 1999, p. 151).

### **Real-Valued Inputs, Outputs, and Constants**

A common argument, with relevance to the theoretical power of analog computation, is that an input to an analog computer must be determined by setting a dial to a number or by typing a number into digital-to-analog conversion device, and therefore that the input will be a rational number. The same argument applies to any internal constants in the analog computation. Similarly, it is argued, any output from an analog computer must be measured, and the accuracy of measurement is limited, so that the result will be a rational number. Therefore, it is claimed that real numbers are irrelevant to analog computing, since any practical analog computer computes a function from the rationals to the rationals and can therefore be simulated by a Turing machine. (See related arguments by Martin Davis 2004, 2006).

There are a number of interrelated issues here, which may be considered briefly. First, the argument is couched in terms of the input or output of digital representations, and the numbers so represented are necessarily rational (more generally, computable). This seems natural enough

when we think of an analog computer as a calculating device, and in fact many historical analog computers were used in this way and had digital inputs and outputs (since this is our most reliable way of recording and reproducing quantities).

However, in many analog control systems, the inputs and outputs are continuous physical quantities that vary continuously in time (also a continuous physical quantity); that is, according to current physical theory, these quantities are real numbers, which vary according to differential equations. It is worth recalling that physical quantities are neither rational nor irrational; they can be so classified only in comparison with each other or with respect to a unit, that is, only if they are measured and digitally represented. Furthermore, physical quantities are neither computable nor uncomputable (in a Church–Turing sense); these terms apply only to discrete representations of these quantities (i.e., to numerals or other digital representations).

Therefore, in accord with ordinary mathematical descriptions of physical processes, analog computations can be treated as having arbitrary real numbers (in some range) as inputs, outputs, or internal states; like other continuous processes, continuous-time analog computations pass through all the reals in some range, including non-Turing-computable reals. Paradoxically, however, these same physical processes can be simulated on digital computers.

### **The Issue of Simulation by Turing Machines and Digital Computers**

Theoretical results about the computational power, relative to Turing machines, of neural networks and other analog models of computation raise difficult issues, some of which are epistemological rather than strictly technical. On the one hand, we have a series of theoretical results proving the super-Turing power of analog computation models of various kinds. On the other hand, we have the obvious fact that neural nets are routinely simulated on ordinary digital computers, which have at most the power of Turing machines. Furthermore, it is reasonable to suppose that any physical process that might be used

to realize analog computation – and certainly the known processes – could be simulated on a digital computer, as is done routinely in computational science. This would seem to be incontrovertible proof that analog computation is no more powerful than Turing machines. The crux of the paradox lies, of course, in the non-Turing-computable reals. These numbers are a familiar, accepted, and necessary part of standard mathematics, in which physical theory is formulated, but from the standpoint of Church–Turing (CT) computation, they do not exist. This suggests that the paradox is not a contradiction but reflects a divergence between the goals and assumptions of the two models of computation.

## The Problem of Models of Computation

These issues may be put in context by recalling that the Church–Turing (CT) model of computation is in fact a model, and therefore that it has the limitations of all models. A model is a cognitive tool that improves our ability to understand some class of phenomena by preserving relevant characteristics of the phenomena while altering other, irrelevant (or less relevant) characteristics. For example, a scale model alters the size (taken to be irrelevant) while preserving shape and other characteristics. Often, a model achieves its purposes by making simplifying or idealizing assumptions, which facilitate analysis or simulation of the system. For example, we may use a linear mathematical model of a physical process that is only approximately linear. For a model to be effective, it must preserve characteristics and make simplifying assumptions that are appropriate to the domain of questions it is intended to answer, its *frame of relevance* (MacLennan 2004). If a model is applied to problems outside of its frame of relevance, then it may give answers that are misleading or incorrect, because they depend more on the simplifying assumptions than on the phenomena being modeled. Therefore, we must be especially cautious applying a model outside of its frame of relevance, or even at the limits of its frame, where the simplifying assumptions become progressively less appropriate. The problem is aggravated by the

fact that often the frame of relevance is not explicitly defined but resides in a tacit background of practices and skills within some discipline.

Therefore, to determine the applicability of the CT model of computation to analog computing, we must consider the frame of relevance of the CT model. This is easiest if we recall the domain of issues and questions it was originally developed to address: issues of effective calculability and derivability in formalized mathematics. This frame of relevance determines many of the assumptions of the CT model, for example, that information is represented by finite discrete structures of symbols from a finite alphabet, that information processing proceeds by the application of definite formal rules at discrete instants of time, and that a computational or derivational process must be completed in a finite number of these steps. (See MacLennan 2003, 2004 for a more detailed discussion of the frame of relevance of the CT model.) Many of these assumptions are incompatible with analog computing and with the frames of relevance of many models of analog computation.

## Relevant Issues for Analog Computation

Analog computation is often used for control. Historically, analog computers were used in control systems and to simulate control systems, but contemporary analog VLSI is also frequently applied in control. Natural analog computation also frequently serves a control function, for example, sensorimotor control by the nervous system, genetic regulation in cells, and self-organized cooperation in insect colonies. Therefore, control systems provide one frame of relevance for models of analog computation.

In this frame of relevance, real-time response is a critical issue, which models of analog computation, therefore, ought to be able to address. Thus, it is necessary to be able to relate the speed and frequency response of analog computation to the rates of the physical processes by which the computation is realized. Traditional methods of algorithm analysis, which are based on sequential time and asymptotic behavior, are inadequate in this

frame of relevance. On the one hand, the constants (time scale factors), which reflect the underlying rate of computation, are absolutely critical (but ignored in asymptotic analysis); on the other hand, in control applications the asymptotic behavior of algorithm is generally irrelevant, since the inputs are typically fixed in size or of a limited range of sizes.

The CT model of computation is oriented around the idea that the purpose of a computation is to evaluate a mathematical function. Therefore, the basic criterion of adequacy for a computation is correctness, that is, that given a precise representation of an input to the function, it will produce (after finitely many steps) a precise representation of the corresponding output of the function. In the context of natural computation and control, however, other criteria may be equally or even more relevant. For example, robustness is important: how well does the system respond in the presence of noise, uncertainty, imprecision, and error, which are unavoidable in real natural and artificial control systems, and how well does it respond to defects and damage, which arise in many natural and artificial contexts. Since the real world is unpredictable, flexibility is also important: how well does an artificial system respond to inputs for which it was not designed, and how well does a natural system behave in situations outside the range of those to which it is evolutionarily adapted. Therefore, adaptability (through learning and other means) is another important issue in this frame of relevance. (See MacLennan (2003, 2004) for a more detailed discussion of the frames of relevance of natural computation and control.)

## Transcending Turing Computability

Thus, we see that many applications of analog computation raise different questions from those addressed by the CT model of computation; the most useful models of analog computing will have a different frame of relevance. In order to address traditional questions such as whether analog computers can compute “beyond the Turing limit” or whether they can solve NP-hard problems in polynomial time, it is necessary to construct models of

analog computation within the CT frame of relevance. Unfortunately, constructing such models requires making commitments about many issues (such as the representation of reals and the discretization of time) that may affect the answers to these questions but are fundamentally unimportant in the frame of relevance of the most useful applications of the concept of analog computation. Therefore, being overly focused on traditional problems in the theory of computation (which was formulated for a different frame of relevance) may distract us from formulating models of analog computation that can address important issues in its own frame of relevance.

## Analog Thinking

It will be worthwhile to say a few words about the cognitive implications of analog computing, which are a largely forgotten aspect of analog versus digital debates of the late twentieth century. For example, it was argued that analog computing provides a deeper intuitive understanding of a system than the alternatives do (Bissell 2004; Small 2001, Chap. 8). On the one hand, analog computers afforded a means of understanding analytically intractable systems by means of “dynamic models.” By setting up an analog simulation, it was possible to vary the parameters and explore interactively the behavior of a dynamical system that could not be analyzed mathematically. Digital simulations, in contrast, were orders of magnitude slower and did not permit this kind of interactive investigation. (Performance has improved sufficiently in contemporary digital computers so that in many cases, digital simulations can be used as dynamic models, sometimes with an interface that mimics an analog computer; see Bissell 2004.)

Analog computing is also relevant to the cognitive distinction between knowing how (procedural knowledge) and knowing that (declarative knowledge) (Small 2001, Chap. 8). The latter (“know-that”) is more characteristic of scientific culture, which strives for generality and exactness, often by designing experiments that allow phenomena to be studied in isolation, whereas the former (“know-how”) is more characteristic of engineering culture;

at least it was so through the first half of the twentieth century, before the development of “engineering science” and the widespread use of analytic techniques in engineering education and practice. Engineers were faced with analytically intractable systems, with inexact measurements, and with empirical relationships (characteristic curves, etc.), all of which made analog computers attractive for solving engineering problems. Furthermore, because analog computing made use of physical phenomena that were mathematically analogous to those in the primary system, the engineer’s intuition and understanding of one system could be transferred to the other. Some commentators have mourned the loss of hands-on intuitive understanding attendant on the increasingly scientific orientation of engineering education and the disappearance of analog computers (Bissell 2004; Lang 2000; Owens 1986; Puchta 1996).

I will mention one last cognitive issue relevant to the differences between analog and digital computing. As already discussed (section “[Characteristics of Analog Computation](#)”), it is generally agreed that it is less expensive to achieve high precision with digital technology than with analog technology. Of course, high precision may not be important, for example, when the available data are inexact or in natural computation. Further, some advocates of analog computing argue that high-precision digital results are often misleading (Small 2001). Precision does not imply accuracy, and the fact that an answer is displayed with ten digits does not guarantee that it is accurate to ten digits; in particular, engineering data may be known to only a few significant figures, and the accuracy of digital calculation may be limited by numerical problems. Therefore, on the one hand, users of digital computers might fall into the trap of trusting their apparently exact results, but users of modest-precision analog computers were more inclined to healthy skepticism about their computations. Or so it was claimed.

## Future Directions

Certainly, there are many purposes that are best served by digital technology; indeed, there is a tendency nowadays to think that everything is done better digitally. Therefore, it will be

worthwhile to consider whether analog computation should have a role in future computing technologies. I will argue that the approaching end of Moore’s Law (1965), which has predicted exponential growth in digital logic densities, will encourage the development of new analog computing technologies.

Two avenues present themselves as ways toward greater computing power: faster individual computing elements and greater densities of computing elements. Greater density increases computing power by facilitating parallel computing, and by enabling greater computing power to be put into smaller packages. Other things being equal, the fewer the layers of implementation between the computational operations and the physical processes that realize them, that is to say, the more directly the physical processes implement the computations, the more quickly they will be able to proceed. Since most physical processes are continuous (defined by differential equations), analog computation is generally faster than digital. For example, we may compare analog addition, implemented directly by the additive combination of physical quantities, with the sequential logic of digital addition. Similarly, other things being equal, the fewer physical devices required to implement a computational element, the greater will be the density of these elements. Therefore, in general, the closer the computational process is to the physical processes that realize it, the fewer devices will be required, and so the continuity of physical law suggests that analog computation has the potential for greater density than digital. For example, four transistors can realize analog addition, whereas many more are required for digital addition. Both considerations argue for an increasing role of analog computation in post-Moore’s Law computing.

From this broad perspective, there are many physical phenomena that are potentially usable for future analog computing technologies. We seek phenomena that can be described by well-known and useful mathematical functions (e.g., addition, multiplication, exponential, logarithm, convolution). These descriptions do not need to be exact for the phenomena to be useful in many applications, for which limited range and precision are adequate. Furthermore, in some applications,

speed is not an important criterion; for example, in some control applications, small size, low power, robustness, etc., may be more important than speed, so long as the computer responds quickly enough to accomplish the control task. Of course there are many other considerations in determining whether given physical phenomena can be used for practical analog computation in a given application (MacLennan 2009). These include stability, controllability, manufacturability, and the ease of interfacing with input and output transducers and other devices. Nevertheless, in the post-Moore's Law world, we will have to be willing to consider all physical phenomena as potential computing technologies, and in many cases, we will find that analog computing is the most effective way to utilize them.

Natural computation provides many examples of effective analog computation realized by relatively slow, low-precision operations, often through massive parallelism. Therefore, post-Moore's Law computing has much to learn from the natural world.

## Bibliography

### Primary Literature

- Anderson JA (1995) An introduction to neural networks. MIT Press, Cambridge
- Ashley JR (1963) Introduction to analog computing. Wiley, New York
- Aspray W (1993) Edwin L. Harder and the Anacom: analog computing at Westinghouse. IEEE Ann Hist Comput 15(2):35–52
- Basford DA, Smith JM, Connor RJ, MacLennan BJ, Holleman J (2016) The impact of analog computational error on an analog Boolean satisfiability solver. IEEE international symposium on circuits & systems 2016, Montreal
- Basu A, Brink S, Schlottmann C, Ramakrishnan S, Petre C, Koziol S, Baskaya F, Twigg CM, Hasler P (2010) A floating-gate-based field-programmable analog array. IEEE J Solid State Circuits 45:1781–1794
- Ben-Hur A, Siegelmann HT, Fishman S (2002) A theory of complexity for continuous time systems. J Complex 18:51–86
- Bissell CC (2004) A great disappearing act: the electronic analogue computer. In: IEEE conference on the history of electronics, Bletchley, June 2004. pp 28–30
- Blum L, Shub M, Smale S (1988) On a theory of computation and complexity over the real numbers: NP completeness, recursive functions and universal machines. Bull Am Math Soc 21:1–46

- Blum L, Cucker F, Shub M, Smale S (1998) Complexity and real computation. Springer, Berlin
- Bournez O, Cosnard M (1996) On the computational power of dynamical systems and hybrid systems. Theor Comput Sci 168(2):417–459
- Bournez O, Campagnolo ML, Graça DS, Hainry E (2006) The general purpose analog computer and computable analysis are two equivalent paradigms of analog computation. In: Theory and applications of models of computation (TAMC 2006). Lectures notes in computer science, vol 3959. Springer, Berlin, pp 631–643
- Bowles MD (1996) US technological enthusiasm and British technological skepticism in the age of the analog brain. Ann Hist Comput 18(4):5–15
- Branicky MS (1994) Analog computation with continuous ODEs. In: Proceedings IEEE workshop on physics and computation, Dallas, pp 265–274
- Brockett RW (1988) Dynamical systems that sort lists, diagonalize matrices and solve linear programming problems. In: Proceedings 27th IEEE conference decision and control, Austin, Dec 1988, pp 799–803
- Camazine S, Deneubourg J-L, Franks NR, Sneyd G, Theraulaz J, Bonabeau E (2001) Self-organization in biological systems. Princeton University Press, New York
- Clymer AB (1993) The mechanical analog computers of Hannibal Ford and William Newell. IEEE Ann Hist Comput 15(2):19–34
- Collins CE, Airey DC, Young NA, Leitch DB, Kaas JH (2010) Neuron densities vary across and within cortical areas in primates. Proc Natl Acad Sci 107(36):15927–15932
- Davidson EH (2006) The regulatory genome: gene regulatory networks in development and evolution. Academic, Amsterdam
- Davies JA (2005) Mechanisms of morphogenesis. Elsevier, Amsterdam
- Davis M (2004) The myth of hypercomputation. In: Teuscher C (ed) Alan turing: life and legacy of a great thinker. Springer, Berlin, pp 195–212
- Davis M (2006) Why there is no such discipline as hyper-computation. Appl Math Comput 178:4–7
- Ercsey-Ravasz M, Toroczkai Z (2011) Optimization hardness as transient chaos in an analog approach to constraint satisfaction. Nat Phys 7(12):966–970
- Fakhraie SM, Smith KC (1997) VLSI-compatible implementation for artificial neural networks. Kluwer, Boston
- Franklin S, Garzon M (1990) Neural computability. In: Omnidvar OM (ed) Progress in neural networks, vol 1. Ablex, Norwood, pp 127–145
- Freeth T, Bitsakis Y, Moussas X, Seiradakis JH, Tsaklikas A, Mangou H, Zafeiropoulou M, Hadland R, Bate D, Ramsey A, Allen M, Crawley A, Hockley P, Malzbender T, Gelb D, Ambrisco W, Edmunds MG (2006) Decoding the ancient Greek astronomical calculator known as the Antikythera mechanism. Nature 444:587–591
- Garzon M, Franklin S (1989) Neural computability II (extended abstract). In: Proceedings, IJCNN international joint conference on neural networks,

- vol 1. Institute of Electrical and Electronic Engineers, New York, pp 631–637
- Garzon M, Franklin S (1990) Computation on graphs. In: Omidvar OM (ed) *Progress in neural networks*, vol 2. Ablex, Norwood
- van Gelder T (1997) Dynamics and cognition. In: Haugeland J (ed) *Mind design II: philosophy, psychology and artificial intelligence*. MIT Press, Cambridge, MA, pp 421–450. Revised & enlarged edition, Chap 16
- Goldstine HH (1972) *The computer from Pascal to von Neumann*. Princeton University Press, Princeton
- Grossberg S (1967) Nonlinear difference-differential equations in prediction and learning theory. *Proc Natl Acad Sci U S A* 58(4):1329–1334
- Grossberg S (1973) Contour enhancement, short term memory, and constancies in reverberating neural networks. *Stud Appl Math* LII:213–257
- Grossberg S (1976) Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biol Cybern* 23:121–134
- Hartl DL (1994) *Genetics*, 3rd edn. Jones & Bartlett, Boston
- Hopfield JJ (1984) Neurons with graded response have collective computational properties like those of two-state neurons. *Proc Natl Acad Sci U S A* 81:3088–3092
- Howe RM (1961) Design fundamentals of analog computer components. Van Nostrand, Princeton
- Khatib O (1986) Real-time obstacle avoidance for manipulators and mobile robots. *Int J Robot Res* 5:90–99
- Kirchhoff G (1845) Ueber den Durchgang eines elektrischen Stromes durch eine Ebene, insbesondere durch eine kreisförmige. *Ann Phys Chem* 140(4):497–514
- Lang GF (2000) Analog was *not* a computer trademark! Why would anyone write about analog computers in year 2000? *Sound Vib* 34(8):16–24
- Lipshitz L, Rubel LA (1987) A differentially algebraic replacement theorem. *Proc Am Math Soc* 99(2):367–372
- Maass W, Sontag ED (1999) Analog neural nets with Gaussian or other common noise distributions cannot recognize arbitrary regular languages. *Neural Comput* 11(3):771–782
- MacLennan BJ (1987) Technology-independent design of neurocomputers: the universal field computer. In: Caudill M, Butler C (eds) *Proceedings of the IEEE first international conference on neural networks*, vol 3. IEEE Press, New York, pp 39–49
- MacLennan BJ (1990) Field computation: a theoretical framework for massively parallel analog computation, parts I–IV. Technical report UT-CS-90-100, Department of Computer Science, University of Tennessee, Knoxville. Available from <http://web.eecs.utk.edu/~mclennan>. Accessed 20 May 2008
- MacLennan BJ (1991) Gabor representations of spatiotemporal visual images. Technical report UT-CS-91-144, Department of Computer Science, University of Tennessee, Knoxville. Available from <http://web.eecs.utk.edu/~mclennan>. Accessed 20 May 2008
- MacLennan BJ (1994b) Continuous computation and the emergence of the discrete. In: Pribram KH (ed) *Origins: brain & self-organization*. Lawrence Erlbaum, Hillsdale, pp 121–151
- MacLennan BJ (1994a) Words lie in our way. *Minds Mach* 4(4):421–437
- MacLennan BJ (1995) Continuous formal systems: a unifying model in language and cognition. In: *Proceedings of the IEEE workshop on architectures for semiotic modeling and situation analysis in large complex systems*, Monterey, Aug 1995. IEEE Press, New York, pp 161–172. Also available from <http://web.eecs.utk.edu/~mclennan>. Accessed 20 May 2008
- MacLennan BJ (1999) Field computation in natural and artificial intelligence. *Inf Sci* 119:73–89
- MacLennan BJ (2001) Can differential equations compute? Technical report UT-CS-01-459, Department of Computer Science, University of Tennessee, Knoxville. Available from <http://web.eecs.utk.edu/~mclennan>. Accessed 20 May 2008
- MacLennan BJ (2003) Transcending Turing computability. *Minds Mach* 13:3–22
- MacLennan BJ (2004) Natural computation and non-Turing models of computation. *Theor Comput Sci* 317:115–145
- MacLennan BJ (2009) Super-Turing or non-Turing? Extending the concept of computation. *Int J Unconv Comput* 5:369–387
- Maini PK, Othmer HG (eds) (2001) *Mathematical models for biological pattern formation*. Springer, New York
- Maziarz EA, Greenwood T (1968) Greek mathematical philosophy. Frederick Ungar, New York
- McClelland JL, Rumelhart DE, the PDP Research Group (1986) *Parallel distributed processing: explorations in the microstructure of cognition*, vol 2, Psychological and biological models. MIT Press, Cambridge
- Mead C (1987) Silicon models of neural computation. In: Caudill M, Butler C (eds) *Proceedings, IEEE first international conference on neural networks*, vol 1. IEEE Press, Piscataway, pp 91–106
- Mead C (1989) *Analog VLSI and neural systems*. Addison-Wesley, Reading
- Mills JW (1996) The continuous retina: image processing with a single-sensor artificial neural field network. In: *Proceedings IEEE conference on neural networks*. IEEE Press, Piscataway
- Mills JW, Himebaugh B, Kopecky B, Parker M, Shue C, Weilemann C (2006) “Empty space” computes: the evolution of an unconventional supercomputer. In: *Proceedings of the 3rd conference on computing frontiers*, New York, May 2006. ACM Press, pp 115–126
- Molnár B, Ercsey-Ravasz M (2013) Asymmetric continuous-time neural networks without local traps for solving constraint satisfaction problems. *PLoS One* 8(9):e73400
- Moore GE (1965) Cramming more components onto integrated circuits. *Electronics* 38(8):114–117
- Moore C (1996) Recursion theory on the reals and continuous-time computation. *Theor Comput Sci* 162:23–44
- Murray JD (1977) *Lectures on nonlinear differential-equation models in biology*. Clarendon Press, Oxford
- Omohundro S (1984) Modeling cellular automata with partial differential equations. *Physica D* 10:128–134

- Orponen P (1997) A survey of continuous-time computation theory. In: Advances in algorithms, languages, and complexity. Kluwer, Dordrecht, pp 209–224
- Orponen P, Matamala M (1996) Universal computation by finite two-dimensional coupled map lattices. In: Proceedings, physics and computation 1996. New England Complex Systems Institute, Cambridge, pp 243–7
- Owens L (1986) Vannevar Bush and the differential analyzer: the text and context of an early computer. *Technol Cult* 27(1):63–95
- Peterson GR (1967) Basic analog computation. Macmillan, New York
- Pour-El MB (1974) Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Trans Am Math Soc* 199:1–29
- Pour-El MB, Richards I (1979) A computable ordinary differential equation which possesses no computable solution. *Ann Math Log* 17:61–90
- Pour-El MB, Richards I (1981) The wave equation with computable initial data such that its unique solution is not computable. *Adv Math* 39:215–239
- Pour-El MB, Richards I (1982) Noncomputability in models of physical phenomena. *Int J Theor Phys* 21:553–555
- Puchta S (1996) On the role of mathematics and mathematical knowledge in the invention of Vannevar Bush's early analog computers. *IEEE Ann Hist Comput* 18(4):49–59
- Reiner JM (1968) The organism as an adaptive control system. Prentice-Hall, Englewood Cliffs
- Rimon E, Koditschek DE (1989) The construction of analytic diffeomorphisms for exact robot navigation on star worlds. In: Proceedings of the 1989 I-E. international conference on robotics and automation, Scottsdale. IEEE Press, New York, pp 21–26
- Rogers AE, Connolly TW (1960) Analog computation in engineering design. McGraw-Hill, New York
- Rubel LA (1985) The brain as an analog computer. *J Theor Neurobiol* 4:73–81
- Rubel LA (1988) Some mathematical limitations of the general-purpose analog computer. *Adv Appl Math* 9:22–34
- Rubel LA (1993) The extended analog computer. *Adv Appl Math* 14:39–50
- Rumelhart DE, McClelland JL, The PDP Research Group (1986) Parallel distributed processing: explorations in the microstructure of cognition, Foundations, vol 1. MIT Press, Cambridge
- Sanger TD (1996) Probability density estimation for the interpretation of neural population codes. *J Neurophysiol* 76:2790–2793
- Shannon CE (1941) Mathematical theory of the differential analyzer. *J Math Phys Mass Inst Technol* 20:337–354
- Shannon CE (1993) Mathematical theory of the differential analyzer. In: Sloane NJA, Wyner AD (eds) Claude Elwood Shannon: collected papers. IEEE Press, New York, pp 496–513
- Siegelmann HT (1999) Neural networks and analog computation: beyond the Turing limit. Birkhäuser, Boston
- Siegelmann HT, Sontag ED (1994) Analog computation via neural networks. *Theor Comput Sci* 131:331–360
- Siegelmann HT, Ben-Hur A, Fishman S (1999) Computational complexity for continuous time dynamics. *Phys Rev Lett* 83(7):1463–1466
- Small JS (1993) General-purpose electronic analog computing. *IEEE Ann Hist Comput* 15(2):8–18
- Small JS (2001) The analogue alternative: the electronic analogue computer in Britain and the USA, 1930–1975. Routledge, London/New York
- Stannett M (1990) X-machines and the halting problem: building a super-Turing machine. *Form Asp Comput* 2:331–341
- Teo JJY, Woo SS, Sarpeshkar R (2015) Synthetic biology: a unifying view and review using analog circuits. *IEEE Trans Biomed Circ Syst* 9(4):453–474
- Thomson W (Lord Kelvin) (1876) Mechanical integration of the general linear differential equation of any order with variable coefficients. *Proc R Soc* 24:271–275
- Thomson W (Lord Kelvin) (1878) Harmonic analyzer. *Proc R Soc* 27:371–373
- Thomson W (Lord Kelvin) (1938) The tides. In: The Harvard classics, vol 30, Scientific papers. Collier, New York, pp 274–307
- Truitt TD, Rogers AE (1960) Basics of analog computers. John F Rider, New York
- Weyrick RC (1969) Fundamentals of analog computers. Prentice-Hall, Englewood Cliffs
- Wolpert DH (1991) A computationally universal field computer which is purely linear. Technical report LA-UR-91-2937. Los Alamos National Laboratory, Loa Alamos
- Wolpert DH, MacLennan BJ (1993) A computationally universal field computer that is purely linear. Technical report UT-CS-93-206. Department of Computer Science, University of Tennessee, Knoxville
- Yin X, Sedighi B, Varga M, Ercsey-Ravasz M, Toroczkai Z, Hu XS (2016) Efficient analog circuits for Boolean satisfiability. arXiv:1606.07467

## Books and Reviews

- Bissell CC (2004) A great disappearing act: the electronic analogue computer. In: IEEE conference on the history of electronics, Bletchley, June 2004
- Fifer S (1961) Analog computation: theory, techniques and applications, vol 4. McGraw-Hill, New York
- Lipka J (1918) Graphical and mechanical computation. Wiley, New York
- Mead C (1989) Analog VLSI and neural systems. Addison-Wesley, Reading
- Siegelmann HT (1999b) Neural networks and analog computation: beyond the Turing limit. Birkhäuser, Boston
- Small JS (1993) General-purpose electronic analog computing: 1945–1965. *IEEE Ann Hist Comput* 15(2):8–18
- Small JS (2001) The analogue alternative: the electronic analogue computer in Britain and the USA, 1930–1975. Routledge, London/New York



---

## Mechanical Computing: The Computational Complexity of Physical Devices

John H. Reif

Department of Computer Science, Duke University, Durham, NC, USA

### Article Outline

#### Glossary

Definition of the Subject and Its Importance  
Introduction to Computational Complexity  
The Computational Complexity of Motion  
Planning and Simulation of Mechanical Devices  
Other PSPACE Completeness Results for Mechanical Devices  
Ballistic Collision-Based Computing Machines and PSPACE  
Brownian Machines and PSPACE  
Hardness Results for Mechanical Devices with a Constant Number of Degrees of Freedom  
NP Hardness Results for Path Problems in Two and three Dimensions  
Concrete Mechanical Computing Devices  
Mechanical Devices for Storage and Sums of Numbers  
Analog Mechanical Computing Devices  
Digital Mechanical Devices for Arithmetic Operations  
Digital Mechanical Devices for Mathematical Tables and Functions  
Mechanical Devices for Timing, Sequencing, and Logical Control  
Mechanical Devices Used in Cryptography  
Mechanical and Electrooptical Devices for Integer Factorization  
Future Directions  
Bibliography

## Glossary

**Computable** Capable of being worked out by calculation, especially using a computer.

**Mechanism** A machine or part of a machine that performs a particular task computation; the use of a computer for calculation.

**Simulation** Used to denote both the modeling of a physical system by a computer and the modeling of the operation of a computer by a mechanical system; the difference will be clear from the context.

### Definition of the Subject and Its Importance

Mechanical devices for computation appear to be largely displaced by the widespread use of microprocessor-based computers that are pervading almost all aspects of our lives. Nevertheless, mechanical devices for computation are of interest for at least three reasons:

- (a) *Historical*: The use of mechanical devices for computation is of central importance in the historical study of technologies, with a history dating to thousands of years and with surprising applications even in relatively recent times.
- (b) *Technical and practical*: The use of mechanical devices for computation persists and has not yet been completely displaced by widespread use of microprocessor-based computers. Mechanical computers have found applications in various emerging technologies at the microscale that combine mechanical functions with computational and control functions not feasible by purely electronic processing. Mechanical computers also have been demonstrated at the molecular scale and may also provide unique capabilities at that scale. The physical designs for these modern micro- and molecular-scale

mechanical computers may be based on the prior designs of the large-scale mechanical computers constructed in the past.

- (c) *Impact of physical assumptions on complexity of motion planning, design, and simulation:* The study of computation done by mechanical devices is also of central importance in providing lower bounds on the computational resources such as time and/or space required to simulate a mechanical system observing given physical laws. In particular, the problem of simulating the mechanical system can be shown to be *computationally hard* if a hard computational problem can be simulated by the mechanical system. A similar approach can be used to provide lower bounds on the computational resources required to solve various motion-planning tasks that arise in the field of robotics. Typically, a robotic motion-planning task is specified by a geometric description of the robot (or collection of robots) to be moved, its initial and final positions, the obstacles it is to avoid, as well as a model for the type of feasible motion and physical laws for the movement. The problem of planning such as robotic motion-planning task can be shown to be computationally hard if a hard computational problem can be simulated by the robotic motion-planning task.

## Introduction to Computational Complexity

### Abstract Computing Machine Models

To gage the computational power of a family of mechanical computers, we will use a widely known abstract computational model known as the Turing machine, defined in this section.

**The Turing machine.** The *Turing machine* model formulated by Alan Turing (1937) was the first complete mathematical model of an abstract computing machine that possessed universal computing power. The machine model has (i) a finite-state transition control for logical control of the machine processing, (ii) a tape with a sequence of storage cells containing symbolic values, and (iii) a

tape scanner for reading and writing values to and from the tape cells, which could be made to move (left and right) along the tape cells.

A machine model is *abstract* if the description of the machine transition mechanism or memory mechanism does not provide specification of the mechanical apparatus used to implement them in practice. Since Turing's description did not include any specification of the mechanical mechanism for executing the finite-state transitions, it cannot be viewed as a concrete mechanical computing machine but instead as an abstract machine. Still it is a valuable computational model, due to its simplicity and very widespread use in computational theory.

A *universal* Turing machine simulates any other Turing machine; it takes its input a pair consisting of a string providing a symbolic description of a Turing machine M and the input string x and simulates M on input x. Because of its simplicity and elegance, the Turing machine has come to be the standard computing model used for most theoretical works in computer science. Informally, the Church-Turing hypothesis states that a Turing machine model can simulate a computation by any "reasonable" computational model (we will discuss some other reasonable computational models below).

**Computational problems.** A *computational problem* is given an input string specified by a string over a finite alphabet; determine the Boolean answer: 1 if the answer is *yes* and otherwise 0. For simplicity, we generally will restrict the input alphabet to be the binary alphabet {0,1}. The *input size* of a computational problem is the number of input symbols, which is the number of bits of the binary specification of the input. (Note: It is more common to make these definitions in terms of language acceptance. A *language* is a set of strings over a given finite alphabet of symbols. A computational problem can be identified with the language consisting of all strings over the input alphabet where the answer is 1. For simplicity, we defined each complexity class as the corresponding class of problems.)

**Recursively computable problems and undecidable problems.** There is a large class

of problems, known as *recursively computable* problems, that Turing machines compute in finite computations, that is, always halting in finite time with the answer. There are certain problems that are not recursively computable; these are called *undecidable* problems. The *halting problem* is given a Turing machine description and an input, output 1 if the Turing machine ever halts and else output 0. Turing proved the halting problem is undecidable. His proof used a method known as a diagonalization method; it considered an enumeration of all Turing machines and inputs and showed that a contradiction occurs when a universal Turing machine attempts to solve the halting problem for each Turing machine and each possible input.

**Computational complexity classes.** *Computational complexity* (see Lewis and Papadimitriou 1997) is the amount of computational resources required to solve a given computational problem. A *complexity class* is a family of problems, generally defined in terms of limitations on the resources of the computational model. The complexity classes of interest here will be associated with restrictions on the time (number of steps until the machine halts) and/or space (the number of tape cells used in the computation) of Turing machines. There are a number of notable complexity classes:

**P** is the complexity class associated with efficient computations and is formally defined to be the set of problems solved by Turing machine computations running in time polynomial in the input size (typically, this is the number of bits of the binary specification of the input).

**NP** is the complexity class associated with combinatorial optimization problems which if solved can be easily determined to have correct solutions and is formally defined to be the set of problems solved by Turing machine computations using nondeterministic choice running in polynomial time.

**PSPACE** is the complexity class that is defined to be the set of problems solved by Turing machines running in space polynomial in the input size.

**EXPTIME** is the complexity class that is defined to be the set of problems solved by Turing machine computations running in time exponential in the input size.

**NP** and **PSPACE** are widely considered to have instances that are not solvable in P, and it has been proved that **EXPTIME** has problems that are not in P.

**Polynomial time reductions.** A *polynomial time reduction* from a problem Q' to a problem Q is a polynomial time Turing machine computation that transforms any instance of the problem Q' into an instance of the problem Q which has an answer *yes* if and only if the problem Q' has an answer *yes*. Informally, this implies that problem Q can be used to efficiently solve the problem Q'. A problem Q is *hard* for a family F of problems if, for every problem Q' in F, there is a polynomial time reduction from Q' to Q. Informally, this implies that problem Q can be used to efficiently solve *any* problem in F. A problem Q is *complete* for a family F of problems if Q is in C and also hard for F.

**Hardness proofs for mechanical problems.** We will later consider various mechanical problems and characterize their computation power:

*Undecidable mechanical problems:* typically this was proven by a computable reduction from the halting problem for universal Turing machine problems to an instance of the mechanical problem; this is equivalent to showing that the mechanical problem can be viewed as a computational machine that can simulate a universal Turing machine computation.

*Mechanical problems that are hard for NP, PSPACE, or EXPTIME:* typically this was proven by a polynomial time reduction from the problems in the appropriate complexity class to an instance of the mechanical problem; again, this is equivalent to showing that the mechanical problem can be viewed as a computational machine that can simulate a Turing machine computation in the appropriate complexity class.

The simulation proofs in either case often provide insight into the intrinsic computational power of the mechanical problem or mechanical machine.

### Other Abstract Computing Machine Models

There are a number of abstract computing models discussed in this chapter that are equivalent or nearly equivalent to conventional deterministic Turing machines:

**Reversible Turing machines.** A computing device is (*logically*) reversible if each transition of its computation can be executed both in forward direction and in reverse direction, without loss of information. Landauer (1961) showed that irreversible computations must generate heat in the computing process and that reversible computations have the property that, if executed slowly enough, can (in the limit) consume no energy in an adiabatic computation. A *reversible Turing machine* model allows the scan head to observe three consecutive tape symbols and to execute transitions both in forward and in reverse direction. Bennett (1973) showed that any computing machine (e.g., an abstract machine such as a Turing machine) can be transformed to do only reversible computations, which implied that reversible computing devices are capable of universal computation. Bennett's reversibility construction required extra space to store information to insure reversibility, but this extra space can be reduced by increasing the time. Vitanyi (Li and Vitanyi 1996) gives trade-offs between time and space in the resulting reversible machine. Lewis and Papadimitriou (Turing 1937) showed that reversible Turing machines are equivalent in computational power to conventional Turing machines when the computations are bounded by polynomial time, and Crescenzi and Papadimitriou (1995) proved a similar result when the computations are bounded by polynomial space. This implies that the definitions of the complexity classes **P** and **PSPACE** do not depend on the Turing machines being reversible or not. Reversible Turing machines are used in many of the

computational complexity proofs to be mentioned involving simulations by mechanical computing machines.

**Cellular automata.** These are sets of finite-state machines that are typically connected together by a grid network. There are known efficient simulations of Turing machine by cellular automata (e.g., see Wolfram (1984) for some known universal simulations). A number of the particle-based mechanical machines to be described are known to simulate cellular automata.

**Randomized Turing machines.** The machine can make random choices in its computation. While the use of randomized choice can be very useful in many efficient algorithms, there is evidence that randomization only provides limited additional computational power above conventional deterministic Turing machines. (In particular, there are a variety of pseudorandom number generation methods proposed for producing long pseudorandom sequences from short truly random seeds, which are widely conjectured to be indistinguishable from truly random sequences by polynomial time Turing machines.) A number of the mechanical machines to be described using Brownian motion have natural sources of random numbers.

There are also a number of abstract computing machine models that appear to be more powerful than conventional deterministic Turing machines:

**Real-valued Turing machines.** In these machines, due to Blum et al. (1996), each storage cell or register can store any real value (which may be transcendental). Operations are extended to allow infinite precision arithmetic operations on real numbers. To our knowledge, none of the analog computers that we will describe in this chapter have this power.

**Quantum computers.** A *quantum superposition* is a linear superposition of basis states; it is defined by a vector of complex amplitudes whose absolute magnitudes sum to 1. In a quantum computer, the quantum superposition of basis states is transformed in each step by a unitary transformation (this is a linear mapping

that is reversible and always preserves the value of the sum of the absolute magnitudes of its inputs). The outputs of a quantum computation are read by observations that project the quantum superposition to classical values; a given state is chosen with probability defined by the magnitude of the amplitude of that state in the quantum superposition. Feynman (1982) and Benioff (1982) were the first to suggest the use of quantum mechanical principles for doing computation, and Deutsch (1985) was the first to formulate an abstract model for quantum computing and showed it was universal. Since then, there is a large body of work in quantum computing (see Gruska 1999; Nielsen and Chuang 2000) and quantum information theory (see Jaeger 2006; Reif 2009a). Some of the particle-based methods for mechanical computing described below make use of quantum phenomena but generally are not considered to have the full power of quantum computers.

## The Computational Complexity of Motion Planning and Simulation of Mechanical Devices

### Complexity of Motion Planning for Mechanical Devices with Articulated Joints

The first known computational complexity result involving mechanical motion or robotic motion planning was in 1979 by Reif (1979). He considers a class of mechanical systems consisting of a finite set of connected polygons with articulated joints, which are required to be moved between two configurations in three-dimensional space avoiding a finite set of fixed polygonal obstacles. To specify the movement problem (as well as the other movement problems described below unless otherwise stated), the object to be moved, as well as its initial and final positions, and the obstacles are all defined by linear inequalities with rational coefficients with a finite number of bits. He showed that this class of motion-planning problems is hard for PSPACE. Since it is widely conjectured that PSPACE contains problems that are not solvable in polynomial time, this result

provided the first evidence that these robotic motion-planning problems are not solvable in time polynomial in  $n$  if the number of degrees of freedom grows with  $n$ . His proof involved simulating a reversible Turing machine with  $n$  tape cells by a mechanical device with  $n$ -articulated polygonal arms that had to be maneuvered through a set of fixed polygonal obstacles similar to the channels in Swiss cheese. These obstacles were devised to force the mechanical device to simulate transitions of the reversible Turing machine to be simulated, where the positions of the arms encoded the tape cell contents, and tape read/write operations were simulated by channels of the obstacles which forced the arms to be reconfigured appropriately. This class of movement problems can be solved by reduction to the problem of finding a path in an  $O(n)$ -dimensional space avoiding a fixed set of polynomial obstacle surfaces, which can be solved by a PSPACE algorithm due to Canny (1988). Hence, this class of movement problems is PSPACE complete. (In the case the object to be moved consists of only one rigid polygon, the problem is known as the piano mover's problem and has a polynomial time solution by Schwartz and Sharir (1983).)

### Other PSPACE Completeness Results for Mechanical Devices

There were many subsequent PSPACE completeness results for mechanical devices (two of which we mention below), which generally involved *multiple degrees of freedom*:

**The warehouseman's problem.** Schwartz and Sharir (Hopcroft et al. 1984) showed in 1984 that moving a set of  $n$  disconnected polygons in two dimensions from an initial position to a final position among finite set of fixed polygonal obstacles PSPACE hard.

There are two classes of mechanical dynamic systems, the ballistic machines and the Browning machines described below that can be shown to provide simulations of polynomial space Turing machine computations.

## Ballistic Collision-Based Computing Machines and PSPACE

A *ballistic computer* (see Bennett 1982, 2003) is a conservative dynamical system that follows a mechanical trajectory isomorphic to the desired computation. It has the following properties:

Trajectories of distinct ballistic computers cannot be merged.

All computational operations must be reversible. Computations, when executed at constant velocity, require no consumption of energy.

Computations must be executed without error and need to be isolated from external noise and heat sources.

*Collision-based computing* (Adamatzky 2001) is computation by a set of particles, where each particle holds a finite state value, and state transformations are executed at the time of collisions between particles. Since collisions between distinct pairs of particles can be simultaneous, the model allows for parallel computation. In some cases the particles can be configured to execute cellular automata computations (Squier and Steiglitz 1994). Most proposed methods for *collision-based computing* are ballistic computers as defined above. Examples of concrete physical systems for collision-based computing are:

**The billiard ball computers.** Fredkin and Toffoli (1982) considered a mechanical computing model, the *billiard ball computer*, consisting spherical billiard balls with polygonal obstacles, where the billiard balls were assumed to have perfect elastic collisions with no friction. They showed in 1982 that a billiard ball computer, with an unbounded number of billiard balls, could simulate a reversible computing machine model that used reversible Boolean logical gates known as Toffoli gates. When restricted to finite set of n spherical billiard balls, their construction provides a simulation of a polynomial space reversible Turing machine.

**Particle-like waves in excitable medium.** Certain classes of excitable medium have discrete models that can exhibit particle-like waves that propagate through the media (Adamatzky 1996), and using this phenomenon, Adamatzky (1998a) gave a simulation of a universal Turing machine that, if restricted to n particle waves, provided a simulation of a polynomial space Turing machine.

**Soliton computers.** A soliton is a wave packet that maintains a self-reinforcing shape as it travels at constant speed through a nonlinear dispersive media. A soliton computer (Jakubowski et al. 1998, 2001) makes use of optical solitons to hold state, and state transformations are made by colliding solitons.

## Brownian Machines and PSPACE

In a mechanical system exhibiting *fully Brownian motion*, the parts move freely and independently, up to the constraints that either link the parts together or force the parts to exert on each other. In a fully Brownian motion, the movement is entirely due to heat, and there is no other source of energy driving the movement of the system. An example of mechanical systems with fully Brownian motion is a set of particles exhibiting Browning motion with electrostatic interaction. The rate of movement of mechanical system with fully Brownian motion is determined entirely by the drift rate in the random walk of their configurations.

Other mechanical systems, known as *driven Brownian motion systems*, exhibit movement only partly due to heat; in addition, there is a source of energy driving the movement of the system. Examples of driven Brownian motion systems are:

Feynman's ratchet and pawl (Feynman 1963), which is a mechanical ratchet system that has a driving force but that can operate reversibly Polymerase enzyme, which uses ATP as fuel to drive their average movement forward but also can operate reversibly

There is no energy consumed by fully Brownian motion devices, whereas driven Brownian motion devices require power that grows as a quadratic function of the drive rate in which operations are executed (see Bennett 2003).

Bennett (1982) provides two examples of Brownian computing machines:

An *enzymatic machine*. This is a hypothetical biochemical device that simulates a Turing machine, using polymers to store symbolic values in a manner similar to Turing machine tapes, and uses hypothetical enzymatic reactions to execute state transitions and read/write operations into the polymer memory. Shapiro (1999) also describes a mechanical Turing machine whose transitions are executed by hypothetical enzymatic reactions.

A *clockwork computer*. This is a mechanism with linked articulated joints, with a Swiss-cheese-like set of obstacles, which force the device to simulate a Turing machine. In the case where the mechanism of Bennett's clockwork computer is restricted to have a linear number of parts, it can be used to provide a simulation of PSPACE similar to that of Reif (1979).

## **Hardness Results for Mechanical Devices with a Constant Number of Degrees of Freedom**

There were also additional computation complexity hardness results for mechanical devices, which only involved a *constant number of degrees of freedom*. These results exploited special properties of the mechanical systems to do the simulation:

**Motion planning with moving obstacles.** Reif and Sharir (1985) considered the problem of planning the motion of a rigid object (the robot) between two locations, avoiding a set of obstacles, some of which are rotating. They showed this problem is PSPACE hard. This result was perhaps surprising, since the number of degrees of freedom of movement of the

object to be moved was constant. However, the simulation used the rotational movement of obstacles to force the robot to be moved only to a position that encoded all the tape cells of M. The simulation of a Turing machine M was made by forcing the object between such locations (which encoded the entire n tape cell contents of M) at particular times and further forced that object to move between these locations over time in a way that simulated state transitions of M.

## **NP Hardness Results for Path Problems in Two and three Dimensions**

Shortest path problems in fixed dimensions involve only a constant number of degrees of freedom. Nevertheless, there are a number of NP hardness results for such problems. These results also led to proofs that certain physical simulations (in particular, simulation of multi-body molecular and celestial simulations) are NP hard and therefore not likely efficiently computable with high precision:

### **Finding shortest paths in three dimensions.**

Consider the problem of finding the shortest path of a point in three dimensions (where distance is measured in the Euclidean metric) avoiding fixed polyhedral obstacles whose coordinates are described by rational numbers with a finite number of bits. This shortest path problem can be solved in PSPACE (Canny 1988), but the precise complexity of the problem is an open problem. Canny and Reif (1987) were the first to provide a hardness complexity result for this problem; they showed the problem is NP hard. Their proof used novel techniques called *free path encoding* that used  $2^n$  homotopy equivalence classes of shortest paths. Using these techniques, they constructed exponentially many shortest path classes (with distinct homotopy) in single-source multiple-destination problems involving  $O(n)$  polygonal obstacles. They used each of these paths to encode a possible configuration

of the nondeterministic Turing machine with  $n$  binary storage cells. They also provided a technique for simulating each step of the Turing machine by the use of polygonal obstacles whose edges forced a permutation of these paths that encoded the modified configuration of the Turing machine. This encoding allowed them to prove that the single-source single-destination problem in three dimensions is NP hard. Similar free path encoding techniques were used for a number of other complexity hardness results for mechanical simulations described below.

**Kinodynamic planning.** *Kinodynamic planning* is the task of motion planning while subject to simultaneous kinematic and dynamics constraints. The algorithms for various classes of kinodynamic planning problems were first developed in (Canny et al. 1988). Canny and Reif (1987) also used free path encoding techniques to show that two-dimensional kinodynamic motion planning with bounded velocity is NP hard.

**Shortest curvature-constrained path planning in two dimensions.** We now consider *curvature-constrained shortest path problems*, which involve finding the shortest path by a point among polygonal obstacles, where there is an upper bound on the path curvature. A class of curvature-constrained shortest path problems in two dimensions was shown to be NP hard by Reif and Wang (1998), by devising a set of obstacles that forced the shortest curvature-constrained path to simulate a given nondeterministic Turing machine.

### PSPACE Hard Physical Simulation Problems

**Ray tracing with a rational placement and geometry.** Ray tracing is given an optical system, and the position and direction of an initial light ray determine if the light ray reaches some given final position. This problem of determining the path of light ray through an optical system was first formulated by Newton in his book on optics. Ray tracing has been used for designing and analyzing optical systems. It is also used extensively in computer graphics to render scenes with complex curved objects under global illumination. Reif

et al. (1990) first showed in 1990 the problem of ray tracing in various three-dimensional optical systems, where the optical devices either consist of reflective objects defined by quadratic equations or refractive objects defined by linear equations, but in either case the coefficients are restricted to be rational. They showed these ray tracing problems are PSPACE hard. Their proof used free path encoding techniques for simulating a nondeterministic linear space Turing machine, where the position of the ray as it enters a reflective or refractive optical object (such as a mirror or prism face) encodes the entire memory of the Turing machine to be simulated, and further steps of the Turing machine are simulated by optically inducing appropriate modifications in the position of the ray as it enters other reflective or refractive optical objects. This result implies that the apparently simple task of highly precise ray tracing through complex optical systems is not likely to be efficiently executed by a polynomial time computer. These results for ray tracing are another example of the use of a physical system to do powerful computations. A number of subsequent papers showed the NP hardness (recall NP is a subset of PSPACE, so NP hardness is a weaker type of hardness result PSPACE hardness) of various optical ray problems, such as the problem of determining if a light ray can reach a given position within a given time duration (Haist and Osten 2007; Oltean and Muntean 2008, 2009; Oltean 2008; Muntean and Oltean 2009), optical masks (Dolev and Fitoussi 2010), and ray tracing with multiple optical frequencies (Goliae and Jalili 2009, 2012) (see Woods and Naughton 2009 for a survey of these and related results in optical computing). A further PSPACE hardness result for an optics problem is given in a recent paper (Goliae and Foroughmand-Araabi 2013) concerning ray tracing with multiple optical frequencies, with an additional concentration operation.

**Molecular and gravitational mechanical systems.** The work of Tate and Reif (1993) on the complexity of  $n$ -body simulation provides an interesting example of the use of natural physical systems to do computation. They showed that the problem of  $n$ -body simulation is PSPACE hard and therefore not likely efficiently computable with

high precision. In particular, they considered multi-body systems in three dimensions with  $n$  particles and inverse polynomial force laws between each pair of particles (e.g., molecular systems with Columbic force laws or celestial simulations with gravitational force laws). It is quite surprising that such systems can be configured to do computation. Their hardness proof made use of free path encoding techniques similar to their proof (Reif et al. 1990) of the PSPACE-hardness of ray tracing. A single particle, which we will call the *memory-encoding particle*, is distinguished. The position of a memory-encoding particle as it crosses a plane encodes the entire memory of the given Turing machine to be simulated, and further steps of the Turing machine are simulated by inducing modifications in the trajectory of the memory-encoding particle. The modifications in the trajectory of the memory-encoding particle are made by use of other particles that have trajectories that induce force fields that essentially act like force mirrors, causing reflection-like changes in the trajectory of the memory-encoding particle. Hence, highly precise  $n$ -body molecular simulation is not likely to be efficiently executed by a polynomial time computer.

### A Provably Intractable Mechanical Simulation Problem: Compliant Motion Planning with Uncertainty in Control

Next, we consider compliant motion planning with uncertainty in control. Specifically, we consider a point in three dimensions which is commanded to move in a straight line but whose actual motion may differ from the commanded motion, possibly involving sliding against obstacles. Given that the point initially lies in some start region, the problem is to find a sequence of commanded velocities that is guaranteed to move the point to the goal. This problem was shown by Canny and Reif (1987) to be non-deterministic EXPTIME hard, making it the first provably intractable problem in robotics. Their proof used free path encoding techniques that exploited the uncertainty of position to encode exponential number of memory bits in a Turing machine simulation.

### Undecidable Mechanical Simulation Problems

**Motion planning with friction.** Consider a class of mechanical systems whose parts consist of a finite number of rigid objects defined by linear or quadratic surface patches connected by frictional contact linkages between the surfaces. (Note: This class of mechanisms is similar to the analytical engine developed by Babbage as described in the next sections, except that there are smooth frictional surfaces rather than toothed gears.) Reif and Sun (1998) proved that an arbitrary Turing machine could be simulated by a (universal) frictional mechanical system in this class consisting of a finite number of parts. The entire memory of a universal Turing machine was encoded in the rotational position of a rod. In each step, the mechanism used a construct similar to Babbage's machine to execute a state transition. The key idea in their construction is to utilize frictional clamping to allow for setting arbitrary high gear transmission. This allowed the mechanism to execute state transitions for arbitrary number of steps. Simulation of a universal Turing machine implied that the movement problem is undecidable when there are frictional linkages. (A problem is *undecidable* if there is no Turing machine that solves the problem for all inputs in finite time.) It also implied that a mechanical computer could be constructed with only a constant number of parts that has the power of an unconstrained Turing machine.

**Ray tracing with nonrational positioning.** Consider again the problem of ray tracing in a three-dimensional optical systems, where the optical devices again may be either consist of reflective objects defined by quadratic equations or refractive objects defined by linear equations. Reif et al. (1990) also proved that in the case where the coefficients of the defining equations are not restricted to be rational and include at least one irrational coefficient, then the resulting ray tracing problem could simulate a universal Turing machine, and so is undecidable. This ray tracing problem for reflective objects is equivalent to the problem of tracing the trajectory of a single particle bouncing between quadratic surfaces, which is also undecidable by this same result of Reif et al. (1990). An independent result of Moore (1990)

also showed the undecidability of the problem of tracing the trajectory of a single particle bouncing between quadratic surfaces.

**Dynamics and nonlinear mappings.** Moore (Moore and Shifts 1991), Ditto (Sinha and Ditto 1999), and Munakata et al. (2002) have also given universal Turing machine simulations of various dynamical systems with nonlinear mappings.

## Concrete Mechanical Computing Devices

Mechanical computers have a very extensive history; some surveys are given by Knott (1915), Hartree (1950), Engineering Research Associates (1950), Chase (1980), Martin (1992), and Davis (2000). Norman (2002) gave an overview of the literature of mechanical calculators and other historical computers, summarizing the contributions of notable manuscripts and publications on this topic.

## Mechanical Devices for Storage and Sums of Numbers

Mechanical methods, such as notches on stones and bones and knots and piles of pebbles, have been used since the Neolithic period for storing and summing integer values. One example of such a device, the *abacus*, which may have been developed and invented in Babylonia approximately 5000 years ago, makes use of beads sliding on cylindrical rods to facilitate addition and subtraction calculations.

## Analog Mechanical Computing Devices

Computing devices will be considered here to be *analog* (as opposed to digital) if they do not provide a method for restoring calculated values to discrete values, whereas digital devices provide restoration of calculated values to discrete values. (Note that both analog and digital computers use some kind of physical quantity to represent values that are stored and computed, so the use of physical encoding of computational values is not

necessarily the distinguishing characteristic of analog computing.) Descriptions of early analog computers are given by Horsburgh (1914), Turck (1921), Svoboda (1948), Hartree (1950), Engineering Research Associates (1950), and Soroka (1954). There are a wide variety of mechanical devices used for analog computing:

### Mechanical devices for astronomical and celestial calculation.

While we have no sufficient space in this article to fully discuss this rich history, we note that various mechanisms for predicting lunar and solar eclipses using optical illumination of configurations of stones and monoliths (e.g., Stonehenge) appear to date to the Neolithic period. The Hellenistic civilization in the classical period of ancient history seems to develop a number of analog calculating devices for astronomy calculations. A *planisphere*, which appears to have been used in the *Tetrabiblos* by Ptolemy in the second century, is a simple analog calculator for determining for any given date and time the visible portion of a star chart and consists of two disks rotating on the same pivot. *Astrolabes* are a family of analog calculators used for solving problems in spherical astronomy, often consisting of a combination of a planisphere and a dioptra (a sighting tube). An early form of an astrolabe is attributed to Hipparchus in the mid-second century. Other more complex mechanical mechanisms for predicting lunar and solar eclipses seem to have been developed in Hellenistic period. The most impressive and sophisticated known example of an ancient gear-based mechanical device from the Hellenistic period is the *Antikythera mechanism*, and recent research (Freeth et al. 2006) provides evidence it may have used to predict celestial events such as lunar and solar eclipses by the analog calculation of arithmetic-progression cycles. Like many other intellectual heritages, some elements of the design of such sophisticated gear-based mechanical devices may have been preserved by the Arabs at the end of that Hellenistic period and then transmitted to the Europeans in the middle ages.

**Planimeters.** There is a considerable history of mechanical devices that integrate curves. A *planimeter* is a mechanical device that integrates the area of the region enclosed by a two-dimensional closed curve, where the curve is presented as a function of the angle from some fixed interior point within the region. One of the first known planimeters was developed by J.A. Hermann in 1814 and improved (as the polar planimeter) by J.A. Hermann in 1856. This led to a wide variety of mechanical integrators known as wheel-and-disk integrators, whose input is the angular rotation of a rotating disk and whose output, provided by a tracking wheel, is the integral of a given function of that angle of rotation. More general mechanical integrators known as ball-and-disk integrators, whose input provided 2 degrees of freedom (the phase and amplitude of a complex function), were developed by James Thomson in 1886. There are also devices, such as the intergraph of Abdank Abakanowicz (1878) and C.V. Boys (1882), which integrate a one-variable real function of  $x$  presented as a curve  $y = f(x)$  on the Cartesian plane. Mechanical integrators were later widely used in WWI and WWII military analog computers for solution of ballistics equations, artillery calculations, and target tracking. Various other integrators are described in Morin (1913).

**Harmonic analyzers.** A *harmonic analyzer* is a mechanical device that calculates the coefficients of the Fourier transform of a complex function of time such as a sound wave. Early harmonic analyzers were developed by Thomson (1878) and Henrici (1894) using multiple pulleys and spheres, known as ball-and-disk integrators.

**Harmonic synthesizers.** A *harmonic synthesizer* is a mechanical device that interpolates a function given the Fourier coefficients. Thomson (then known as Lord Kelvin) in 1886 developed (Kelvin 1878) the first known harmonic analyzer that used an array of James Thomson's (his brother) ball-and-disk integrators. Kelvin's harmonic synthesizer made use of these Fourier coefficients to reverse this process and interpolate function values, by

using a wire wrapped over the wheels of the array to form a weighted sum of their angular rotations. Kelvin demonstrated that the use of these analog devices is to predict the tide heights of a port: first, his harmonic analyzer calculated the amplitude and phase of the Fourier harmonics of solar and lunar tidal movements, and then his harmonic synthesizer formed their weighted sum, to predict the tide heights over time. Many other harmonic analyzers were later developed, including one by Michelson and Stratton (1898) that performed Fourier analysis, using an array of springs. Miller (1916) gives a survey of these early harmonic analyzers. Fisher (1911) made improvements to the tide predictor, and later Doodson and Lége increase the scale of this design to a 42-wheel version that was used up to the early 1960s.

**Analog equation solvers.** There are various mechanical devices for calculating the solution of sets of equations. Kelvin also developed one of the first known mechanical mechanisms for equation solving, involving the motion of pulleys and tilting plate that solved sets of simultaneous linear equations specified by the physical parameters of the ropes and plates. John Wilbur in the 1930s increased the scale of Kelvin's design to solve nine simultaneous linear algebraic equations. Leonardo Torres Quevedo constructed various rotational mechanical devices, for determining real and complex roots of a polynomial. Svoboda (1948) describes the state of art in the 1940s of mechanical analog computing devices using linkages.

**Differential analyzers.** A *differential analyzer* is a mechanical analog device using linkages for solving ordinary differential equations. Vannevar Bush (1931) developed in 1931 the first differential analyzer at MIT that used a torque amplifier to link multiple mechanical integrators. Although it was considered a general-purpose mechanical analog computer, this device required a physical reconfiguration of the mechanical connections to specify a given mechanical problem to be solved. In subsequent differential analyzers, the

reconfiguration of the mechanical connections was made automatic by resetting electronic relay connections. In addition to the military applications already mentioned above, analog mechanical computers incorporating differential analyzers have been widely used for flight simulations and for industrial control systems.

**Mechanical simulations of Physical Processes: Crystallization and Packing.** There are a variety of macroscopic devices used for simulations of physical processes, which can be viewed as analog devices. For example, a number of approaches have been used for mechanical simulations of crystallization and packing:

*Simulation using solid macroscopic ellipsoid bodies.* Simulations of kinetic crystallization processes have been made by collections of macroscopic solid ellipsoidal objects – typically of diameter of a few millimeters – which model the molecules comprising the crystal. In these physical simulations, thermal energy is modeled by introducing vibrations; low level of vibration is used to model freezing and increasing the level of vibrations models melting. In simple cases, the molecule of interest is a sphere, and ball bearings or similar objects are used for the molecular simulation. For example, to simulate the dense random packing of hard spheres within a crystalline solid, Bernal (1964) and Finney (1970) used up to 4000 ball bearings on a vibrating table. In addition, to model more general ellipsoidal molecules, orzo pasta grains as well as M&M candies (Jerry Gollub at Princeton University) have been used. Also, Cheerios have been used to simulate the liquid state packing of benzene molecules. To model more complex systems, mixtures of balls of different sizes and/or composition have been used; for example, a model ionic crystal formation has been made by using a mixture of balls composed of different materials that acquired opposing electrostatic charges.

*Simulations using bubble rafts* (Bragg and Nye 1947; Bragg and Lomer 1948). These are the structures that assemble among equal-sized bubbles floating on water. They typically form two-dimensional hexagonal arrays and

can be used for modeling the formation of close-packed crystals. Defects and dislocations can also be modeled (Corcoran et al. 1997); for example, by deliberately introducing defects in the bubble rats, they have been used to simulate crystal dislocations, vacancies, and grain boundaries. Also, impurities in crystals (both interstitial and substitutional) have been simulated by introducing bubbles of other sizes.

### Reaction-Diffusion Chemical Computers.

Adamatzky (Adamatzky et al. 2005; Adamatzky 1998b) described a class of analog computers where there is a chemical medium which has multiple chemical species, where the concentrations of these chemical species vary spatially and which diffuse and react in parallel. The memory values (as well as inputs and outputs) of the computer are encoded by the concentrations of these chemical species at a number of distinct locations (also known as micro-volumes). The computational operations are executed by chemical reactions whose reagents are these chemical species. Example computations (Adamatzky et al. 2005; Adamatzky 1998b) include (i) Voronoi diagram, which determines the boundaries of the regions closest to a set of points on the plane, (ii) skeleton of planar shape, and (iii) a wide variety of two-dimensional patterns periodic and aperiodic in time and space.

## Digital Mechanical Devices for Arithmetic Operations

Recall that we have distinguished digital mechanical devices from the analog mechanical devices described above by their use of mechanical mechanisms for insuring the values stored and computed are discrete. Such discretization mechanisms include geometry and structure (e.g., the notches of Napier's bones described below) or cogs and spokes of wheeled calculators. Surveys of the history of some of these digital mechanical calculators are given by Knott (1915), Turck (1921), Hartree (1950), Engineering Research Associates (1950), Chase (1980), Martin (1992), Davis (2000), and Norman (2002):

**Leonardo da Vinci's mechanical device and mechanical counting devices.** This intriguing device, which involved a sequence of interacting wheels positioned on a rod, which appear to provide a mechanism for digital carry operations, was illustrated in 1493 in Leonardo da Vinci's Codex Madrid (1493). A working model of its possible mechanics was constructed in 1968 by Joseph Mirabella. Its function and purpose is not decisively known, but it may have been intended for counting rotations (e.g., for measuring the distance traversed by a cart). There are a variety of apparently similar mechanical devices used to measuring distances traversed by vehicles.

**Napier's bones.** John Napier (1614) developed in 1614 a mechanical device known as Napier's bones that allowed multiplication and division (as well as square and cube roots) to be done by addition and multiplication operations. It consists of rectilinear rods, which provided a mechanical transformation to and from logarithmic values. Wilhelm Schickard developed in 1623 a six-digit mechanical calculator that combined the use of Napier's bones using columns of sliding rods, with the use of wheels used to sum up the partial products for multiplication.

**Slide rules.** Edmund Gunter devised in 1620 a method for calculation that used a single log scale with dividers along a linear scale; this anticipated key elements of the first slide rule described by William Oughtred (1632) in 1632. A very large variety of slide machines were later constructed.

**Pascaline: Pascal's wheeled calculator.** Blaise Pascal (1645) developed in 1642 a calculator known as the Pascaline that could calculate all four arithmetic operations (addition, subtraction, multiplication, and division) on up to eight digits. A wide variety of mechanical devices were then developed that used revolving drums or wheels (cogwheels or pinwheels) to do various arithmetical calculations.

**Stepped drum calculators.** Gottfried Wilhelm von Leibniz developed in 1671 an improved calculator known as the stepped reckoner, which used a cylinder known as a *stepped*

*drum* with nine teeth of different lengths that increase in equal amounts around the drum. The stepped drum mechanism allowed the use of moving slide for specifying a number to be inputted to the machine and made use of the revolving drums to do the arithmetic calculations. Charles Xavier Thomas de Colbrar developed in 1820 a widely used arithmetic mechanical calculator based on the stepped drum known as the arithmometer. Other stepped drum calculating devices included Otto Shweiger's millionaire calculator (1893) and Curt Herzstark's curta (early 1940s).

**Pinwheel calculators.** Another class of calculators, independently invented by Frank S. Baldwin and W. T. Odhner in the 1870s, is known as pinwheel calculators; they used a pinwheel for specifying a number input to the machine and use revolving wheels to do the arithmetic calculations. Pinwheel calculators were widely used up to the 1950s, for example, in William S. Burroughs's calculator/printer and the German Brunsviga.

## Digital Mechanical Devices for Mathematical Tables and Functions

**Babbage's difference engine.** Charles Babbage (1822, 1825) in 1820 invented a mechanical device known as the *difference engine* for calculation of tables of an analytical function (such as the logarithm) that summed the change in values of the function when a small difference is made in the argument. That difference calculation required for each table entry involved a small number of simple arithmetic computations. The device made use of columns of cogwheels to store digits of numerical values. Babbage planned to store 1000 variables, each with 50 digits, where each digit was stored by a unique cogwheel. It used cogwheels in registers for the required arithmetical calculations and also made use of a rod-based control mechanism specialized for control of these arithmetic calculations. The design and operation of the mechanisms of the device were described by a symbolic scheme developed by Babbage

(1826). He also conceived of a printing mechanism for the device. In 1801, Joseph-Marie Jacquard invented an automatic loom that made use of punched cards for the specification of fabric patterns woven by his loom, and Charles Babbage proposed the use of similar punched cards for providing inputs to his machines. He demonstrated over a number of years certain key portions of the mechanics of the device but never completed a complete function device.

**Other difference engines.** In 1832 Ludgate (1909) independently designed, but did not construct, a mechanical computing machine similar but smaller in scale to Babbage's analytical engine. In 1853 Pehr and Edvard Scheutz (Lindgren 1990) constructed in Sweden a cogwheel mechanical calculating device (similar to the difference engine originally conceived by Babbage) known as the tabulating machine for computing and printing out tables of mathematical functions. This (and a later construction of Babbage's difference engine by Doron Swade (1991) of the London Science Museum) demonstrated the feasibility of Babbage's difference engine.

**Babbage's analytical engine.** Babbage further conceived (but did not attempt to construct) a mechanical computer known as the analytical engine to solve more general mathematical problems. Lovelace's extended description of Babbage's analytical engine (Lovelace 1843) (translation of "Sketch of the Analytical Engine" by L. F. Menabrea) describes, in addition to arithmetic operations, also mechanisms for looping and memory addressing. However, the existing descriptions of Babbage's analytical engine lack the ability to execute a full repertory of logical and/or finite state transition operations required for general computation. Babbage's background was very strong in analytic mathematics, but he (and the architects of similar cogwheel-based mechanical computing devices at that date) seemed to have lacked knowledge of sequential logic and its Boolean logical basis, required for controlling the sequence of complex computations. This (and his propensity for changing designs prior to the completion of the machine construction) might have been the real reason for

the lack of complete development of a universal mechanical digital computing device in the early 1800s.

**Subsequent electromechanical digital computing devices with cogwheels.** Other electromechanical computing digital devices (see Engineering Research Associates Staff 1950) developed in the late 1940s and 1950s that contain cogwheels included Howard Aiken's Mark 1 (Cohen and Welch 1999) constructed at Harvard University and Konrad Zuse's Z series computer constructed in Germany.

## Mechanical Devices for Timing, Sequencing, and Logical Control

We will use the term *mechanical automata* here to denote mechanical devices that exhibit autonomous control of their movements. These can require sophisticated mechanical mechanisms for timing, sequencing, and logical control:

**Mechanisms used for timing control.** Mechanical clocks and other mechanical devices for measuring time have a very long history and include a very wide variety of designs, including the flow of liquids (e.g., water clocks) or sands (e.g., sand clocks), and more conventional pendulum-and-gear-based clock mechanisms. A wide variety of mechanical automata and other control devices make use of mechanical timing mechanisms to control the order and duration of events automatically executed (e.g., mechanical slot machines dating up to the 1970s made use of such mechanical clock mechanisms to control the sequence of operations used for payout of winnings). As a consequence, there is an interwoven history in the development of mechanical devices for measuring time and the development of devices for the control of mechanical automata.

**Logical control of computations.** A critical step in the history of computing machines was the development in the middle 1800s of Boolean logic by George Boole (1847, 1854). Boole's innovation was to assign values to logical propositions: 1 for true propositions and 0 for

false propositions. He introduced the use of Boolean variables which are assigned to these values, as well as the use of Boolean connectives (and and or), for expressing symbolic Boolean logic formulas. Boole's symbolic logic is the basis for the logical control used in modern computers. Shannon (1938) was the first to make use of Boole's symbolic logic to analyze relay circuits (these relays were used to control an analog computer, namely, MITs Differential Equalizer).

#### **The Jevons' logic piano: a mechanical logical inference machine.**

In 1870 William Stanley Jevons (who also significantly contributed to the development of symbolic logic) constructed a mechanical device (Jevons 1870, 1873) for the inference of logical proposition that used a piano keyboard for inputs. This *mechanical inference machine* is less widely known than it should be, since it may have had impact in the subsequent development of logical control mechanisms for machines.

#### **Mechanical logical devices used to play games.**

Mechanical computing devices have also been constructed for executing the logical operations for playing games. For example, in 1975, a group of MIT undergraduates including Danny Hillis and Brian Silverman constructed a computing machine made of Tinkertoys that plays a perfect game of tic-tac-toe.

## **Mechanical Devices Used in Cryptography**

#### **Mechanical cipher devices using cogwheels.**

Mechanical computing devices that used cogwheels were also developed for a wide variety of other purposes beyond merely arithmetic. A wide variety of mechanical computing devices were developed for the encryption and decryption of secret messages. Some of these (most notably the family of German electromechanical cipher devices known as *Enigma Machines* (Hamer et al. 1998) developed in the early 1920s for commercial use and refined in the late 1920s and 1930s for military use) made use of sets of

cogwheels to permute the symbols of text message streams. Similar (but somewhat more advanced) electromechanical cipher devices were used by the USSR up to the 1970s.

**Electromechanical computing devices used in breaking ciphers.** In 1934 Marian Rejewski and a team including Alan Turing constructed an electrical/mechanical computing device known as the bomb, which had an architecture similar to the abstract Turing machine described below and which was used to decrypt ciphers made by the German Enigma cipher device mentioned above.

## **Mechanical and Electrooptical Devices for Integer Factorization**

**Lehmer's number sieve computer.** In 1926 Derrick Lehmer (1928) constructed a mechanical device called the number sieve computer for various mathematical problems in number theory including factorization of small integers and solution of Diophantine equations. The device made use of multiple bicycle chains that rotated at distinct periods to discover solutions (such as integer factors) to these number theoretic problems.

**Shamir's TWINKLE.** Adi Shamir (n.d., 1999; Lenstra and Shamir 2000) proposed a design for an optical/electric device known as TWINKLE for factoring integers, with the goal of breaking the RSA public-key cryptosystem. This was unique among mechanical computing devices in that it used time durations between optical pulses to encode possible solution values. In particular, LEDs were made to flash at certain intervals of time (where each LED is assigned a distinct period and delay) at a very high clock rate so as to execute a sieve-based integer factoring algorithm.

**Mechanical Computation at the Microscale: MEMS Computing Devices.** Mechanical computers can have advantages over electronic computation at certain scales; they are already having widespread use at the microscale. Microelectromechanical systems (MEMSs) are manufactured by lithographic etching methods similar in nature to the processes microelectronics are manufactured and have a similar microscale.

A wide variety of MEMS devices (Madou 2002) have been constructed for sensors and actuators, including accelerometers used in automobile safety devices and disk readers, and many of these MEMS devices execute mechanical computation to do their task. Perhaps the MEMS device most similar in architecture to the mechanical calculators described above is the recordable locking device (Plummer et al. 1999) constructed in 1998 at Sandia Labs, which made use of microscopic gears that acted as a mechanical lock and which was intended for mechanically locking strategic weapons.

## Future Directions

### Mechanical Self-Assembly Processes

Most of the mechanical devices discussed in this chapter have been assumed to be constructed top-down; that is, they are designed and then assembled by other mechanisms generally of large scale. However, a future direction to consider is bottom-up processes for assembly and control of devices. Self-assembly is a basic bottom-up process found in many natural processes and in particular in all living systems:

**Domino tiling problems.** The theoretical basis for self-assembly has its roots in domino tiling problems (also known as Wang tilings) as defined by Wang (1963) (also see the comprehensive text of Grunbaum et al. (1987)). The input is a finite set of unit size square tiles, each of whose sides is labeled with symbols over a finite alphabet. Additional restrictions may include the initial placement of a subset of these tiles and the dimensions of the region where tiles must be placed. Assuming an arbitrarily large supply of each tile, the problem is to place the tiles, without rotation (a criterion that cannot apply to physical tiles), to completely fill the given region so that each pair of abutting tiles has identical symbols on their contacting sides.

**Turing-universal and NP complete self-assemblies.** Domino tiling problems over an infinite domain with only a constant number of tiles were first proved by Berger (1966) to be undecidable. Lewis and Papadimitriou (1981)

showed the problem of tiling a given finite region is NP complete.

### Theoretical models of tiling self-Assembly processes.

Domino tiling problems do not presume or require a specific process for tiling. Winfree (Winfree et al. 1996) proposed kinetic models for self-assembly processes. The sides of the tiles are assumed to have some methodology for selective affinity, which we call pads. Pads function as programmable binding domains, which hold together the tiles. Each pair of pads has specified binding strengths (a real number on the range [0,1] where 0 denotes no binding and 1 denotes perfect binding). The self-assembly process is initiated by a singleton tile (the seed tile) and proceeds by tiles binding together at their pads to form aggregates known as tiling assemblies. The preferential matching of tile pads facilitates the further assembly into tiling assemblies.

**Pad binding mechanisms.** These provide a mechanism for the preferential matching of tile sides can be provided by various methods:

*Magnetic attraction*, e.g., pads with magnetic orientations (these can be constructed by curing ferrite materials (e.g., PDMS polymer/ferrite composites) in the presence of strong magnet fields) and also pads with patterned strips of magnetic orientations

*Capillary force*, using hydrophobic/hydrophilic (capillary) effects at surface boundaries that generate lateral forces

*Shape matching* (also known as *shape complementarity* or *conformational affinity*), using the shape of the tile sides to hold them together

Also see the sections below *discussion of the used of molecular affinity* for pad binding.

**Materials for tiles.** There are a variety of distinct materials for tiles, at a variety of scales: Whitesides (see Xia and Whitesides (1998) and <http://www-chem.harvard.edu/GeorgeWhitesides.html>) has developed and tested multiple technologies for mesoscale self-assembly, using capillary forces, shape complementarity, and magnetic forces. Rothemund (Rothemund 2000) experimentally demonstrated some of the most complex known mesoscale tiling self-assemblies using polymer tiles on fluid

boundaries with pads that use hydrophobic/hydrophilic forces. A material science group at the University of Wisconsin (<http://mrsec.wisc.edu/edetc/selfassembly>) has also tested mesoscale self-assembly using magnetic tiles.

**Mesoscale tile assemblies.** Mesoscale tiling assemblies have tiles of size a few millimeters up to a few centimeters. They have been experimentally demonstrated by a number of methods, such as placement of tiles on a liquid surface interface (e.g., at the interface between two liquids of distinct density or on the surface of an air/liquid interface) and use of mechanical agitation with shakers to provide a heat source for the assembly kinetics (i.e., a temperature setting is made by fixing the rate and intensity of shaker agitation).

**Applications of mesoscale assemblies.** There are a number of applications, including:  
Simulation of the thermodynamics and kinetics of molecular-scale self-assemblies  
For placement of a variety of microelectronics and MEMS parts

### Computation at the Molecular Scale: DNA Computing Devices.

**DNA Computing Devices.** Due to the difficulty of constructing electrical circuits at the molecular scale, alternative methods for computation, and in particular mechanical methods, may provide unique opportunities for computing at the molecular scale. In particular the bottom-up self-assembly processes described above have unique applications at the molecular scale:

### Self-assembled DNA nanostructures.

Molecular-scale structures known as *DNA nanostructures* (see surveys by Seeman 2004; Reif et al. (Reif and LaBean 2009)) can be made to self-assemble from individual synthetic strands of DNA. When added to a test tube with the appropriate buffer solution and the test tube is cooled, the strands self-assemble into DNA nanostructures. This self-assembly of DNA nanostructures can be viewed as a mechanical process and in fact can be used to do computation. The first known example of a computation by using DNA was by Adleman (1994, 1998) in 1994; he used the self-assembly

of DNA strands to solve a small instance of a combinatorial optimization problem known as the Hamiltonian path problem.

**DNA tiling assemblies.** The Wang tiling (1963) paradigm for self-assembly was the basis for scalable and programmable approach proposed by Winfree et al. (1998) for doing molecular computation using DNA. First, a number of distinct DNA nanostructures known as DNA tiles are self-assembled. End portions of the tiles, known as pads, are designed to allow the tiles to bind together a programmable manner similar to Wang tiling but in this case use the molecular affinity for pad binding due to hydrogen bonding of complementary DNA bases. This programmable control of the binding together of DNA tiles provides a capability for doing computation at the molecular scale. When the temperature of the test tube containing these tiles is further lowered, the DNA tiles bind together to form complex patterned tiling lattices that correspond to computations.

### Assembling patterned DNA tiling assemblies.

Programmed patterning at the molecular scale can be produced by the use of strands of DNA that encode the patterns; this was first done by Yan et al. (2003a) in the form of bar-cord-striped patterns, and more recently Rothemund (2006) self-assembled complex 2D molecular patterns and shapes using a technique known as DNA origami. Another method for molecular patterning of DNA tiles is via computation done during the assembly, as described below.

**Computational DNA tiling assemblies.** The first experimental demonstration of computation via the self-assembly of DNA tiles was in 2000 by Mao et al. (2000, Yan et al. 2003b), which provided a one-dimensional computation of a binary-carry computation (known as prefix sum) associated with binary adders. Rothemund et al. (2004) in 2004 demonstrated a two-dimensional computational assemblies of tiles displaying a pattern known as the Sierpinski triangle, which is the modulo 2 version of Pascal's triangle.

**Other autonomous DNA devices.** DNA nanostructures can also be made to make sequences of movement, and a demonstration of an autonomous moving DNA robotic device that

moved without outside mediation across DNA nanostructures was given by Yin et al. (2004). The design of an autonomous DNA device that moves under programmed control is described in (Reif and Sahu 2008). Surveys of DNA autonomous devices are given in Reif and LaBean (2007), Chandran et al. (2013), and Bath and Turberfield (2007).

### Analog Computation by Chemical Reactions

*Chemical reaction systems* have a set of reactants, whose concentrations evolve by a set of differential equations determined by chemical reactions. Magnasco (1997) showed that a chemical reaction can be designed to simulate any given digital circuit. Soloveichik et al. (2008) showed that a chemical reaction can be designed to simulate a universal Turing machine, and Soloveichik et al. (2010) showed that this can be done by a class of chemical reactions involving only DNA hybridization reactions. Senum and Riedel (2011) gave detailed design rules for chemical reactions that executed various analog computational operations such as addition, multipliers, and logarithm calculations.

### Analog Computation by Bacterial Genetic Circuits

Sarpeshkar et al. have developed analog transistor models for the concentrations of various reactants within bacterial genetic circuits (Danial et al. 2011) and then used these models to experimentally demonstrate various computations, such as square root calculation, within living bacteria cells (Daniel et al. 2013).

## Bibliography

- Adamatzky AI (1996) On the particle-like waves in the discrete model of excitable medium. *Neural Netw World* 1:3–10
- Adamatzky AI (1998a) Universal dynamical computation in multidimensional excitable lattices. *Int J Theory Phys* 37:3069–3108
- Adamatzky AI (1998b) Chemical processor for computation of voronoi diagram. *Adv Mater Opt Electron* 6(4):191–196
- Adamatzky A (ed) (2001) Collision-based computing. Springer, London

- Adamatzky A, De Lacy CB, Asai T (2005) Reaction-diffusion computers. Elsevier, New York. isbn:0444520422
- Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266(11): 1021–1024
- Adleman L (1998) Computing with DNA. *Sci Am* 279(2):34–41
- Babbage C (1822) On machinery for calculating and printing mathematical tables. *Edinb Philos J*. In: Jameson R, Brewster D (eds) vol VII. Archibald Constable, Edinburgh, pp 274–281
- Babbage C (1825) Observations on the application of machinery to the computation of mathematical tables. *Phil Mag J LXV*:311–314. London: Richard Taylor
- Babbage C (1826) On a method of expressing by signs the action of machinery. *Philos Trans R Soc Lond* 116(Part III):250–265
- Bath J, Turberfield AJ (2007) DNA nanomachines. *Nat Nanotechnol* 2:275–284
- Benioff P (1982) Quantum mechanical models of Turing machines that dissipate no energy. *Phys Rev Lett* 48:1581
- Bennett CH (1973) Logical reversibility of computation. *IBM J Res Dev* 17(6):525–532
- Bennett CH (1982) The thermodynamics of computation – a review. *Int J Theor Phys* 21(12):905–940
- Bennett CH (2003) Notes on Landauer’s principle, reversible computation, and Maxwell’s demon. *Stud Hist Philos Mod Phys* 34:501–510. eprint physics/0210005
- Berger R (1966) The undecidability of the domino problem. *Mem Am Math Soc* 66:1–72
- Bernal JD (1964) The structure of liquids. *Proc R Soc Lond Ser A* 280:299
- Blum L, Cucker F, Shub M, Smale S (1996) Complexity and real computation: a manifesto. *Int J Bifurc Chaos* 6(1):3–26. World Scientific, Singapore
- Boole G (1847) Mathematical analysis of logic: the mathematical analysis of logic: Being an essay towards a calculus of deductive reasoning, pamphlet
- Boole G (1854) An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilities. Macmillan, London
- Bragg L, Lomer WM (1948) A dynamical model of a crystal structure II. *Proc R Soc A* 196:171–181
- Bragg L, Nye JF (1947) A dynamical model of a crystal structure. *Proc R Soc A* 190:474–481
- Bush V (1931) The differential analyzer: a new machine for solving differential equations. *J Frankl Inst* 212:447
- Canny J (1988) Some algebraic and geometric computations in PSPACE. In: Cole R (ed) Proceedings of the 20th annual ACM symposium on the theory of computing. ACM Press, Chicago, pp 460–467
- Canny J, Reif JH (1987) New lower bound techniques for robot motion planning problems. In: 28th annual IEEE symposium on foundations of computer science, Los Angeles, pp 49–60
- Canny J, Donald B, Reif JH, Xavier P (1988) On the complexity of kinodynamic planning. In: 29th annual IEEE symposium on foundations of computer science, White Plains, pp 306–316. Published as Kinodynamic motion planning *J ACM* 40(5): 1048–1066 (1993)

- Chandran H, Gopalkrishnan N, Reif J (2013) In: Mavroidis C, Ferreira A (eds) DNA nanoRobotics, chapter, nanorobotics: current approaches and techniques. Springer, New York, pp 355–382. ISBN 13 : 9781461421184, ISBN 10 : 1461421187
- Chase GC (1980) History of mechanical computing machinery. *IEEE Ann Hist Comput* 2(3):198–226
- Cohen IB, Welch GW (1999) Makin' numbers: Howard Aiken and the computer. MIT Press, Cambridge, MA
- Corcoran SG, Colton RJ, Lilleodden ET, Gerberich WW (1997) *Phys Rev B* 190:474
- Crescenzi P, Papadimitriou CH (1995) Reversible simulation of space-bounded computations. *Theor Comput Sci* 143(1):159–165
- da Vinci L (1493) Codex Madrid I
- Danial R, Woo SS, Turicchia L, Sarpeshkar R (2011) Analog transistor models of bacterial genetic circuits. In: Proceedings of the 2011 I.E. biological circuits and systems (BioCAS) conference, San Diego, pp 333–336
- Daniel R, Rubens J, Sarpeshkar R, Lu T (2013) Synthetic analog computation in living cells. *Nature*. <https://doi.org/10.1038/nature12148>
- Davis M (2000) The universal computer: the road from Leibniz to Turing. Norton Press, Norton
- Deutsch D (1985) Quantum theory, the church-Turing principle and the universal quantum computer. *Proc R Soc Lond A* 400:97–117
- Dolev S, Fitoussi H (2010) Masking traveling beams: optical solutions for NP-complete problems, trading space for time. *Theor Comput Sci* 411:837–853
- Engineering Research Associates Staff (1950) High-speed computing devices. McGraw-Hill Book, New York City
- Feynman RP (1963) “ratchet and pawl”, chapter 46. In: Feynman RP, Leighton RB, Sands M (eds) The Feynman lectures on physics, vol 1. Addison-Wesley, Reading
- Feynman RP (1982) Simulating physics with computers. *Int J Theor Phys* 21(6/7):467–488
- Finney JL (1970) Random packings and the structure of simple liquids. I The geometry of random close packing. *Proc R Soc Lond A Math Phys Sci* 319(1539):479–493
- Fisher EG (1911) Tide-predicting machine. *Eng News* 66:69–73
- Fredkin E, Toffoli T (1982) Conservative logic. *Int J Theory Phys* 21:219–253
- Freeth T, Bitsakis Y, Moussas X, Seiradakis JH, Tselikas A, Mangou H, Zafeiropoulou M, Hadland R, Bate D, Ramsey A, Allen M, Crawley A, Hockley P, Malzbender T, Gelb D, Ambrisco W, Edmunds MG (2006) Decoding the ancient Greek astronomical calculator known as the Antikythera mechanism. *Nature* 444:587–591
- Goliaei S, Foroughmand-Araabi M (2013) Light ray concentration reduces the complexity of the wavelength-based machine on PSPACE languages, unpublished manuscript
- Goliaei S, Jalili S (2009) An optical wavelength-based solution to the 3-SAT problem. In: Dolev S, Oltean M (eds) Optical supercomputing, Lecture notes in computer science, vol. 5882, pp 77–85
- Goliaei S, Jalili S (2012) An optical wavelength-based computational machine. *Unconventional computation and natural computation lecture notes in computer science*, vol 7445, pp 94–105. Also, *Int J Unconv Comput* (in press)
- Grunbaum S, Branko SGC (1987) Tilings and patterns, chapter 11. H Freeman, San Francisco
- Gruska J (1999) Quantum computing. McGraw-Hill, New York
- Haist T, Osten W (2007) An optical solution for the traveling salesman problem. *Opt Express* 15(16): 10473–10482
- Hamer D, Sullivan G, Weierud F (1998) Enigma variations: an extended family of machines. *Cryptologia* 22(3):211–229
- Hartree DR (1950) Calculating instruments and machines. Cambridge University Press, London
- Henrici (1894) On a new harmonic analyzer, *Phil Mag* 38:110
- Hopcroft JE, Schwartz JT, Sharir M (1984) On the complexity of motion planning for multiple independent objects: PSPACE hardness of the warehouseman's problem. *Int J Robot Res* 3(4):76–88
- Horsburgh EM (1914) Modern instruments of calculation. G. Bell & Sons, London, p 223
- Jaeger G (2006) Quantum information: an overview. Springer, Berlin
- Jakubowski MH, Steiglitz K, Squier R (1998) State transformations of colliding optical solitons and possible application to computation in bulk media. *Phys Rev E* 58:6752–6758
- Jakubowski MH, Steiglitz K, Squier R (2001) Computing with solitons: a review and prospectus, collision-based computing. Springer, London, pp 277–297
- Jevons WS (1870) On the mechanical performance of logical inference. *Philos Trans R Soc* 160(Part II): 497–518
- Jevons SW (1873) The principles of science; a treatise on logic and scientific method. Macmillan, London
- Kelvin L (1878) Harmonic analyzer and synthesizer. *Proc R Soc* 27:371
- Knott CG (ed) (1915) Napier tercentenary memorial volume. Published for the Royal Society of Edinburgh by Longmans, Green, London
- Landauer R (1961) Irreversibility and heat generation in the computing process. *IBM J Res Dev* 5:183
- Lehmer DH (1928) The mechanical combination of linear forms. *Am Math Mon* 35:114–121
- Lenstra AK, Shamir A (2000) Analysis and optimization of the TWINKLE factoring device, proc. Eurocrypt 2000, LNCS 1807. Springer, Heidelberg, pp 35–52
- Lewis HR, Papadimitriou CH (1981) Elements of the theory of computation. Prentice-Hall, Upper Saddle River, pp 296–300. and 345–348
- Lewis HR, Papadimitriou CH (1997) Elements of the theory of computation, 2nd edn. Prentice Hall, Upper Saddle River
- Li M, Vitanyi P (1996) Reversibility and adiabatic computation: trading time and space for energy. *Proc R Soc*

- Lond Ser A 452:769–789. (Online preprint quant-ph/9703022)
- Lindgren M (1990) Glory and failure: difference engines of Johann Muller, Charles Babbage and Georg and Edvard Scheutz. MIT Press, Cambridge, MA
- Lovelace A, translation of “Sketch of the Analytical Engine” by L. F. Menabrea with Ada’s notes and extensive commentary. Ada Lovelace (1843) Sketch of the analytical engine invented by Charles Babbage. Esq Scientific Memoirs 3:666–731
- Ludgate P (1909–1910) On a proposed analytical engine. Sci Proc Roy Dublin Soc 12:77–91
- Madou MJ (2002) Fundamentals of microfabrication: the science of miniaturization, 2nd edn. CRC Publishers, Boca Raton
- Magnasco MO (1997) Chemical kinetics is Turing universal. Phys Rev Lett 78:1190–1193
- Mao C, LaBean TH, Reif JH, Seeman NC (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature 407:493–495
- Martin E (1992) The calculating machines. The MIT Press, Cambridge, MA
- Miller D (1916) The Henrici harmonic analyzer and devices for extending and facilitating its use. J Franklin Inst 181:51–81 and 182:285–322
- Moore C (1990) Undecidability and unpredictability in dynamical systems. Phys Rev Lett 64:2354–2357
- Moore C (1991) Generalized shifts: undecidability and unpredictability in dynamical systems. Nonlinearity 4:199–230
- de Morin H (1913) Les appareils d’intégration: intégrateurs simples et composés; planimètres; intéromètres; intégraphes et courbes intégrales; analyse harmonique et analyseurs. Gauthier-Villars Publishers, Paris
- Munakata T, Sinha S, Ditto WL (2002) Chaos computing: implementation of fundamental logical gates by chaotic elements. IEEE Trans Circ Syst-I Fundam Theory Appl 49(11):1629–1633
- Muntean O, Oltean M (2009) Deciding whether a linear diophantine equation has solutions by using a light-based device. J Optoelectron Adv Mater 11(11): 1728–1734
- Napier J (1614) Mirifici logarithmorum canonis descriptio (the description of the wonderful canon of logarithms). Hart, Edinburgh
- Nielsen M, Chuang I (2000) Quantum computation and quantum information. Cambridge University Press, Cambridge
- Norman JM (ed) (2002) The origins of cyberspace: from Gutenberg to the internet: a sourcebook on the history of information technology. Norman Publishing, Novato
- Oltean M (2008) Solving the Hamiltonian path problem with a light-based computer. Nat Comput 6(1):57–70
- Oltean M, Muntean O (2008) Exact cover with light. New Gener Comput 26(4):329–346
- Oltean M, Muntean O (2009) Solving the subset-sum problem with a light-based device. Nat Comput 8(2):321–331
- Oughtred W (1632) Circles of proportion and the horizontal instrument. Translated and Published by William Forster, London
- Pascal E (1645) Lettre dédicatoire à Monseigneur le Chancelier sur le sujet de la machine nouvellement inventée par le sieur B. P pour faire toutes sortes d’opérations d’arithmétique par un mouvement réglé sans plume ni jetons, suivie d’un avis nécessaire à ceux qui auront curiosité de voir ladite machine et de s’en servir
- Plummer D, Dalton LJ, Peter F (1999) The recodable locking device. Commun ACM 42(7):83–87
- Reif JH (1979) Complexity of the mover’s problem and generalizations. In: 20th Annual IEEE symposium on foundations of computer science, San Juan, Puerto Rico, pp 421–427. Also appearing in Chapter 11 in Planning, geometry and complexity of robot motion. Schwartz J (ed) Ablex Pub, Norwood, pp 267–281 (1987)
- Reif JH (2009a) Quantum information processing: algorithms, technologies and challenges, invited chapter. In: Eshaghian-Wilner MM (ed) Nano-scale and bio-inspired computing. Wiley, Hoboken
- Reif JH (2009b) Mechanical computation: it’s computational complexity and technologies, invited chapter. In: Meyers RA (ed) Encyclopedia of complexity and system science. Unconventional Computing (Section Editor: Andrew Adamatzky). Springer, New York, ISBN: 978-0-387-75888-6
- Reif JH, LaBean TH (2007) Autonomous programmable biomolecular devices using self-assembled DNA nanostructures, communications of the ACM (CACM), Special Section entitled “New Computing Paradigms (edited by Toshinori Munakata)
- Reif JH, LaBean TH (2009) Nanostructures and autonomous devices assembled from DNA. Invited chapter. In: Eshaghian-Wilner MM (ed) Nano-scale and bio-inspired computing. Wiley, Hoboken
- Reif JH, Sahu S (2008) Autonomous programmable DNA nanorobotic devices using DNAzymes, 13th international meeting on DNA computing (DNA 13), Memphis, June 4–8, 2007. In: Garzon M, Yan H (eds) DNA computing: DNA13, Springer-Verlag lecture notes for computer science (LNCS), vol 4848. Springer, Berlin, pp 66–78. Published in Special Journal Issue on Self-Assembly, Theoretical Computer Science (TCS) 410(15):1428–1439 (2009)
- Reif JH, Sharir M (1985) Motion planning in the presence of moving obstacles. In: 26th annual IEEE symposium on foundations of computer science, Portland, pp 144–154. Published in Journal of the ACM (JACM) 41:4, pp 764–790 (1994)
- Reif JH, Sun Z (1998) The computational power of frictional mechanical systems. In: Third international workshop on algorithmic foundations of robotics, (WAFR98), Pub. by A. K. Peters Ltd, Houston, pp 223–236. Published as On frictional mechanical systems and their computational power, SIAM Journal of Computing(SICOMP) 32(6):1449–1474 (2003)
- Reif JH, Wang H (1998) The complexity of the two dimensional curvature-constrained shortest-path problem. In: Third international workshop on algorithmic foundations of robotics (WAFR98), Pub. by A. K. Peters Ltd, Houston, pp 49–57

- Reif JH, Tygar D, Yoshida A (1990) The computability and complexity of optical beam tracing. 31st annual IEEE symposium on foundations of computer science, St. Louis, pp 106–114. Published as The computability and complexity of ray tracing in discrete & computational geometry 11:265–287 (1994)
- Reif J, Chandran H, Gopalkrishnan N, LaBean T (2012) Self-assembled DNA nanostructures and DNA devices. Invited chapter 14. In: Cabrini S, Kawata S (eds) Nanofabrication handbook. CRC Press, Taylor and Francis Group, New York, pp 299–328. isbn13:9781420090529, isbn10: 1420090526
- Rothenmund PWK (2000) Using lateral capillary forces to compute by self-assembly. Proc Natl Acad Sci USA 97:984–989
- Rothenmund PWK (2006) Folding DNA to create nanoscale shapes and patterns. Nature 440:297–302
- Rothenmund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2(12): electronic pub. e424. <https://doi.org/10.1371/journal.pbio.0020424>
- Schwartz JT, Sharir M (1983) On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. Commun Pure Appl Math 36:345–398
- Seeman NC (2004) Nanotechnology and the double helix. Sci Am 290(6):64–75
- Senum P, Riedel M (2011) Rate-independent constructs for chemical computation. PLoS One 6(6):e21414
- Shamir A (1999) Factoring large numbers with the TWIN-KLE device, cryptographic hardware and embedded systems (CHES) 1999, LNCS 1717, 2–12. Springer, Heidelberg
- Shamir A (n.d.) Method and apparatus for factoring large numbers with optoelectronic devices, patent 475920, filed 12/30/1999 and awarded 08/05/2003
- Shannon C (1938) A symbolic analysis of relay and switching circuits. Trans Am Inst Electr Eng 57:713–719
- Shapiro E (1999) A mechanical turing machine: blueprint for a biomolecular computer. In: Fifth international meeting on DNA-based computers at the Massachusetts Institute of Technology, Proc. DNA Based Computers V: Cambridge
- Sinha S, Ditto W (1999) Computing with distributed chaos. Phys Rev E Stat Phys Plasmas Fluids Relat Interdiscip Top 60(1):363–377
- Soloveichik D, Cook M, Winfree E, Bruck J (2008) Computation with finite stochastic chemical reaction networks. Nat Comput 7(4):615–633
- Soloveichik D, Seelig G, Winfree E (2010) DNA as a universal substrate for chemical kinetics. Proc Natl Acad Sci 107:5393–5398
- Soroka WA (1954) Analog methods in computation and simulation. McGraw-Hill, New York
- Squier R, Steiglitz K (1994) Programmable parallel arithmetic in cellular automata using a particle model. Complex Syst 8:311–323
- Svoboda A (1948) Computing mechanisms and linkages. McGraw-Hill, New York
- Swade D (1991) Charles Babbage and his calculating engines. Michigan State University Press, East Lansing
- Tate SR, Reif JH (1993) The complexity of N-body simulation. In: Proceedings of the 20th annual colloquium on automata, languages and programming (ICALP'93), Lund, pp 162–176
- Thomson W (later known as Lord Kelvin) (1878) Harmonic analyzer. Proc R Soc Lond 27:371–373
- Turck JAV (1921) Origin of modern calculating machines. The Western Society of Engineers, Chicago
- Turing A (1937) On computable numbers, with an application to the Entscheidungs problem. In: Proceedings of the London mathematical society, second series, vol 42, London, pp 230–265. Erratum in vol 43, pp 544–546
- Wang H (1963) Dominoes and the AEA case of the decision problem. In: Fox J (ed) Mathematical theory of automata. Polytechnic Press, Brooklyn, pp 23–55
- Winfree E, Yang X, Seeman NC (1996) Universal computation via self-assembly of DNA: some theory and experiments, DNA based computers II, volume 44 of DIMACS. American Mathematical Society, Providence, pp 191–213
- Winfree E, Liu F, Wenzler LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394:539–544
- Wolfram S (1984) Universality and complexity in cellular automata. Physica D10:1–35
- Woods D, Naughton TJ (2009) Optical computing. Appl Math Comput 215(4):1417–1430
- Xia Y, Whitesides GM (1998) Soft lithography. Ann Rev Mater Sci 28:153–184
- Yan H, LaBean TH, Feng L, Reif JH (2003a) Directed nucleation assembly of barcode patterned DNA lattices. Proc Natl Acad Sci U S A 100(14): 8103–8108
- Yan H, Feng L, LaBean TH, Reif J (2003b) DNA nanotubes, parallel molecular computations of pairwise exclusive-or (XOR) using DNA “string tile” self-assembly. J Am Chem Soc (JACS) 125(47): 14246–14247
- Yin P, Yan H, Daniel XG, Turberfield AJ, Reif JH (2004) A unidirectional DNA walker moving autonomously along a linear track. Angew Chem Int Ed 43(37):4906–4911



## Optical Computing

Thomas J. Naughton<sup>1</sup> and Damien Woods<sup>2</sup>

<sup>1</sup>Department of Computer Science, National University of Ireland, Maynooth County Kildare, Ireland

<sup>2</sup>Computer Science, California Institute of Technology, Pasadena, CA, USA

### Article Outline

Glossary

Definition of the Subject

Introduction

History

Selected Elements of Optical Computing Systems

Continuous Space Machine (CSM)

Example CSM Data Structures and Algorithms

C<sub>2</sub>-CSM

Optical Computing and Computational Complexity

Future Directions

Bibliography

### Glossary

**Coherent light** Light of a narrowband of wavelengths (temporally coherent) and a light beam whose phase is approximately constant over its cross-sectional area (spatial coherence). For example, coherent light can be produced by a laser.

**Continuous space machine (CSM)** A general optical model of computation that is defined in section “[Continuous Space Machine \(CSM\)](#).”

**Detector** A device for sensing light.

**Incoherent light** Light which is not spatially coherent and not temporally coherent. For

example, incoherent light is produced by a conventional light bulb.

**P, NP, PSPACE, NC** Complexity classes: these classes are respectively defined as the set of problems solvable on polynomial time deterministic Turing machines, polynomial time nondeterministic Turing machines, polynomial space Turing machines, and parallel computers that use polylogarithmic time and polynomial hardware (Papadimitriou 1995).

**Parallel computation thesis** This thesis states that parallel time corresponds, within a polynomial, to sequential space, for reasonable parallel and sequential machines (Chandra and Stockmeyer 1976; Goldschlager 1977; Karp and Ramachandran 1990; Parberry 1987; van Emde Boas 1990).

**Source** A device for generating light.

**Spatial light modulator (SLM)** A device that imposes some form of spatially varying modulation on a beam of light. An SLM may modulate the intensity, phase, or both of the light.

### Definition of the Subject

An optical computer is a physical information processing device that uses photons to transport data from one memory location to another and processes the data while it is in this form. In contrast, a conventional digital electronic computer uses electric fields (traveling along conductive paths) for this task. The optical data paths in an optical computer are effected by refraction (such as the action of a lens) or reflection (such as the action of a mirror). A principal advantage of an optical data path over an electrical data path is that optical data paths can intersect and even completely overlap without corrupting the data in either path. Optical computers make use of this property to efficiently transform the optically encoded data from one representation to another, for example, to shuffle or reverse the order of an array of parallel paths or to convolve the data in

several arrays of parallel paths. Other advantages of optical computers include inherent parallelism and the ability to encode a two-dimensional spatial function in the cross section of a single beam of light, higher bandwidths (in contrast to the free transmission of photons, electric fields generate noise in parallel conductors as they are pushed down their conductor), lower energy consumption (an argument deriving from the fact that optical computers in principle generate very little heat), easier circuit design, and lower latency in comparison to electrical transmission.

However, the property of noninterference of intersecting data paths means that it is not straightforward to effect a switch or branch instruction in optics since in a vacuum, the presence or absence of light in one data path cannot affect another path. In order to perform a conventional computation (e.g., solve a decision problem), optical computers invariably need to be equipped with an electronic interface, which would sense the presence or absence of light at some stage of the computation, and set the optical computer on a new course. Although this limitation has been addressed with varying levels of success through the development of nonlinear optical materials for all-optical switching and storage devices, it is generally accepted that if optical computers become mainstream, it will be through a symbiotic relationship with their extremely flexible digital electronic counterparts. Furthermore, currently, there is no convincing alternative to using digital electronics for optical computer data input devices (e.g., liquid-crystal display panels) and data output devices (e.g., digital cameras).

Optical computing is an inherently multidisciplinary subject whose study routinely involves a spectrum of expertise that threads optical physics, materials science, optical engineering, electrical engineering, computer architecture, computer programming, and computer theory. Applying ideas from theoretical computer science, such as analysis of algorithms and computational complexity, enables us to place optical computing in a framework where we can try to answer a number of important questions. For example, which problems are optical computers

suitable for solving? Also, how does the resource usage on optical computers compare with more standard (e.g., digital electronic) architectures? Furthermore, optical computing gives one an opportunity to apply computer theory on a completely new suite of machine models. In contrast to a number of other nature-inspired models of computation, optical computers have very real and immediate realization possibilities.

Traditionally, in optical information processing, a distinction was made between signal/image processing through optics and numerical processing through optics, with only the latter (and often only the digital version of the latter) being called optical computing (Feitelson 1988; Karim and Awwal 1992; McAulay 1991; Yu et al. 2001). However, it was always difficult to clearly delineate between the two, since it was largely a question of the interpretation the programmer attached to the output optical signals. The most important argument for referring to the latter only as optical computing had to do with the fact that the perceived limits (or at least, ambitions) of the former was simply for special-purpose signal/image processing devices while the ambitions for the latter was general-purpose computation Given recent results on the computational power of optical image processing architectures (Naughton 2000a; Woods 2005a; Woods and Naughton 2005), it is not the case that such architectures are limited to special-purpose tasks. Furthermore, as the field becomes increasingly multidisciplinary, and in particular as computer scientists play a more prominent role, it is necessary to bring the definition of optical computing in line with the broad definition of computing. In particular, this facilitates analysis from the theoretical computer science point of view. The distinction between analog optical computing and digital optical computing is similarly blurred given the prevalence of digital multiplication schemes effected through analog convolution (Karim and Awwal 1992). Our broad interpretation of the term optical computing has been espoused before (Caulfield et al. 1977).

## Introduction

The three most basic hardware components of an optical information processing system are a source, a modulator, and a detector. A source generates the light, a modulator multiplies the light by a (usually, spatially varying) function, and a detector senses the resulting light. The simplest example of a modulator encoding a spatially varying function is a transparency (a sheet of clear plastic or photographic film) with an opaque pattern handwritten or printed onto it. When placed in the path of an advancing wavefront, which we define simply as being a wide beam of light, the modulator encodes its pattern onto this wavefront. The common liquid-crystal display projector is a programmable example of the same principle. Keeping this kind of system in mind, we now highlight some attributes of optical information processing systems.

### Time Efficiency

Consider a light detector that converts incident light into an electrical current. Consider also an encoding scheme whereby the intensity in a beam of light represented a particular nonnegative integer. Further, assume there are no fluctuations in the light source output, that the encoding scheme is linear, and that the detector's response is linear. Then, the sum of two such nonnegative integers incident on the detector could be determined by measuring the detector's current. In fact, several nonnegative integers could be summed in this way, with a single measurement (see Fig. 1). (This concept is not unknown to designers of analog electrical ANNs. However, the important difference is that since the medium is free space, the practical fan-in limitations of Kirchhoff law of summation (Mead 1989) in analog electronics do not apply here.) Such an optical arrangement can find the sum of  $n$  nonnegative integers in  $O(1)$  addition steps. On a model of a sequential digital electronic computer, this would require  $n-1$  addition operations, and even a parallel digital electronic machine with  $n$  or more processors requires  $O(\log n)$  timesteps. Tasks that rely on scalar summation operations (such as matrix

multiplication) would benefit greatly from an optical implementation of the scalar sum operation. Similarly,  $O(1)$  multiplication and  $O(1)$  convolution operations can be realized optically. In section “[Optical Computing and Computational Complexity](#),” we formally describe the time efficiency of a broad class of optical computers. Very recently, an optics-based digital signal processing platform has been marketed that claims digital processing speeds of tera ( $10^{12}$ ) operations per second (Lenslet Labs [2001](#)).

### Efficiency in Interconnection Complexity

As optical pathways can cross in free space without measurable effect on the information in either channel, high interconnection densities are possible with optics (Caulfield [1989a](#); Caulfield et al. [1989](#)). Architectures with highly parallel many-to-many interconnections between parallel surfaces have already been proposed for common tasks such as sorting (Beyette et al. [1994](#); Desmulliez et al. [1996](#); Louri et al. [1995](#); Stirk and Athale [1988](#)). Currently, intra-chip, inter-chip, and inter-board connections are being investigated for manufacturing feasibility (Miller [2000](#)).

### Energy Efficiency

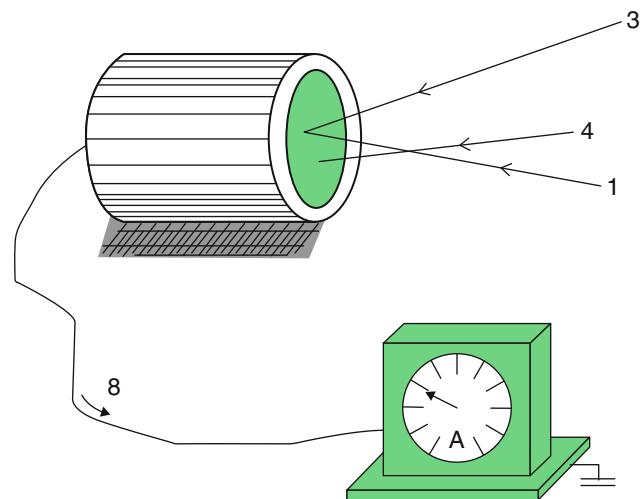
Electrical wires suffer from induced noise and heat, which increases dramatically whenever wires are made thinner or placed closer together or whenever the data throughput is increased (Miller [2000](#)). As a direct consequence of their resistance-free pathways and noise-reduced environments, optical systems have the potential to generate less waste heat and so consume less energy per computation step than electronic systems (Caulfield [1989b](#)). This has been demonstrated experimentally with general-purpose digital optical processors (Guilfoyle et al. [1998](#); Stone [1994](#); Stone et al. [1991](#)).

### Coherence

Mutually spatially coherent optical wavefronts (such as from a laser) interfere with each other just as waves in a water tank do. In the theory of physical optics, coherent wavefronts can be described by, and thus can represent, a complex-

### Optical Computing,

**Fig. 1** A light detector apparatus converts incident light into an electrical current. Multiple nonnegative integers, encoded in beams of light, can be summed in unit time



valued function (both positive and negative values in each of the real and imaginary axes). Using the language of this theory to interpret optical phenomena permits the definition of (at least) three important information processing constant-time operations: spatial modulation, Fourier transformation, and signal squaring (Goodman 1977). The ability to perform such operations has resulted in many constant-time optical implementations of standard convolution-based digital image processing tasks.

While incoherent light (such as from an ordinary light bulb) has some advantages in terms of tolerances in misalignment of optical components and less susceptibility to certain types of noise, coherent light is more general (in that its mathematics can be used to describe both coherent and incoherent wavefronts). Incoherent wavefronts can be modeled as being nonnegative everywhere and so only admit possibilities to directly represent nonnegative spatially varying and temporally varying functions.

### Optical Image Processing

It has long been appreciated that spatial optical signals are the most natural means of representing continuous tone 2D signals. There are many positive aspects to processing information using these (sometimes unwieldy and always inaccurate) physical signals instead of the more accurate

digital electronic representations of 2D signals. These include the ability to concurrently modify all parts of an image (spatial light modulation), the capability to substitute space computational complexity for time computational complexity when performing certain transformations (Caulfield 1990; Louri and Post 1992; Naughton 2000a; Reif and Tyagi 1997; Woods 2005a) (such as constant-time Fourier transformation with coherent light), the potential significant energy savings (Caulfield 1989b) (in both creating the signal and effecting the computation), and the ease with which analog signals can be digitized or resampled at an arbitrary frequency for subsequent digital electronic handling. The most common applications of optical image processing are pattern recognition and numerical matrix computations (see section “History” for elaboration).

Pattern recognition is one of the most commonly implemented signal, image, and information processing tasks. A significant number of the algorithms for these tasks involve a convolution operation, either as the principal operation (e.g., comparing two images on a pixel-by-pixel basis using correlation) or as part of necessary preprocessing (e.g., edge detection prior to applying a Hough transform (Hough 1962)). The inherent parallel nature of optical systems can be used to facilitate low time and space complexity implementations of the convolution operation,

either by multiplication in the Fourier domain or by systolic action (Caulfield et al. 1981).

### Overview of the Chapter

In this chapter, we focus on optical image processing as it is an optical computing paradigm that makes full use of the degrees of freedom afforded by optics. Other optical computing architectures that seek to emulate perfectly in step-by-step fashion the operations of digital electronic architectures (e.g., architectures built upon all-optical flip-flops (Clavero et al. 2005) and all-optical network routers (Dorren et al. 2003)) occupy an important place in the taxonomy of optical computing. However, in terms of computational complexity, the analyses of these digital optical computers are in many respects identical to that of the digital electronic counterparts they emulate.

In section “History,” we give a brief overview of the history of optical computing, commonly referred to as optical information processing. Section “History” also includes an overview of existing optical models of computation. This is followed in section “Selected Elements of Optical Computing Systems” by a summary of the most important elements of optical computing that could be used to define the functionality of an optical model of computation. In section “Continuous Space Machine (CSM),” we take a detailed look at a particular model of optical computing (the CSM) that encompasses most of the functionality that coherent optical information processing has to offer. We begin by defining the CSM and a total of seven complexity measures that are inspired by real-world (optical) resources. We go on to discuss how the CSM’s operations could be carried out physically. Section “Example CSM Data Structures and Algorithms” contains some example data structures and algorithms for the CSM. In section “ $\mathcal{C}_2$ -CSM,” we motivate and introduce an important restriction of the model called the  $\mathcal{C}_2$ -CSM, and in section “Optical Computing and Computational Complexity,” we describe a number of  $\mathcal{C}_2$ -CSM computational complexity results

and their implications. We conclude with some future directions in section “Future Directions.”

### History

It could be argued that the field of optical information processing began in earnest with the realization that spatially coherent laser light could be used to conveniently Fourier transform an image, allow one to modify the complex-valued spatial frequency components, and then inverse Fourier transform back to the spatial domain.

This concept is called spatial filtering (Cutrona et al. 1960, 1966; Leith 1977; O’Neill 1956; Turin 1960; VanderLugt 1964, 1974), it is a generalization that encompasses convolution and correlation operations, and it could be performed over two-dimensional (2D) images in constant time while limited in speed only by the refresh rates of the input and output devices. It first found application in the 1950s for parallel processing and analysis of the huge amounts of radar data produced at the time. The initial special-purpose spatial filtering systems performed optical Fourier transforms, performed image processing (e.g., noise reduction and edge enhancement), and recognized patterns through correlation. The fundamentals of optical spatial filtering were formulated in that decade and built upon previous work on optimum linear filtering in communication theory. Achieving the full potential of optical spatial filtering theory requires filters that are complex valued, and a technique to obtain such filters was first proposed by VanderLugt (1964, 1992). The technique allows one to physically encode a complex-valued image on an amplitude-modulating SLM such as an LCD panel.

Research continued into this form of image-based computation. Many important image processing tasks were demonstrated at that time, from character recognition (Armitage and Lohmann 1965), to real-time tracking of moving objects (Gara 1979; Upatnieks 1983), to telescope/microscope image deblurring (Stroke et al. 1974). Two important strands of this

research at the time were the development of sophisticated pattern recognition algorithms and numerical computation using values encoded in the complex amplitude or intensity of the light.

### Optical Pattern Recognition

In pattern recognition (Arsenault et al. 1984; Casasent 1984; Casasent and Psaltis 1976a, b; Caulfield and Haimes 1980; Chang et al. 2000; Hsu and Arsenault 1982; Javidi and Wang 1995), effort focused on achieving systems invariant to scaling, rotation, out-of-plane rotation, deformation, and signal-dependent noise while retaining the existing invariance to translating, adding noise to, and obscuring parts of the input. Effort also went into injecting nonlinearities into these inherently linear systems to achieve wider functionality (Javidi 1989, 1990). Improvements were made to the fundamental limitations of the VanderLugt filter, most notably the joint transform correlator architecture (Weaver and Goodman 1966).

Optical correlators that use incoherent sources of illumination (both spatially and temporally) rather than lasers are also possible (Casasent and House 1994a, b; Esteve-Taboada et al. 2000; Pe'er et al. 1999; Zhai et al. 1999). The simplest incoherent correlator would have the same basic architecture as that used for matched filtering. While coherent systems in principle are more capable than incoherent systems (principally because the former naturally represents complex functions while the latter naturally represents real functions), incoherent systems require less precise positioning when it comes to system construction and are less susceptible to noise.

A common example of an optical correlator's use in practical systems involved it being used as a front end to a generalized hybrid object recognition system. The optical processing component would quickly and efficiently identify regions of interest in a cluttered scene and pass these on to the slower but more accurate digital electronic components for false-alarm reduction, feature extraction, and classification. Today, the matched filter and the joint transform correlator are the two most widely used optical pattern recognition

techniques (Feng et al. 1995; Horner 1987; Lu et al. 1990; Yu 1996).

Trade-offs between space and time were proposed and demonstrated. These included time integrating correlators (VanderLugt 1992) (in contrast to the space integrating correlators mentioned thus far) and systolic architectures (Casasent et al. 1983; Caulfield et al. 1981; Ghosh et al. 1985) where, for example, the propagation of an amplitude-modulated pressure wave through an acousto-optic device naturally effects the required correlation lags (Rhodes 1981). In addition to pattern recognition, a common application for these classes of architectures was numerical calculation.

### Analog Optical Numerical Computation

An important strand of image-based optical computation involved numerical calculations: analog computation as well as multilevel discrete computation. Matrix-vector and matrix-matrix multiplication systems were proposed and demonstrated (Arsenault and Sheng 1992; Feitelson 1988; Horner 1987; Karim and Awwal 1992; Lee 1995; McAulay 1991; VanderLugt 1992). The capability to expand a beam of light and to focus many beams of light to a common point directly corresponded to high fan-out and fan-in capabilities, respectively. The limitations of encoding a number simply as an intensity value (finite dynamic range and finite intensity resolution in the modulators and detectors) could be overcome by representing the numbers in some base. Significant effort went into dealing with carry operations so that in additions, subtractions, and multiplications, each digit could be processed in parallel. Algorithms based on convolution to multiply numbers in this representation were demonstrated (Karim and Awwal 1992), with a single post-processing step to combine the sub-calculations and deal with the carry operations. Residue arithmetic was demonstrated as a viable alternative in which carry operations did not arise at all, and for which a matrix-vector multiplier was proposed (Huang et al. 1979), but of course conversion to and from residue format is necessary.

An application that benefited greatly from the tightly coupled parallelism afforded by optics was the solving of sets of simultaneous equations and matrix inversion (Abushagur and Caulfield 1987; Caulfield and Abushagur 1986). An application that, further, was tolerant to the inherent inaccuracies and noise of analog optics was optical neural networks (Caulfield et al. 1989; Farhat and Psaltis 1984; Hsu et al. 1990; Psaltis and Farhat 1985) including online neural learning in the presence of noise (Naughton et al. 1999).

### Digital Optical Computing

The next major advances came in the form of optical equivalents of digital computers (Huang 1984). The flexibility of digital systems over analog systems in general was a major factor behind the interest in this form of optical computation (Sawchuk and Strand 1984). Specific drawbacks of the analog computing paradigm in optics that this new paradigm addressed included no perceived ability to perform general-purpose computation, accumulation of noise from one computation step to another, and systematic errors introduced by imperfect analog components. The aim was to design digital optical computers that followed the same principles as conventional electronic processors but which could perform many binary operations in parallel. These systems were designed from logic gates using nonlinear optical elements: semitransparent materials whose transmitted intensity has a nonlinear dependence on the input intensity. Almost always, the coherence of the light was not used in the computation. All-optical bistable devices acting as flip-flops were demonstrated. The field drew on many decades of research into fast on-off optical switching speeds which was heralding an explosion in optical fiber communications. The difficulties that the digital optical paradigm experienced included how to fully exploit the theoretical parallelism of optics within an optical logic framework, how to efficiently manufacture very large systems of cascaded nonlinear optical elements (for which integrated optics holds promise (Goldberg and Lee 1979)), and the more fundamental mathematical problem of how to parallelize arbitrary

algorithms in the first place to exploit the parallelism afforded by digital optics.

Digital optical computing was also proposed as an application of architectures designed originally for image-based processing, for example, logic effected through symbolic substitution (Brenner et al. 1986, 1990). At the confluence of computing and communication, optical techniques were proposed for the routing of signals in long-haul networks (Feitelson 1988; Yu et al. 2001). This is a promising application given that most long-haul communications already use light in optical fibers, and the conversion from optical to electronic in order to switch, and then back to optical to retransmit, can be costly. Initial implementations followed the concept of an optoelectronic crossbar switch with  $n$  inputs and  $n$  outputs (Sawchuk and Strand 1984), while latterly more effort is now going into all-optical packet switching in a single channel configuration (Clavero et al. 2005; Dorren et al. 2003; Geldenhuys et al. 2004).

### Optical Models of Computation

As already discussed, optical computers were designed and built to emulate conventional microprocessors (digital optical computing) and for image processing over continuous wavefronts (analog optical computing and pattern recognition). Here we are interested in the latter class: optical computers that store data as images. Numerous physical implementations exist and example applications include fast pattern recognition and matrix-vector algebra (Goodman 2005; VanderLugt 1992). There have been much resources devoted to designs, implementations, and algorithms for such optical information processing architectures (e.g., see Arsenault and Sheng 1992; Caulfield 1990; Durand-Lose 2006; Feitelson 1988; Goodman 2005; Lee 1995; Louri and Post 1992; McAulay 1991; Naughton et al. 1999; Reif and Tyagi 1997; VanderLugt 1992; Yu et al. 2001 and their references).

However, the computational complexity theory of optical computers (i.e., finding lower and upper bounds on computational power in terms of known complexity classes) had received relatively little attention when compared with other

nature-inspired computing paradigms. Some authors have even complained about the lack of suitable models (Feitelson 1988; Louri and Post 1992). Many other areas of natural computing (e.g., Adleman 1994; Grover 1996; Head 1987; Lipton 1995; Moore 1991, 1997; Păun 2002; Shor 1994; Yokomori 2002) have not suffered from this problem. Even so, we discuss some optical computation research that is close to the goals of the theoretical computer scientist.

Reif and Tyagi (1997) study two optically inspired models. The first model is a 3D VLSI model augmented with a 2D discrete Fourier transform (DFT) primitive and parallel optical interconnections. The second model is a DFT circuit with operations (multiplication, addition, comparison of two inputs, DFT) that compute over an ordered ring. Parallel time complexity is defined for both models in the obvious way. For the first model, volume complexity is defined as the volume of the smallest convex box enclosing an instance of the model. For the DFT circuit, size is defined as the number of edges plus gates. Constant time, polynomial size/volume, and algorithms for a number of problems are reported including 1D DFT, matrix multiplication, sorting, and string matching (Reif and Tyagi 1997).

Feitelson (1988) gives a call to theoretical computer scientists to apply their knowledge and techniques to optical computing. He then goes on to generalize the concurrent read, concurrent write parallel random access machine, by augmenting it with two optically inspired operations. The first is the ability to write the same piece of data to many global memory locations at once. Secondly, if many values are concurrently written to a single memory location, then a summation of those values is computed in a single timestep. Essentially, Feitelson is using “unbounded fan-in with summation” and “unbounded fan-out.” His architecture mixes a well-known discrete model with some optical capabilities.

A symbolic substitution model of computation has been proposed by Huang and Brenner, and a proof sketched of its universality (Brenner et al. 1986). This model of digital computation operates over discrete binary images and derives its efficiency by performing logical operations on

each pixel in the image in parallel. It has the functionality to copy, invert, and shift laterally individual images and OR and AND pairs of images. Suggested techniques for its optical implementation are outlined.

In computer science, there are two famous classes of problems called P and NP (Papadimitriou 1995); P contains those problems that are solvable in polynomial time on a standard sequential computer, while NP is the class of problems that are solvable in polynomial time on a nondeterministic computer. NP contains P, and it is widely conjectured that they are not equal. A direct consequence of this conjecture is that there are (NP-hard) problems for which we strongly believe there is no polynomial time algorithm on a standard sequential computer.

It is known that it is possible to solve any NP (and even any PSPACE) problem in polynomial time on optical computers, albeit with exponential use of some other, space-like, resources (Woods 2005a, 2006; Woods and Gibson 2005b). In section “ *$\mathcal{C}_2$ -CSM and Parallel Complexity Theory*,” we describe how parallel optical algorithms can solve such problems.

Along with these rather general results, there are a number of specific examples of algorithms with related resource usage for NP-hard problems. Shaked et al. (2006, 2007) design an optical system for solving the NP-hard traveling salesman problem in polynomial time. Basically, they use an optical matrix-vector multiplier to multiply the (exponentially large) matrix of tours by the vector of intercity weights. They give both optical experiments and simulations. Dolev and Fitoussi (2007) give optical algorithms that make use of (exponentially large) masks to solve a number of NP-hard problems. Oltean (2006) and Haist and Osten (2007) give polynomial time algorithms for Hamiltonian path and traveling salesman problem, respectively, via light traveling through optical cables. As is to be expected, both suffer from exponential (space-like) resource use. Nature-inspired systems that apparently solve NP-hard problems in polynomial time, while using an exponential amount of some other resource(s), have been around for many years. So the existence of massively

parallel optical systems for NP-hard problems should not really surprise the reader. Nevertheless, it is interesting to know the computational abilities, limitations, and resource trade-offs of such optical architectures, as well as to find particular (tractable or intractable) problems which are particularly suited to optical algorithms.

Reif et al. (1990) examined the computational complexity of ray tracing problems. In such problems, we are concerned about the geometry of the optical system where diffraction is ignored, and we wish to predict the position of light rays after passing through some system of mirrors and lenses. They gave undecidability and PSPACE hardness results, which gives an indication of the power of these systems as computational models.

## Selected Elements of Optical Computing Systems

If one is designing an optical model of computation, one will incorporate the functionality of a subset of the following selected elements (devices and functionality) of optical computing systems.

### Sources

Lasers are a common source of illumination because at some levels, they are mathematically simpler to understand, but incoherent sources such as light-emitting diodes are also used frequently for increased tolerance to noise and when nonnegative functions are sufficient for the computation. Usually, the source is monochromatic to avoid the problem of color dispersion as the light passes through refracting optical components, unless this dispersion is itself the basis for the computation.

### Spatial Light Modulators

It is possible to encode a spatial function (a 2D image) in an optical wavefront. A page of text when illuminated with sunlight, for example, does this job perfectly. This would be called an amplitude-modulating reflective SLM. Modulators can also act on phase and polarization and

can be transmissive rather than reflective. They include photographic film and electro-optic, magneto-optic, and acousto-optic devices (Arsenault and Sheng 1992; Feitelson 1988; Goodman 1977, 2005; Karim and Awwal 1992). One class of note are the optically addressed SLMs, in which, typically, a 2D light pattern falling on a photosensitive layer on one side of the SLM spatially varies (with an identical pattern) the reflective properties of the other side of the SLM. A beam splitter then allows one to read out this spatially varying reflectance pattern. The liquid-crystal light valve (Efron et al. 1985; Grinberg et al. 1975; Jacobson et al. 1972; VanderLugt 1992; Wang and Saffman 1999) is one instance of this class. Other classes of SLMs such as liquid-crystal display panels and acousto-optic modulators allow one to dynamically alter the pattern using electronics. It is possible for a single device (such as an electronically programmed array of individual sources) to act as both source and modulator.

### Detectors and Nature's Square Law

Optical signals can be regarded as having both an amplitude and phase. However, detectors will measure only the square of the amplitude of the signal (referred to as its intensity). This phenomenon is known as nature's detector square law and applies to detectors from photographic film to digital cameras to the human eye. Detectors that obey this law are referred to as square-law detectors. This law is evident in many physical theories of light. In quantum theory, the measurement of a complex probability function is formalized as a projection onto the set of real numbers through a squaring operation. Square-law detectors need to be augmented with an interferometric or holographic arrangement to measure both amplitude and phase rather than intensity (Caulfield 1979; Gabor 1948) or need to be used for multiple captures in different domains to heuristically infer the phase.

Since it squares the absolute value of a complex function, this square law can be used for some useful computation (e.g., in the joint transform correlator (Weaver and Goodman 1966)). Detectors most commonly used include

high-range point (single-pixel) detectors such as photodiodes, highly sensitive photon detectors such as photomultiplier tubes, and 1D and 2D array detectors such as CCD or CMOS digital cameras. Intensity values outside the range of a detector (outside the lowest and highest intensities that the detector can record) are thresholded accordingly. The integration time of some detectors can be adjusted to sum all of the light intensity falling on them over a period of time. Other detectors can have quite large light sensitive areas and can sum all of the light intensity falling in a region of space.

### Lenses

Lenses can be used to effect high fan-in and fan-out interconnections, to rescale images linearly in either one or two dimensions, and to take Fourier transforms. In fact, a coherent optical wavefront naturally evolves into its Fresnel transform, and subsequently into its Fourier transform at infinity, and the lens simply images those frequency components at a finite fixed distance.

VanderLugt (1992) has derived an expression for the coherent optical Fourier transform, given here in 1D for convenience, using the Fresnel transform of a complex-valued function  $f(x)$  positioned in the front focal plane of a convex lens (plane P1 in Fig. 2), which is illuminated from the back by a plane wave of constant amplitude and phase. In terms of the physical coordinate  $\xi$ , the signal at the back focal plane P2 can be written as

$$F(\xi) = \sqrt{\frac{i}{\lambda L}} \int_{-\infty}^{\infty} f(x) \exp(i2\pi x \xi / \lambda L) dx, \quad (1)$$

where  $\lambda$  is the wavelength of the illumination and  $L$  is the focal length of the lens. Rewriting so it is a function of the spatial frequency variable  $\alpha$  ( $\alpha$  is a measurement of radians per unit distance) gives the common equation

$$F(\alpha) = \sqrt{\frac{i}{\lambda L}} \int_{-\infty}^{\infty} f(x) \exp(i2\pi \alpha x) dx, \quad (2)$$

where  $\xi = \lambda L \alpha$  and where we ignore the architecture-specific scaling constant. The Fourier transform in optics can be formed under a wide variety of conditions and not just with a plane wave and not just in the focal plane of the lens (VanderLugt 1992). This formalism adopts the paraxial approximation that the distance between planes P1 and P2 is very much greater than the extent of the information in P1, thus avoiding the need for curved opposing surfaces in planes P1 and P3. When a Fourier transform is detected directly, only its square, called the power spectrum, is recorded. As mentioned, holography (Caulfield 1979; Gabor 1948) overcomes this.

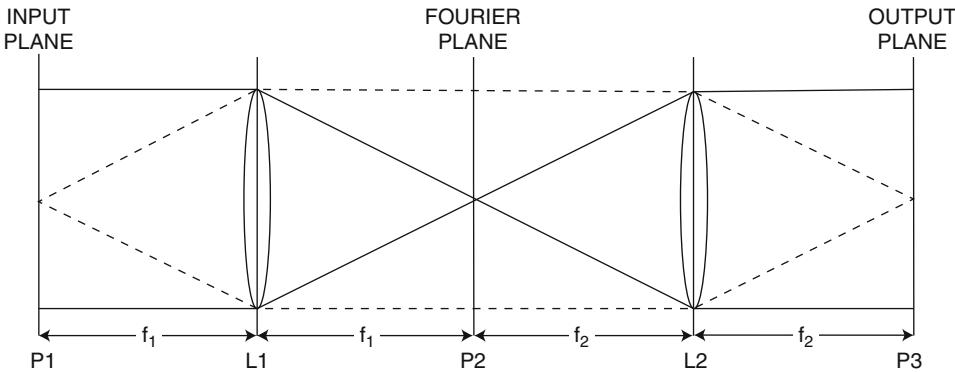
### Interference and Complex Addition

Although photons, being bosons, do not interact with each other, coherent wavefronts can be made to interact at a detector. The addition (called superposition) of complex-valued wavefronts at a measurement is termed interference. Optically, interference is the same phenomenon as diffraction and is responsible for the formation of optical Fourier transforms. The linearity property is a useful property when analyzing coherent optical phenomena. If several images are coplanar, then the optical field at their common Fourier plane is the superposition of their frequency spectra. This superposition, or interference, of complex-valued signals can be regarded as a pointwise addition of the complex amplitudes of the images.

Incoherent wavefronts are nonnegative everywhere. The addition of several incoherent wavefronts is linear in the intensity of each of those wavefronts. The addition of coherent wavefronts is linear in their complex amplitudes.

### Image Copying and Combining

Images can be copied using optically addressed SLMs or by dividing the optical energy along two paths using a beam splitter (Chiou 1999; VanderLugt 1992; Weaver and Goodman 1966). Beam splitters can also be used to combine several images to make them coplanar.



**Optical Computing, Fig. 2** Optical spatial frequency filtering using two convex lenses with a plane wave illuminating the input from the left. Lenses L1 and L2 have focal lengths of  $f_1$  and  $f_2$ , respectively. If we assume that the physical dimensions of the lenses allow all diffracted

orders to pass, the output is a transposed (and rescaled if  $f_1 \neq f_2$ ) but otherwise identical version of the input. Any modification to the light field in the Fourier plane results in a spatial frequency filtered output

### Multiplication of Images

When a signal's Fourier spectrum  $F(\alpha, \beta)$  is coplanar with a transparency that encodes a second Fourier spectrum  $H(\alpha, \beta)$ , and if their centers are coincident, the complex signal in the plane immediately behind the transparency can be described as the pointwise multiplication of the two spectra

$$G(\alpha, \beta) = F(\alpha, \beta)H(\alpha, \beta). \quad (3)$$

The signal  $G(\alpha, \beta)$ , which is also a frequency spectrum, could be inverse Fourier transformed to reveal a suitably correlated, convolved, or spatial frequency filtered original signal. A significant proportion of analog optics' role in the area of computation through filtering concerns convolution and those signal processing operations derived from it. Convolution filters are used extensively in the digital signal processing world to perform such tasks as deblurring, restoration, enhancement, and recognition (Karim and Awwal 1992). The possibility of performing constant-time convolution operations using coherent light is a promising concept.

To an approximation, optical systems can be regarded as both linear and shift invariant. This is the basis for their convolution capabilities. Referring to Eq. 3, one can see that  $H(\alpha, \beta)$  acts as a

spatial frequency filter, altering the frequency content of the signal  $F$ . It could be used as a simple band-pass filter, damping the high-frequency components (noise suppression) or low-frequency components (edge enhancement), or used to modulate the frequency spectrum with a more sophisticated spatial function. Mathematically, convolution with a mask  $A$  is equivalent to a frequency-domain multiplication with the Fourier transform of  $A$ .

Phase shifts can be introduced to a coherent wavefront by adding by a constant phase value. These could be constant shifts to effect numerical subtraction (Feitelson 1988) or time varying using a piezoelectric transducer mirror (Yamaguchi and Zhang 1997).

### Other Elements of Optical Computation

A mirror changes the direction of the wavefront and simultaneously reflects it along some axis. A phase conjugate mirror (Chiou 1999) returns an image along the same path at which it approached the mirror.

In-plane rotation of an image by 180° can be accomplished using the apparatus in Fig. 2, a single lens, or even a pinhole camera. An out-of-plane tilt can be accomplished using a prism. In-plane flipping of an image (mirror image) can be accomplished using a prism or using a beam splitter and some mirrors. Arbitrary

in-plane rotation (with some tilting and translation, if required) can be achieved by combining several flip operations using a Dove prism or Pechan prism (Paek et al. 1997; Sullivan 1972) or by combining several shearing operations (Lohmann 1993). Image rescaling can be accomplished using a combination of lenses (rescaling both dimensions identically) or using cylindrical lenses or an anamorphic prism (rescaling in one dimension only).

A prism or diffraction grating can be used to separate by wavelength the components of a multiwavelength optical channel. For optical fiber communications applications, more practical (robust, economical, and scalable) alternatives exist to achieve the same effect (Yu et al. 2001).

Polarization is an important property of wavefronts, in particular in coherent optical computing, and is the basis for how liquid-crystal displays work. At each point, an optical wavefront has an independent polarization value dependent on the angle, in the range  $[0, 2\pi)$ , of its electrical field. This can be independent of its successor (in the case of randomly polarized wavefronts), or dependent (as in the case of linear polarization), or dependent and time varying (as in the case of circular or elliptical polarization). Mathematically, a polarization state, and the transition from one polarization state to another, can be described using the Mueller calculus or the Jones calculus.

Photons can also be used for quantum computation, and quantum computers using linear optical elements (such as mirrors, polarizers, beam splitters, and phase shifters) have been proposed and demonstrated (Cerf et al. 1998; Knill et al. 2001; Pittman et al. 2003).

## Continuous Space Machine (CSM)

For the remainder of this chapter, we focus on an optical model of computation called the CSM. The model was originally proposed by Naughton (2000a, b). The CSM is inspired by analog Fourier optical computing architectures, specifically pattern recognition and matrix algebra processors (Goodman 2005; Naughton et al. 1999). For example, these architectures have the ability to do unit

time Fourier transformation using coherent (laser) light and lenses. The CSM computes in discrete timesteps over a number of two-dimensional images of fixed size and arbitrary spatial resolution. The data and program are stored as images. The (constant-time) operations on images include Fourier transformation, multiplication, addition, thresholding, copying, and scaling. The model is designed to capture much of the important features of optical computers while at the same time be amenable to analysis from a computer theory point of view. Toward these goals, we give an overview of how the model relates to optics as well as gives a number of computational complexity results for the model.

Section “[CSM Definition](#)” begins by defining the model. We analyze the model in terms of seven complexity measures inspired by real-world resources; these are described in section “[Complexity Measures](#). [”](#) In section “[Optical Realization](#), [”](#) we discuss possible optical implementations for the model. We then go on to give an example of algorithms and data structures in section “[Example CSM Data Structures and Algorithms](#). [”](#) The CSM definition is rather general, and so in section “[CSM Definition](#), [”](#) we define a more restricted model called the  $\mathcal{C}_2$ -CSM.

Compared to the CSM, the  $\mathcal{C}_2$ -CSM is somewhat closer to optical computing as it happens in the laboratory. Finally, in section “[Optical Computing and Computational Complexity](#), [”](#) we show the power and limitations of optical computing, as embodied by the  $\mathcal{C}_2$ -CSM, in terms of computational complexity theory. Optical information processing is a highly parallel form of computing, and we make this intuition more concrete by relating the  $\mathcal{C}_2$ -CSM to parallel complexity theory by characterizing the parallel complexity class NC. For example, this shows the kind of worst-case resource usage one would expect when applying CSM algorithms to problems that are known to be suited to parallel solutions.

### CSM Definition

We begin this section by describing the CSM model in its most general setting; this brief overview is not intended to be complete and more details are to be found in Woods (2005a).

A complex-valued image (or simply, image) is a function  $f : [0, 1] \times [0, 1] \rightarrow \mathbb{C}$ , where  $[0, 1]$  is the half-open real unit interval. We let  $\mathcal{I}$  denote the set of complex-valued images. Let  $\mathbb{N}^+ = \{1, 2, 3, \dots\}$ ,  $\mathbb{N} = \mathbb{N}^+ \cup \{0\}$ , and for a given CSM  $M$ , let  $\mathcal{N}$  be a countable set of images that encode  $M$ 's addresses. An address is an element of  $\mathbb{N} \times \mathbb{N}$ . Additionally, for a given  $M$ , there is an *address encoding function*  $\mathfrak{E} : \mathbb{N} \rightarrow \mathcal{N}$  such that  $\mathfrak{E}$  is Turing machine decidable, under some *reasonable* representation of images as words.

#### Definition 1 (CSM)

A CSM is a quintuple  $M = (\mathfrak{E}, L, I, P, O)$ , where

- $\mathfrak{E} : \mathbb{N} \rightarrow \mathcal{N}$  is the address encoding function
- $L = ((s_\xi, s_\eta), (a_\xi, a_\eta), (b_\xi, b_\eta))$  are the addresses: *sta*, *a*, and *b*, where  $a \neq b$
- *I* and *O* are finite sets of input and output addresses, respectively
- $P = \left\{ \left( \zeta_1, p_{1_\zeta}, p_{1_\eta} \right), \dots, \left( \zeta_r, p_r, p_{r_\eta} \right) \right\}$  are the  $r$  programming symbols  $\zeta_j$  and their addresses  $(p_{j_\xi}, p_{j_\eta})$

where  $\zeta_j \in (\{h, v, *, ., +, \rho, st, ld, br, hlt\} \cup N) \subset I$ .

Each address is an element from  $\{0, \dots, \Xi - 1\} \times \{0, \dots, \mathcal{Y} - 1\}$ , where  $\Xi, \mathcal{Y} \in \mathbb{N}^+$ .

Addresses whose contents are not specified by  $P$  in a CSM definition are assumed to contain the constant image  $f(x, y) = 0$ . We interpret this definition to mean that  $M$  is (initially) defined on a grid of images bounded by the constants  $\Xi$  and  $y$ , in the horizontal and vertical directions, respectively. The grid of images may grow in size as the computation progresses.

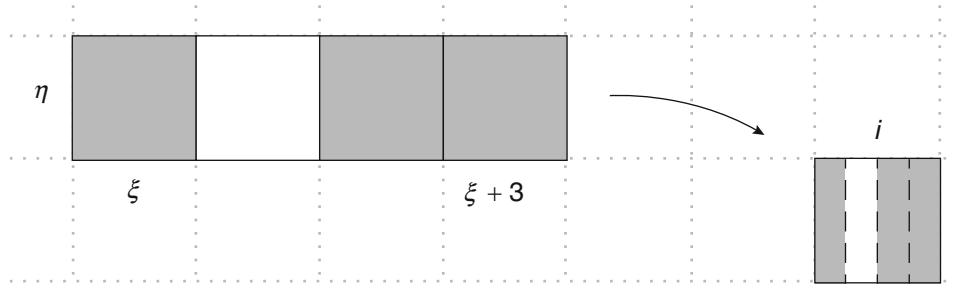
In our grid notation, the first and second elements of an address tuple refer to the horizontal and vertical axes of the grid, respectively, and image  $(0, 0)$  is located at the lower left-hand corner of the grid. The images have the same orientation as the grid. For example, the value  $f(0, 0)$  is located at the lower left-hand corner of the image  $f$ .

In Definition 1, the tuple  $P$  specifies the CSM program using programming symbol images  $\zeta_j$  that are from the (low-level) CSM programming language (Woods 2005a; Woods and Naughton 2005). We refrain from giving a description of this programming language and instead describe a less cumbersome high-level language (Woods 2005a). Figure 3 gives the basic instructions of this high-level language. The copy instruction is illustrated in Fig. 4. There are also **if/else** and **while** control flow instructions with conditional equality tests of the form  $(f_\psi == f_\phi)$  where  $f_\psi$  and  $f_\phi$  are *binary symbol images* (see Fig. 5a, b).

Address *sta* is the start location for the program so the programmer should write the first program instruction at *sta*. Addresses *a* and *b* define special images that are frequently used by some program instructions. The function  $\mathfrak{E}$  is specified by the programmer and is used to map addresses to image pairs. This enables the

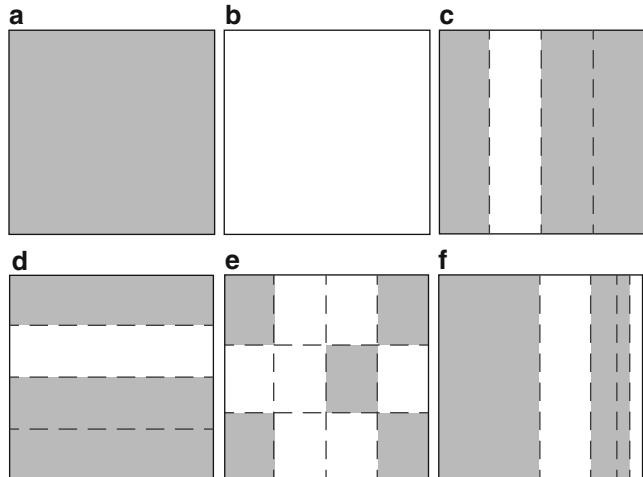
$h(i_1; i_2)$	: replace image at $i_2$ with horizontal 1D Fourier transform of $i_1$ .
$v(i_1; i_2)$	: replace image at $i_2$ with vertical 1D Fourier transform of image at $i_1$ .
$*(i_1; i_2)$	: replace image at $i_2$ with the complex conjugate of image at $i_1$ .
$*i_1, i_2; i_3$	: pointwise multiply the two images at $i_1$ and $i_2$ . Store result at $i_3$ .
$+i_1, i_2; i_3$	: pointwise addition of the two images at $i_1$ and $i_2$ . Store result at $i_3$ .
$\rho(i_1, z_l, z_u; i_2)$	: filter the image at $i_1$ by amplitude using $z_l$ and $z_u$ as lower and upper amplitude threshold images, respectively. Place result at $i_2$ .
$[\xi'_1, \xi'_2, \eta'_1, \eta'_2] \leftarrow [\xi_1, \xi_2, \eta_1, \eta_2]$	: copy the rectangle of images whose bottom left-hand address is $(\xi_1, \eta_1)$ and whose top right-hand address is $(\xi_2, \eta_2)$ to the rectangle of images whose bottom left-hand address is $(\xi'_1, \eta'_1)$ and whose top right-hand address is $(\xi'_2, \eta'_2)$ . See illustration in Fig. 4.

**Optical Computing, Fig. 3** CSM high-level programming language instructions. In these instructions,  $i, z_l, z_u \in \mathbb{N} \times \mathbb{N}$  are image addresses and  $\xi, \eta \in \mathbb{N}$ . The control flow instructions are described in the main text



**Optical Computing, Fig. 4** Illustration of the instruction  $i \leftarrow [\xi, \xi + 3, \eta, \eta]$  that copies four images to a single image that is denoted  $i$

**Optical Computing,**  
**Fig. 5** Representing binary data. The *shaded areas* denote value 1 and the *white areas* denote value 0. (a) Binary symbol image representation of 1 and (b) of 0, (c) list (or row) image representation of the word 1011, (d) column image representation of 1011, e $3 \times 4$  matrix image, and (f) binary stack image representation of 1101. Dashed lines are for illustration purposes only



programmer to choose her own address encoding scheme. We typically don't want  $\mathfrak{E}$  to hide complicated behavior; thus, the computational power of this function should be somewhat restricted. For example, we put such a restriction on  $\mathfrak{E}$  in Definition 7. At any given timestep, a configuration is defined in a straightforward way as a tuple  $\langle c, e \rangle$  where  $c$  is an address called the control and  $e$  represents the grid contents.

### Complexity Measures

In this section, we define a number of CSM complexity measures. As is standard, all resource-bounding functions map from  $\mathbb{N}$  into  $\mathbb{N}$  and are assumed to have the usual properties (Balcazar et al. 1988). We begin by defining CSM TIME complexity in a manner that is standard among parallel models of computation (Definition 2).

### Definition 2

The *TIME complexity* of a CSM  $M$  is the number of configurations in the computation sequence of  $M$ , beginning with the initial configuration and ending with the first final configuration.

The first of our six space-like resources is called GRID (Definition 3).

### Definition 3

The *GRID complexity* of a CSM  $M$  is the minimum number of images, arranged in a rectangular grid, for  $M$  to compute correctly on all inputs.

Let  $S : \mathcal{I} \times (\mathbb{N} \times \mathbb{N}) \rightarrow \mathcal{I}$ , where  $S(f(x, y), (\Phi, \Psi))$  is a raster image, with  $\Phi\Psi$  constant-valued pixels arranged in  $\Phi$  columns and  $\Psi$  rows, that approximates  $f(x, y)$ . If we choose a reasonable and realistic  $S$ , then the details of  $S$  are not important (Definition 4).

#### Definition 4

The *SPATIALRES* complexity of a CSM  $M$  is the minimum  $\Phi\Psi$  such that if each image  $f(x, y)$  in the computation of  $M$  is replaced with  $S(f(x, y), (\Phi, \Psi))$ , then  $M$  computes correctly on all inputs.

One can think of SPATIALRES as a measure of the number of pixels needed during a computation. In optical image processing terms, and given the fixed size of our images, SPATIALRES corresponds to the space-bandwidth product of a detector or SLM (Definition 5).

#### Definition 5

The *DYRANGE* complexity of a CSM  $M$  is the ceiling of the maximum of all the amplitude values stored in all of  $M$ 's images during  $M$ 's computation.

In optical processing terms, DYRANGE corresponds to the dynamic range of a signal.

We also use complexity measures called AMPLRES, PHASERES, and FREQ (Woods 2005a; Woods and Naughton 2005). Roughly speaking, the AMPLRES of a CSM  $M$  is the number of discrete, evenly spaced, amplitude values per unit amplitude of the complex numbers in  $M$ 's images, and so AMPLRES corresponds to the amplitude quantization of a signal. The PHASERES of  $M$  is the total number (per  $2\pi$ ) of discrete evenly spaced phase values in  $M$ 's images, and so PHASERES corresponds to the phase quantization of a signal. Finally, the FREQ complexity of a CSM  $M$  is the minimum optical frequency necessary for  $M$  to compute correctly; this concept is explained further in Woods and Naughton (2005).

Often we wish to make analogies between space on some well-known model and CSM “space-like” resources. Thus, we define the following convenient term (Definition 6).

#### Definition 6

The *SPACE* complexity of a CSM  $M$  is the product of all of  $M$ 's complexity measures except TIME.

### Optical Realization

In this section, we outline how some of the elementary operations of the CSM could be carried out physically.

We do not intend to specify the definitive realization of any of the operations, but simply convince the reader that the model's operations have physical interpretations. Furthermore, although we concentrate on implementations employing visible light (optical frequencies detectable to the human eye), the CSM definition does not preclude employing other portions of the electromagnetic spectrum.

A complex-valued image could be represented physically by a spatially coherent optical wavefront. Spatially coherent illumination (light of a single wavelength and emitted with the same phase angle) can be produced by a laser. SLM could be used to encode the image onto the expanded and collimated laser beam. One could write to an SLM offline (expose photographic film or laser print or relief etch a transparency) or online (in the case of a liquid-crystal display (Naughton et al. 1999; Wang and Saffman 1999; Yu et al. 1990) or holographic material (Chen et al. 1968; Pu et al. 1997)). The functions  $h$  and  $v$  could be effected using two convex cylindrical lenses, oriented horizontally and vertically, respectively (Goodman 1977, 2005; Naughton et al. 1999; VanderLugt 1992). As mentioned, a coherent optical wavefront will naturally evolve into its own Fourier spectrum as it propagates to infinity. What we do with a convex lens is simply image, at a finite distance, this spectrum at infinity. This finite distance is called the focal length of the lens. The constant 9 used in the definitions of  $h$  and  $v$  could be effected using Fourier spectrum

size reduction techniques (Goodman 2005; VanderLugt 1992) such as varying the focal length of the lens, varying the separation of the lens and SLM, employing cascaded Fourier transformation, increasing the dimensions/reducing the spatial resolution of the SLM, or using light with a shorter wavelength. The function  $*$  could be implemented using a phase conjugate mirror (Chiou 1999). The function could be realized by placing an SLM encoding an image  $f$  in the path of a wavefront encoding another image  $g$  (Goodman 2005; VanderLugt 1964, 1992). The wavefront immediately behind the SLM would then be  $(f \cdot g)$ . The function  $+$  describes the superposition of two optical wavefronts. This could be achieved using a 50:50 beam splitter (Chiou 1999; VanderLugt 1992; Weaver and Goodman 1966). The function  $\rho$  could be implemented using an electronic camera or a liquid-crystal light valve (Wang and Saffman 1999). The parameters  $z_1$  and  $z_u$  would then be physical characteristics of the particular camera/light valve used.  $z_1$  corresponds to the minimum intensity value that the device responds to, known as the dark current signal, and  $z_u$  corresponds to the maximum intensity (the saturation level).

A note will be made about the possibility of automating these operations. If suitable SLMs can be prepared with the appropriate 2D pattern(s), each of the operations  $h$ ,  $v$ ,  $*$ ,  $\bullet$ , and  $+$  could be effected autonomously and without user intervention using appropriately positioned lenses and free space propagation. The time to effect these operations would be the sum of the flight time of the image (distance divided by velocity of light) and the response time of the analog 2D detector, both of which are constants independent of the size or resolution of the images if an appropriate 2D detector is chosen. Examples of appropriate detectors would be holographic material (Chen et al. 1968; Pu et al. 1997) and a liquid-crystal light valve with a continuous (not pixelated) area (Wang and Saffman 1999). Since these analog detectors are also optically addressed SLMs, we can very easily arrange for the output of one function to act as the input to another, again in constant time independent of the size or resolution of the image. A set of angled mirrors will allow the optical image to be fed

back to the first SLM in the sequence, also in constant time. It is not known, however, if  $\rho$  can be carried out completely autonomously for arbitrary parameters. Setting arbitrary parameters might fundamentally require offline user intervention (adjusting the gain of the camera and so on), but at least for a small range of values, this can be simulated online using a pair of liquid-crystal intensity filters.

We have outlined some optics principles that could be employed to implement the operations of the model. The simplicity of the implementations hides some imperfections in our suggested realizations. For example, the implementation of the  $+$  operation outlined above results in an output image that has been unnecessarily multiplied by the constant factor 0.5 due to the operation of the beam splitter. Also, in our suggested technique, the output of the  $\rho$  function is squared unnecessarily. However, all of these effects can be compensated for with a more elaborate optical setup and/or at the algorithm design stage.

A more important issue concerns the quantum nature of light. According to our current understanding, light exists as individual packets called photons. As such, in order to physically realize the CSM, one would have to modify it such that images would have discrete, instead of continuous, amplitudes. The atomic operations outlined above, in particular the Fourier transform, are not affected by the restriction to quantized amplitudes, as the many experiments with electron interference patterns indicate. We would still assume, however, that in the physical world, space is continuous.

A final issue concerns how a theoretically infinite Fourier spectrum could be represented by an image (or encoded by an SLM) of finite extent. This difficulty is addressed with the FREQ complexity measure (Woods and Naughton 2005).

## Example CSM Data Structures and Algorithms

In this section, we give some example data representations. We then go on to give an example of CSM algorithm that efficiently squares a binary matrix.

## Representing Data as Images

There are many ways to represent data as images, and interesting new algorithms sometimes depend on a new data representation. Data representations should be in some sense reasonable, for example, it is unreasonable that the input to an algorithm could (nonuniformly) encode solutions to NP-hard or even undecidable problems.

From section “[C<sub>2</sub>-CSM](#),” the CSM address encoding function gives the programmer room to be creative, so long as the representation is logspace computable (assuming a reasonable representation of images as words).

Here we mention some data representations that are commonly used. Figure 5a, b is the binary symbol image representations of 1 and 0, respectively. These images have an everywhere constant value of 1 and 0, respectively, and both have SPATIALRES of 1. The row and column image representations of the word 1011 are respectively given in Fig. 5c, d. These row and column images both have SPATIALRES of 4. In the matrix image representation in Fig. 5e, the first matrix element is represented at the top-left corner, and elements are ordered in the usual matrix way. This  $3 \times 4$  matrix image has SPATIALRES of 12. Finally, the binary stack image representation, which has exponential SPATIALRES of 16, is given in Fig. 5f.

Figure 4 shows how we might form a list image by copying four images to one in a single timestep. All of the abovementioned images have DYRANGE, AMPLRES, and PHASERES of 1.

Another useful representation is where the value of a pixel directly encodes a number; in this case DYRANGE becomes crucial. We can also encode values as phase values, and naturally, PHASERES becomes a useful measure of the resources needed to store such values.

## A Matrix Squaring Algorithm

Here we give an example of CSM algorithm (taken from Woods 2006) that makes use of the data representations described above. The algorithm squares  $n \times n$  matrix in  $O(\log n)$  TIME and  $O(n^3)$  SPATIALRES (number of pixels), while all other CSM resources are constant.

### Lemma 1

Let  $n$  be a power of 2 and let  $A$  be a  $n \times n$  binary matrix. The matrix  $A^2$  is computed by a  $C_2$ -CSM, using the matrix image representation, in TIME  $O(\log n)$ , SPATIALRES  $O(n^3)$  GRID  $O(1)$ , DYRANGE  $O(1)$ , AMPLRES 1, and PHASERES 1.

### Proof

In this proof the matrix and its matrix image representation (see Fig. 5e) are both denoted  $A$ . We begin with some precomputation, then one parallel pointwise multiplication step, followed by  $\log n$  additions to complete the algorithm.

We generate the matrix image  $A_1$  that consists of  $n$  vertically juxtaposed copies of  $A$ . This is computed by placing one copy of  $A$  above the other, scaling to one image, and repeating to give a total of  $\log n$  iterations. The image  $A_1$  is constructed in TIME  $O(\log n)$ , GRID  $O(1)$ , and SPATIALRES  $O(n^3)$ .

Next, we transpose  $A$  to the column image  $A_2$ . The first  $n$  elements of  $A_2$  are row 1 of  $A$ , the second  $n$  elements of  $A_2$  are row 2 of  $A$ , etc. This is computed in TIME  $O(\log n)$ , GRID  $O(1)$ , and SPATIALRES  $O(n^2)$  as follows. Let  $A' = A$  and  $i = n$ . We horizontally split  $A'$  into a left image  $A'_L$  and a right image  $A'_R$ . Then,  $A'_L$  is pointwise multiplied (or masked) by the column image that represents  $(10)^i$ , in TIME  $O(1)$ . Similarly,  $A'_R$  is pointwise multiplied (or masked) by the column image that represents  $(01)^i$ . The masked images are added. The resulting image has half the number of columns as  $A'$  and double the number of rows, and, for example, row 1 consists of the first half of the elements of row 1 of  $A'$  and row 2 consists of the latter half of the elements of row 1 of  $A'$ . We call the result  $A'$  and we double the value of  $i$ . We repeat the process to give a total of  $\log n$  iterations. After these iterations, the resulting column image is denoted  $A_2$ .

We pointwise multiply  $A_1$  and  $A_2$  to give  $A_3$  in TIME  $O(1)$ , GRID  $O(1)$ , and SPATIALRES  $O(n^3)$ .

To facilitate a straightforward addition, we first transpose  $A_3$  in the following way:  $A_3$  is vertically split into a bottom and a top image, the top image is placed to the left of the bottom, and the two are scaled to a single image; this splitting and scaling is repeated to give a total of  $\log n$  iterations, and we call the result  $A_4$ . Then, to perform the addition, we vertically split  $A_4$  into a bottom and a top image. The top image is pointwise added to the bottom image and the result is thresholded between 0 and 1. This splitting, adding, and thresholding are repeated a total of  $\log n$  iterations to create  $A_5$ . We “reverse” the transposition that created  $A_4$ : image  $A_5$  is horizontally split into a left and a right image, the left image is placed above the right, and the two are scaled to a single image; this splitting and scaling is repeated a total of  $\log n$  iterations to give  $A^2$ .

The algorithm highlights a few points of interest about the CSM. The CSM has quite a number of space-like resources, and it is possible to have trade-offs between them. For example, in the algorithm above, if we allow GRID to increase from  $O(1)$  to  $O(n)$ , then the SPATIALRES can be reduced from  $O(n^3)$  to  $O(n^2)$ . In terms of optical architectures modeled by the CSM, this phenomenon could be potentially very useful as certain resources may well be more economically viable than others. The algorithm is used in the proof that that polynomial TIME CSMs (and  $\mathcal{C}_2$ -CSMs, see below) compute problems that are in the PSPACE class of languages. PSPACE includes the famous NP class. Such computational complexity results are discussed further in section “[Optical Computing and Computational Complexity](#)” below.

There are a number of existing CSM algorithms; for these, we point the reader to the literature (Naughton 2000a, b; Naughton and Woods 2001; Woods 2005a, 2006; Woods and Gibson 2005b; Woods and Naughton 2005).

## $\mathcal{C}_2$ -CSM

In this section we define the  $\mathcal{C}_2$ -CSM. One of the motivations for this model is the need to put reasonable upper bounds on the power of reasonable optical computers. As discussed below, it

turns out that CSMs can very quickly use massive amounts of resources, and the  $\mathcal{C}_2$ -CSM definition is an attempt to rein in this power.

### Worst-Case CSM Resource Usage

For the case of sequential computation, it is usually obvious how the execution of a single operation will affect resource usage. In parallel models, execution of a single operation can lead to large growth in a single timestep. Characterizing resource growth is useful for proving upper bounds on power and choosing reasonable model restrictions.

We investigated the growth of complexity resources over TIME, with respect to CSM operations (Woods 2005a; Woods and Gibson 2005a). As expected, under certain operations, some measures do not grow at all. Others grow at rates comparable to massively parallel models. By allowing operations like the Fourier transform, we are mixing the continuous and discrete worlds; hence, some measures grow to infinity in one timestep. This gave strong motivation for CSM restrictions.

Table 1 summarizes these results; the table defines the value of a complexity measure after execution of an operation (at TIME  $T + 1$ ). The complexity of a configuration at TIME  $T + 1$  is at least the value it was at TIME  $T$ , since complexity functions are nondecreasing. Our definition of TIME assigns unit time cost to each operation; hence, we do not have a TIME column. Some entries are immediate from the complexity measure definitions, for others proofs are given in the references (Woods 2005a; Woods and Gibson 2005a).

### $\mathcal{C}_2$ -CSM

Motivated by a desire to apply standard complexity theory tools to the model, we defined (Woods 2005a; Woods and Gibson 2005a) the  $\mathcal{C}_2$ -CSM, a restricted class of CSM.

#### Definition 7

( $\mathcal{C}_2$ -CSM) A  $\mathcal{C}_2$ -CSM is a CSM whose computation TIME is defined for  $t \in \{1, 2, \dots, T(n)\}$  and has the following restrictions:

(continued)

**Optical Computing, Table 1** CSM resource usage after one timestep. For a given operation and complexity measure pair, the relevant table entry defines the worst-case CSM resource usage at TIME  $T + 1$ , in terms of the resources

	GRID	SPATIALRES	AMPLRES	DYRANGE	PHASERES	FREQ
$h$	$G_T$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$v$	$G_T$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
*	$G_T$	$R_{s,T}$	$R_{a,T}$	$R_{d,T}$	$R_{p,T}$	$\mathcal{V}_T$
.	$G_T$	$R_{s,T}$	$(R_{a,T})^2$	$(R_{d,T})^2$	$R_{p,T}$	$\mathcal{V}_T$
+	$G_T$	$R_{s,T}$	$\infty$	$2R_{d,T}$	$\infty$	$\mathcal{V}_T$
$\rho$	Unbounded	$R_{s,T}$	$R_{a,T}$	$R_{d,T}$	$R_{p,T}$	$\mathcal{V}_T$
$St$	Unbounded	$R_{s,T}$	$R_{a,T}$	$R_{d,T}$	$R_{p,T}$	$\mathcal{V}_T$
$Id$	Unbounded	Unbounded	$R_{a,T}$	$R_{d,T}$	$R_{p,T}$	Unbounded
$br$	$G_T$	$R_{s,T}$	$R_{a,T}$	$R_{d,T}$	$R_{p,T}$	$\mathcal{V}_T$
$hlt$	$G_T$	$R_{s,T}$	$R_{a,T}$	$R_{d,T}$	$R_{p,T}$	$\mathcal{V}_T$

- For all TIME  $t$ , both AMPLRES and PHASERES have constant value of 2.
- For all TIME  $t$ , each of GRID, SPATIALRES, and DYRANGE is  $2^{O(t)}$ , and SPACE is redefined to be the product of all complexity measures except TIME and FREQ.
- Operations  $h$  and  $v$  compute the discrete Fourier transform in the horizontal and vertical directions, respectively.
- Given some *reasonable* binary word representation of the set of addresses  $\mathcal{N}$ , the address encoding function  $\mathfrak{E} : \mathbb{N} \rightarrow \mathcal{N}$  is decidable by a logspace Turing machine.

Let us discuss these restrictions. The restrictions on AMPLRES and PHASERES imply that  $\mathcal{C}_2$ -CSM images are of the form  $f : [0, 1] \times [0, 1] \rightarrow \{0, \pm 1/2, \pm 1, \pm 3/2, \dots\}$ . We have replaced the Fourier transform with the discrete Fourier transform (Bracewell 1978); this essentially means that FREQ is now solely dependent on SPATIALRES; hence, FREQ is not an interesting complexity measure for  $\mathcal{C}_2$ -CSMs, and we do not analyze  $\mathcal{C}_2$ -CSMs in terms of FREQ complexity (Woods 2005a; Woods and Gibson 2005a). Restricting the growth of SPACE is not unique to our model;

used at TIME  $T$ . At TIME  $T$  we have GRID =  $G_T$ , SPATIALRES =  $R_{s,T}$ , AMPLRES =  $R_{a,T}$ , DYRANGE =  $R_{d,T}$ , PHASERES =  $R_{p,T}$ , and FREQ =  $\mathcal{V}_T$

such restrictions are to be found elsewhere (Goldschlager 1982; Parberry 1987; Pratt and Stockmeyer 1976).

In section “[CSM Definition](#),” we stated that the address encoding function  $\mathfrak{E}$  should be Turing machine decidable; here, we strengthen this condition. At first glance sequential logspace computability may perhaps seem like a strong restriction, but in fact it is quite weak. From an optical implementation point of view, it should be the case that  $\mathfrak{E}$  is not complicated; otherwise, we cannot assume fast addressing. Other (sequential/parallel) models usually have a very restricted “addressing function”: in most cases it is simply the identity function on  $\mathbb{N}$ . Without an explicit or implicit restriction on the computational complexity of  $\mathfrak{E}$ , finding nontrivial upper bounds on the power of  $\mathcal{C}_2$ -CSMs is impossible as  $\mathfrak{E}$  could encode an arbitrarily complex Turing machine. As a weaker restriction, we could give a specific  $\mathfrak{E}$ . However, this restricts the generality of the model and prohibits the programmer from developing novel, reasonable, addressing schemes.

## Optical Computing and Computational Complexity

There have been a number of optical algorithms given that use the inherent parallelism of optics to speed up the solutions to certain problems. An

alternative approach is to ask the following question: How does a given optical model relate to standard sequential and parallel models? Establishing a relationship with computational complexity theory, by describing both upper and lower bounds on the model, gives immediate access to a large collection of useful algorithms and proof techniques.

The parallel computation thesis (Chandra and Stockmeyer 1976; Goldschlager 1977; Karp and Ramachandran 1990; Parberry 1987; van Emde Boas 1990) states that parallel time (polynomially) corresponds to sequential space, for reasonable parallel and sequential models. An example would be the fact that the class of problems solvable in polynomial space on a number of parallel models is equivalent to PSPACE, the class of problems solvable on Turing machines that use at most polynomial space (Alhazov and de Jesús Pérez-Jiménez 2007; Borodin 1977; Chandra et al. 1981; Fortune and Wyllie 1978; Goldschlager 1978, 1982; Hartmanis and Simon 1974; Sosik 2003; Sosik and Rodriguez-Paton 2007; Tromp and van Emde Boas 1993; van Leeuwen and Wiedermann 1987).

Of course the thesis can never be proved; it relates the intuitive notion of reasonable parallelism to the precise notion of a Turing machine. When results of this type were first shown, researchers were suitably impressed; their parallel models truly had great power. For example, if model  $M$  verifies the thesis, then  $M$  decides PSPACE (including NP) languages in polynomial time. However, there is another side to this coin. It is straightforward to verify that given our current best algorithms,  $M$  will use at least a superpolynomial amount of some other resource (like space or number of processors) to decide a PSPACE-complete or NP-complete language. Since the composition of polynomials is itself a polynomial, it follows that if we restrict the parallel computer to use at most polynomial time and polynomial other resources, then it can at most solve problems in P.

Nevertheless, asking if  $M$  verifies the thesis is an important question. Certain problems, such as those in the class NC, are efficiently parallelizable. NC can be defined as the class of

problems that are solvable in polylogarithmic time on a parallel computer that uses a polynomial amount of hardware. So one can think of NC as those problems in P which are solved exponentially faster on parallel computation thesis models than on sequential models. If  $M$  verifies the thesis, then we know it will be useful to apply  $M$  to these problems. We also know that if  $M$  verifies the thesis, then there are (P-complete) problems for which it is widely believed that we will not find exponential speed up using  $M$ .

### $\mathcal{C}_2$ -CSM and Parallel Complexity Theory

Here we summarize some characterizations of the computing power of optical computers. Such characterizations enable the algorithm designer to know what kinds of problems are solvable with resource bounded optical algorithms.

Theorem 1 below gives lower bounds on the computational power of the  $\mathcal{C}_2$ -CSM by showing that it is at least as powerful as models that verify the parallel computation thesis.

**Theorem 1 (Woods 2006; Woods and Gibson 2005b)**

$$\text{NSPACE}(S(n)) \subseteq \mathcal{C}_2 - \text{CSM} - \text{TIME}(O(S^2(n)))$$

In particular, polynomial TIME  $\mathcal{C}_2$ -CSMs accept the PSPACE languages. PSPACE is the class of problems solvable by Turing machines that use polynomial space, which includes the famous class NP, and so NP-complete problems can be solved by  $\mathcal{C}_2$ -CSMs in polynomial TIME. However, any  $\mathcal{C}_2$ -CSM algorithm that we could presently write to solve PSPACE or NP problems would require exponential SPACE.

Theorem 1 is established by giving a  $\mathcal{C}_2$ -CSM algorithm that efficiently generates, and squares, the transition matrix of a  $S(n) = \Omega(\log n)$  space-bounded Turing machine. This transition matrix represents all possible computations of the Turing machine and is of size  $O(2^s) \times O(2^s)$ . The matrix squaring part was already given as an example (Lemma 1), and the remainder of the algorithm is given in Woods (2006). The algorithm uses space

that is cubic in one of the matrix dimensions. In particular the algorithm uses cubic SPATIALRES,  $O(2^{3S})$ , and all other space-like resources are constant. This theorem improves upon the time overhead of a previous, but similar, result (Woods 2005a; Woods and Gibson 2005b) that was established via  $\mathcal{C}_2$ -CSM simulation of the vector machines (Pratt and Stockmeyer 1976; Pratt et al. 1974) of Pratt, Rabin, and Stockmeyer.

From the resource usage point of view, it is interesting to see that the older of these two algorithms uses GRID, DYRANGE, and SPATIALRES that are each  $O(2^S)$ , while the newer algorithm shows that if we allow more SPATIALRES, we can in fact use only constant GRID and DYRANGE. It would be interesting to find other such resource trade-offs within the model.

Since NP is contained in PSPACE, Theorem 1 and the corresponding earlier results in Woods (2005a) and Woods and Gibson (2005b) show that this optical model solves NP-complete problems in polynomial TIME. As described in section “[Optical Models of Computation](#),” this has also been shown experimentally, for example, Shaked et al. (2006) have recently given a polynomial time, exponential space, and optical algorithm to solve the NP-complete traveling salesperson problem. Their optical setup can be implemented on the CSM.

The other of the two inclusions that are necessary in order to verify the parallel computation thesis has also been shown:  $\mathcal{C}_2$ -CSMs computing in TIME  $T(n)$  are no more powerful than  $T^{\tilde{O}(1)}(n)$  space-bounded deterministic Turing machines. More precisely, we have:

#### **Theorem 2 (Woods 2005a; Woods 2005b)**

$$\mathcal{C}_2 - \text{CSM} - \text{TIME}(T(n)) \subseteq \text{DSPACE}(O(T^2(n)))$$

This result gives an upper bound on the power of  $\mathcal{C}_2$ -CSMs and was established via  $\mathcal{C}_2$ -CSM simulation by logspace uniform circuits of size and depth polynomial in SPACE and TIME, respectively (Woods 2005b).

Via the proofs of Theorems 1 and 2, we get another (stronger) result:  $\mathcal{C}_2$ -CSMs that

simultaneously use polynomial SPACE and polylogarithmic TIME solve exactly those problems in the class NC.

#### **Corollary 1**

$$\mathcal{C}_2 - \text{CSM} - \text{SPACE}, \text{TIME}(n^{O(1)}, \log^{O(1)} n) = \text{NC}$$

Problems in NC highlight the power of parallelism, as these problems can be solved exponentially faster on a polynomial amount of parallel resources than on polynomial time sequential machines. As further work in this area, one could try to find alternate characterizations of NC in terms of the  $\mathcal{C}_2$ -CSM. In particular, one could try to find further interesting trade-offs between the various space-like resources of the model. In the real world, this would correspond to computing over various different CSM resources. Also, it might be interesting for optical algorithm designers to try to design optical algorithms for NC problems in an effort to find problems that are well suited to optical solutions. See Woods and Naughton (2008) for details on this argument and also for other CSM characterizations of complexity classes and an implementation of a fast optical search algorithm.

## **Future Directions**

As already noted, optical computing is an inherently multidisciplinary subject whose study routinely involves a spectrum of expertise that threads optical physics, materials science, optical engineering, electrical engineering, computer architecture, computer programming, and computer theory. From the point of view of each of these fields, there are various directions for future work. Also, it is generally accepted that if optical computers become mainstream, it will be through a symbiotic relationship with their extremely flexible digital electronic counterparts.

At the confluence of computing and communication, there is room for optical techniques such as for the routing of signals in long-haul networks via all-optical packet switching in a single channel

configuration. So it seems that whether or not optical computers will be adopted in a widespread manner is both a technological and economic issue.

From the algorithmic point of view, there is plenty of scope for future work. There are a number of questions related to trade-offs between resources, and we believe the CSM gives a good framework to answer such questions. For example, can we give useful parallel algorithms that exploit CSM resources such as PHASERES while at the same time using small SPATIALRES and GRID? In a similar vein, one can explore the computing power of restrictions and generalizations of the CSM with the goal of finding new algorithms and characterizations of complexity classes. This has immediate applications in finding new and efficient implementations of optical solutions to computational problems.

## Bibliography

- Abushagur MAG, Caulfield HJ (1987) Speed and convergence of bimodal optical computers. *Opt Eng* 26(1):22–27
- Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266:1021–1024
- Alhazov A, de Jesús Pérez-Jiménez M (2007) Uniform solution to QSAT using polarizationless active membranes. In: Durand-Lose J, Margenstern M (eds) *Machines, Computations and Universality (MCU)*, vol 4664, LNCS. Springer, Orléans, pp 122–133
- Armitage JD, Lohmann AW (1965) Character recognition by incoherent spatial filtering. *Appl Opt* 4(4):461–467
- Arsenault HH, Sheng Y (1992) An introduction to optics in computers, vol TT8, Tutorial texts in optical engineering. SPIE Press, Bellingham
- Arsenault HH, Hsu YN, Chalasinska-Macukow K (1984) Rotation-invariant pattern recognition. *Opt Eng* 23(6):705–709
- Balcázar JL, Díaz J, Gabarró J (1988) Structural complexity, vol I and II, EATCS monographs on theoretical computer science. Springer, Berlin
- Beyette FR Jr, Mitkas PA, Feld SA, Wilmsen CW (1994) Bitonic sorting using an optoelectronic recirculating architecture. *Appl Opt* 33(35):8164–8172
- Borodin A (1977) On relating time and space to size and depth. *SIAM J Comput* 6(4):733–744
- Bracewell RN (1978) The Fourier transform and its applications, 2nd edn, Electrical and electronic engineering series. McGraw-Hill, New York
- Brenner KH, Huang A, Streibl N (1986) Digital optical computing with symbolic substitution. *Appl Opt* 25(18):3054–3060
- Brenner KH, Kufner M, Kufner S (1990) Highly parallel arithmetic algorithms for a digital optical processor using symbolic substitution logic. *Appl Opt* 29(11): 1610–1618
- Casasent DP (1984) Unified synthetic discriminant function computational formulation. *Appl Opt* 23:1620–1627
- Casasent DP, House GP (1994a) Comparison of coherent and noncoherent optical correlators. In: Optical pattern recognition V. Proceedings of SPIE, vol 2237. SPIE, Bellingham, pp 170–178
- Casasent DP, House GP (1994b) Implementation issues for a noncoherent optical correlator. In: Optical pattern recognition V. Proceedings of SPIE, vol 2237. SPIE, Bellingham, pp 179–188
- Casasent DP, Psaltis D (1976a) Position, rotation, and scale invariant optical correlation. *Appl Opt* 15(7):1795–1799
- Casasent DP, Psaltis D (1976b) Scale invariant optical transforms. *Opt Eng* 15(3):258–261
- Casasent DP, Jackson J, Neuman CP (1983) Frequency-multiplexed and pipelined iterative optical systolic array processors. *Appl Opt* 22:115–124
- Caulfield HJ (ed) (1979) *Handbook of optical holography*. Academic, New York
- Caulfield HJ (1989a) Computing with light. *Byte* 14: 231–237
- Caulfield HJ (1989b) The energetic advantage of analog over digital computing, vol 9, OSA optical computing technical digest series. Optical Society of America, Washington, DC, pp 180–183
- Caulfield HJ (1990) Space-time complexity in optical computing. In: Javidi B (ed) *Optical information-processing systems and architectures II*, vol 1347, SPIE. SPIE Press, Bellingham, pp 566–572
- Caulfield HJ, Abushagur MAG (1986) Hybrid analog-digital algebra processors. In: Optical and hybrid computing II. Proceedings of SPIE, vol 634. SPIE Press, Bellingham, pp 86–95
- Caulfield HJ, Haimes R (1980) Generalized matched filtering. *Appl Opt* 19(2):181–183
- Caulfield HJ, Horvitz S, Winkle WAV (1977) Introduction to the special issue on optical computing. *Proc IEEE* 65(1):4–5
- Caulfield HJ, Rhodes WT, Foster MJ, Horvitz S (1981) Optical implementation of systolic array processing. *Opt Commun* 40:86–90
- Caulfield HJ, Kinser JM, Rogers SK (1989) Optical neural networks. *Proc IEEE* 77:1573–1582
- Cerf NJ, Adami C, Kwiat PG (1998) Optical simulation of quantum logic. *Phys Rev A* 57(3):R1477–R1480
- Chandra AK, Stockmeyer LJ (1976) Alternation. In: 17th annual symposium on foundations of computer science, IEEE, Houston, pp 98–108
- Chandra AK, Kozen DC, Stockmeyer LJ (1981) Alternation. *J ACM* 28(1):114–133
- Chang S, Arsenault HH, Garcia-Martinez P, Grover CP (2000) Invariant pattern recognition based on centroids. *Appl Opt* 39(35):6641–6648
- Chen FS, LaMacchia JT, Fraser DB (1968) Holographic storage in lithium niobate. *Appl Phys Lett* 13(7):223–225
- Chiou AE (1999) Photorefractive phase-conjugate optics for image processing, trapping, and manipulation of microscopic objects. *Proc IEEE* 87(12):2074–2085

- Clavero R, Ramos F, Martí J (2005) All-optical flip-flop based on an active Mach-Zehnder interferometer with a feedback loop. *Opt Lett* 30(21):2861–2863
- Cutrona LJ, Leith EN, Palermo CJ, Porcello LJ (1960) Optical data processing and filtering systems. *IRE Trans Inf Theory* 6(3):386–400
- Cutrona LJ, Leith EN, Porcello LJ, Vivian WE (1966) On the application of coherent optical processing techniques to synthetic-aperture radar. *Proc IEEE* 54(8): 1026–1032
- Desmulliez MPY, Wherrett BS, Waddie AJ, Snowdon JF, Dines JAB (1996) Performance analysis of self-electro-optic-effect-device-based (seed-based) smart-pixel arrays used in data sorting. *Appl Opt* 35(32): 6397–6416
- Dolev S, Fitoussi H (2007) The traveling beam: optical solution for bounded NP-complete problems. In: Crescenzi P, Prencipe G, Pucci G (eds) *The fourth international conference on fun with algorithms (FUN)*. Springer, Heidelberg, pp 120–134
- Dorren HJS, Hill MT, Liu Y, Calabretta N, Srivatsa A, Huijskens FM, de Waardt H, Kho GD (2003) Optical packet switching and buffering by using all-optical signal processing methods. *J Lightwave Technol* 21(1):2–12
- Durand-Lose J (2006) Reversible conservative rational abstract geometrical computation is Turing-universal. In: *Logical approaches to computational barriers, second conference on computability in Europe, (CiE)*. vol 3988, Lecture notes in computer science. Springer, Swansea, pp 163–172
- Efron U, Grinberg J, Braatz PO, Little MJ, Reif PG, Schwartz RN (1985) The silicon liquid crystal light valve. *J Appl Phys* 57(4):1356–1368
- Esteve-Taboada JJ, García J, Ferreira C (2000) Extended scale-invariant pattern recognition with white-light illumination. *Appl Opt* 39(8):1268–1271
- Farhat NH, Psaltis D (1984) New approach to optical information processing based on the Hopfield model. *J Opt Soc Am A* 1:1296
- Feitelson DG (1988) Optical computing: a survey for computer scientists. MIT Press, Cambridge, MA
- Feng JH, Chin GF, Wu MX, Yan SH, Yan YB (1995) Multiobject recognition in a multichannel joint-transform correlator. *Opt Lett* 20(1):82–84
- Fortune S, Wyllie J (1978) Parallelism in random access machines. In: *Proceedings 10th annual ACM symposium on theory of computing*. ACM, New York, pp 114–118
- Gabor D (1948) A new microscopic principle. *Nature* 161(4098):777–778
- Gara AD (1979) Real time tracking of moving objects by optical correlation. *Appl Opt* 18(2):172–174
- Geldenhuys R, Liu Y, Calabretta N, Hill MT, Huijskens FM, Kho GD, Dorren HJS (2004) All-optical signal processing for optical packet switching. *J Opt Netw* 3(12):854–865
- Ghosh AK, Casasent DP, Neuman CP (1985) Performance of direct and iterative algorithms on an optical systolic processor. *Appl Opt* 24(22):3883–3892
- Goldberg L, Lee SH (1979) Integrated optical half adder circuit. *Appl Opt* 18:2045–2051
- Goldschlager LM (1977) Synchronous parallel computation. PhD thesis, University of Toronto, Computer Science Department
- Goldschlager LM (1978) A unified approach to models of synchronous parallel machines. In: *Proceedings 10th annual ACM symposium on theory of computing*. ACM, New York, pp 89–94
- Goldschlager LM (1982) A universal interconnection pattern for parallel computers. *J ACM* 29(4):1073–1086
- Goodman JW (1977) Operations achievable with coherent optical information processing systems. *Proc IEEE* 65(1):29–38
- Goodman JW (2005) *Introduction to fourier optics*, 3rd edn. Roberts & Company, Englewood
- Grinberg J, Jacobson AD, Bleha WP, Miller L, Fraas L, Boswell D, Myer G (1975) A new real-time noncoherent to coherent light image converter: the hybrid field effect liquid crystal light valve. *Opt Eng* 14(3):217–225
- Grover LK (1996) A fast quantum mechanical algorithm for database search. In: *Proceedings 28th annual ACM symposium on theory of computing*. ACM, New York, pp 212–219
- Guilfoyle PS, Hessenbruch JM, Stone RV (1998) Free-space interconnects for high-performance optoelectronic switching. *IEEE Comput* 31(2):69–75
- Haist T, Osten W (2007) An optical solution for the travelling salesman problem. *Opt Express* 15(16): 10473–10482
- Hartmanis J, Simon J (1974) On the power of multiplication in random access machines. In: *Proceedings of the 15th annual symposium on switching and automata theory*. IEEE, The University of New Orleans, pp 13–23
- Head T (1987) Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bull Math Biol* 49(6):737–759
- Horner JL (ed) (1987) *Optical signal processing*. Academic, San Diego
- Hough PVC (1962) Methods and measures for recognising complex patterns. US Patent No. 3,069,654
- Hsu YN, Arsenault HH (1982) Optical pattern recognition using circular harmonic expansion. *Appl Opt* 21(22): 4016–4019
- Hsu KY, Li HY, Psaltis D (1990) Holographic implementation of a fully connected neural network. *Proc IEEE* 78(10):1637–1645
- Huang A (1984) Architectural considerations involved in the design of an optical digital computer. *Proc IEEE* 72(7):780–786
- Huang A, Tsunoda Y, Goodman JW, Ishihara S (1979) Optical computation using residue arithmetic. *Appl Opt* 18(2):149–162
- Jacobson AD, Beard TD, Bleha WP, Morgerum JD, Wong SY (1972) The liquid crystal light valve, an optical-to-optical interface device. In: *Proceedings of the conference on parallel image processing, document X-711-72-308*, Goddard Space Flight Center. NASA, Washington, DC, pp 288–299
- Javidi B (1989) Nonlinear joint power spectrum based optical correlation. *Appl Opt* 28(12):2358–2367

- Javidi B (1990) Generalization of the linear matched filter concept to nonlinear matched filters. *Appl Opt* 29(8):1215–1217
- Javidi B, Wang J (1995) Optimum distortion-invariant filter for detecting a noisy distorted target in nonoverlapping background noise. *J Opt Soc Am A* 12(12):2604–2614
- Karim MA, Awwal AAS (1992) Optical computing: an introduction. Wiley, New York
- Karp RM, Ramachandran V (1990) Parallel algorithms for shared memory machines. In: van Leeuwen J (ed) *Handbook of theoretical computer science*, vol A. Elsevier, Amsterdam, pp 869–941
- Knill E, LaFlamme R, Milburn GJ (2001) A scheme for efficient quantum computation with linear optics. *Nature* 409:46–52
- Lee JN (ed) (1995) Design issues in optical processing. Cambridge studies in modern optics. Cambridge University Press, Cambridge
- Leith EN (1977) Complex spatial filters for image deconvolution. *Proc IEEE* 65(1):18–28
- Lenslet Labs (2001) Optical digital signal processing engine. White paper report, Lenslet Ltd., 12 Hachilazon St., Ramat-Gan, Israel 52522
- Lipton RJ (1995) Using DNA to solve NP-complete problems. *Science* 268:542–545
- Lohmann AW (1993) Image rotation, Wigner rotation, and the fractional Fourier transform. *J Opt Soc Am A* 10(10):2181–2186
- Louri A, Post A (1992) Complexity analysis of optical-computing paradigms. *Appl Opt* 31(26):5568–5583
- Louri A, Hatch JA Jr, Na J (1995) Constant-time parallel sorting algorithm and its optical implementation using smart pixels. *Appl Opt* 34(17):3087–3097
- Lu XJ, Yu FTS, Gregory DA (1990) Comparison of Vander lugt and joint transform correlators. *Appl Phys B* 51:153–164
- McAulay AD (1991) Optical computer architectures: the application of optical concepts to next generation computers. Wiley, New York
- Mead C (1989) Analog VLSI and neural systems. Addison-Wesley, Reading
- Miller DA (2000) Rationale and challenges for optical interconnects to electronic chips. *Proc IEEE* 88(6):728–749
- Moore C (1991) Generalized shifts: undecidability and unpredictability in dynamical systems. *Nonlinearity* 4:199–230
- Moore C (1997) Majority-vote cellular automata, Ising dynamics and P-completeness. *J Stat Phys* 88(3/4):795–805
- Naughton TJ (2000a) Continuous-space model of computation is Turing universal. In: Bains S, Irakliotis LJ (eds) *Critical technologies for the future of computing*. Proc SPIE, vol 4109. SPIE Press, San Diego, pp 121–128
- Naughton TJ (2000b) A model of computation for Fourier optical processors. In: Lessard RA, Galstian T (eds) *Optics in computing 2000*. Proc SPIE, vol 4089. SPIE Press, Quebec, pp 24–34
- Naughton TJ, Woods D (2001) On the computational power of a continuous-space optical model of computation. In: Margenstern M, Rogozhin Y (eds) *Machines, computations and universality: third international conference (MCU'01)*, vol 2055, LNCS. Springer, Heidelberg, pp 288–299
- Naughton T, Javadpour Z, Keating J, Klíma M, Rott J (1999) General-purpose acousto-optic connectionist processor. *Opt Eng* 38(7):1170–1177
- O'Neill EL (1956) Spatial filtering in optics. *IRE Trans Inf Theory* 2:56–65
- Oltean M (2006) A light-based device for solving the Hamiltonian path problem. In: Fifth international conference on unconventional computation (UC'06). LNCS, vol 4135. Springer, New York, pp 217–227
- Paek EG, Choe JY, Oh TK, Hong JH, Chang TY (1997) Nonmechanical image rotation with an acousto-optic dove prism. *Opt Lett* 22(15):1195–1197
- Papadimitriou CH (1995) Computational complexity. Addison-Wesley, Reading
- Parberry I (1987) Parallel complexity theory. Wiley, New York
- Păun G (2002) Membrane computing: an introduction. Springer, Heidelberg
- Pe'er A, Wang D, Lohmann AW, Friesem AA (1999) Optical correlation with totally incoherent light. *Opt Lett* 24(21):1469–1471
- Pittman TB, Fitch MJ, Jacobs BC, Franson JD (2003) Experimental controlled-NOT logic gate for single photons in the coincidence basis. *Phys Rev A* 68:032316–3
- Pratt VR, Stockmeyer LJ (1976) A characterisation of the power of vector machines. *J Comput Syst Sci* 12: 198–221
- Pratt VR, Rabin MO, Stockmeyer LJ (1974) A characterisation of the power of vector machines. In: Proceedings of 6th annual ACM symposium on theory of computing. ACM, New York, pp 122–134
- Psaltis D, Farhat NH (1985) Optical information processing based on an associative-memory model of neural nets with thresholding and feedback. *Opt Lett* 10(2):98–100
- Pu A, Denkewalter RF, Psaltis D (1997) Real-time vehicle navigation using a holographic memory. *Opt Eng* 36(10):2737–2746
- Reif JH, Tyagi A (1997) Efficient parallel algorithms for optical computing with the discrete Fourier transform (DFT) primitive. *Appl Opt* 36(29):7327–7340
- Reif J, Tygar D, Yoshida A (1990) The computability and complexity of optical beam tracing. In: 31st annual IEEE symposium on Foundations of Computer Science (FOCS). IEEE, St. Louis, pp 106114
- Rhodes WT (1981) Acousto-optic signal processing: convolution and correlation. *Proc IEEE* 69(1):65–79
- Sawchuk AA, Strand TC (1984) Digital optical computing. *Proc IEEE* 72(7):758–779
- Shaked NT, Simon G, Tabib T, Mesika S, Dolev S, Rosen J (2006) Optical processor for solving the

- traveling salesman problem (TSP). In: Javidi B, Psaltis D, Caulfield HJ (eds) Proceedings of SPIE, optical information systems IV, vol 63110G. SPIE, Bellingham
- Shaked NT, Messika S, Dolev S, Rosen J (2007) Optical solution for bounded NP-complete problems. *Appl Opt* 46(5):711–724
- Shor P (1994) Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations computer science. ACM, New York, pp 124–134
- Sosik P (2003) The computational power of cell division in P systems: beating down parallel computers? *Nat Comput* 2(3):287–298
- Sosik P, Rodriguez-Paton A (2007) Membrane computing and complexity theory: a characterization of PSPACE. *J Comput Syst Sci* 73(1):137–152
- Stirk CW, Athale RA (1988) Sorting with optical compare-and-exchange modules. *Appl Opt* 27(9): 1721–1726
- Stone RV (1994) Optoelectronic processor is programmable and flexible. *Laser Focus World* 30(8):77–79
- Stone RV, Zeise FF, Guilfoyle PS (1991) DOC II 32-bit digital optical computer: optoelectronic hardware and software. In: Optical enhancements to computing technology, Proc SPIE, vol 1563. SPIE, Bellingham, pp 267–278
- Stroke GW, Halioua M, Thon F, Willasch DH (1974) Image improvement in high-resolution electron microscopy using holographic image deconvolution. *Optik* 41(3):319–343
- Sullivan DL (1972) Alignment of rotational prisms. *Appl Opt* 11(9):2028–2032
- Tromp J, van Emde Boas P (1993) Associative storage modification machines. In: Ambos-Spies K, Homer S, Schoning U (eds) Complexity theory: current research. Cambridge University Press, Cambridge, UK, pp 291–313
- Turin GL (1960) An introduction to matched filters. *IRE Trans Inf Theory* 6(3):311–329
- Upatnieks J (1983) Portable real-time coherent optical correlator. *Appl Opt* 22(18):2798–2803
- van Emde Boas P (1990) Machine models and simulations, chap 1. In: van Leeuwen J (ed) Handbook of theoretical computer science, vol A. Elsevier, Amsterdam
- van Leeuwen J, Wiedermann J (1987) Array processing machines. *BIT* 27:25–43
- VanderLugt A (1964) Signal detection by complex spatial filtering. *IEEE Trans Inf Theory* 10(2):139–145
- VanderLugt A (1974) Coherent optical processing. Proc IEEE 62(10):1300–1319
- VanderLugt A (1992) Optical signal processing. Wiley, New York
- Wang CH, Jenkins BK (1990) Subtracting incoherent optical neuron model – analysis, experiment and applications. *Appl Opt* 29(14):2171–2186
- Wang PY, Saffman M (1999) Selecting optical patterns with spatial phase modulation. *Opt Lett* 24(16): 1118–1120
- Weaver CS, Goodman JW (1966) A technique for optically convolving two functions. *Appl Opt* 5(7): 1248–1249
- Woods D (2005a) Computational complexity of an optical model of computation. PhD thesis, National University of Ireland, Maynooth
- Woods D (2005b) Upper bounds on the computational power of an optical model of computation. In: Deng X, Du D (eds) 16th international symposium on algorithms and computation (ISAAC 2005). LNCS, vol 3827. Springer, Heidelberg, pp 777–788
- Woods D (2006) Optical computing and computational complexity. In: Fifth international conference on unconventional computation (UC'06). LNCS, vol 4135. Springer, pp 27–40
- Woods D, Gibson JP (2005a) Complexity of continuous space machine operations. In: Cooper SB, Löwe B, Torenvliet L (eds) New computational paradigms, first conference on computability in Europe (CiE 2005), vol 3526, LNCS. Springer, Amsterdam, pp 540–551
- Woods D, Gibson JP (2005b) Lower bounds on the computational power of an optical model of computation. In: Calude CS, Dinneen MJ, Păun G, Pérez-Jiménez MJ, Rozenberg G (eds) Fourth international conference on unconventional computation (UC'05), vol 3699, LNCS. Springer, Heidelberg, pp 237–250
- Woods D, Naughton TJ (2005) An optical model of computation. *Theor Comput Sci* 334(1–3):227–258
- Woods D, Naughton TJ (2008) Parallel and sequential optical computing. In: International workshop on optical supercomputing. LNCS. Springer, Heidelberg
- Yamaguchi I, Zhang T (1997) Phase-shifting digital holography. *Opt Lett* 22(16):1268–1270
- Yokomori T (2002) Molecular computing paradigm – toward freedom from Turing's charm. *Nat Comput* 1(4):333–390
- Yu FTS (1996) Garden of joint transform correlators: an account of recent advances. In: Second international conference on optical information processing. Proc SPIE, vol 2969. SPIE, Bellingham, pp 396–401
- Yu FTS, Lu T, Yang X, Gregory DA (1990) Optical neural network with pocket-sized liquid-crystal televisions. *Opt Lett* 15(15):863–865
- Yu FTS, Jutamulia S, Yin S (eds) (2001) Introduction to information optics. Academic, San Diego
- Zhai H, Mu G, Sun J, Zhu X, Liu F, Kang H, Zhan Y (1999) Color pattern recognition in white-light, joint transform correlation. *Appl Opt* 38(35):7238–7244



# Principles of Neuromorphic Photonics

Bhavin J. Shastri, Alexander N. Tait,  
Thomas Ferreira de Lima, Mitchell A. Nahmias,  
Hsuan-Tung Peng and Paul R. Prucnal  
Department of Electrical Engineering, Princeton  
University, Princeton, NJ, USA

## Article Outline

Glossary  
Definition of the Subject  
Introduction  
Neuromorphic Computing: Beyond von Neumann and Moore  
Technology Platforms  
Neuromorphic Photonics  
Photonic Neuron  
Excitable/Spiking Lasers  
Photonic Neural Network Architecture  
Neuromorphic Platform Comparison  
Future Directions  
Summary and Conclusion  
Bibliography

## Glossary

**Benchmark** A standardized task that can be performed by disparate computing approaches, used to assess their relative processing merit in specific cases.

**Bifurcation** A qualitative change in behavior of a dynamical system in response to parameter variation. Examples include cusp (from monostable to bistable), Hopf (from stable to oscillating), and transcritical (exchange of stability between two steady states).

**Brain-inspired computing** (a.k.a. neuro-inspired computing) A biologically inspired approach to build processors, devices, and computing models for applications including adaptive control,

machine learning, and cognitive radio. Similarities with biological signal processing include architectural, such as distributed; representational, such as analog or spiking; or algorithmic, such as adaptation.

**Broadcast and Weight** A multiwavelength analog networking protocol in which multiple all photonic neuron outputs are multiplexed and distributed to all neuron inputs. Weights are reconfigured by tunable spectral filters.

**Excitability** A far-from-equilibrium nonlinear dynamical mechanism underlying all-or-none responses to small perturbations.

**Fan-in** The number of inputs to a neuron.

**Layered network** A network topology consisting of a series of sets (i.e., layers) of neurons. The neurons in each set project their outputs only to neurons in the subsequent layer. Most commonly used type of network used for machine learning.

**Metric** A quantity assessing performance of a device in reference to a specific computing approach.

**Microring weight bank** A silicon photonic implementation of a reconfigurable spectral filter capable of independently setting transmission at multiple carrier wavelengths.

**Modulation** The act of representing an abstract variable in a physical quantity, such as photon rate (i.e., optical power), free carrier density (i.e., optical gain), and carrier drift (i.e., current). Electro-optic modulators are devices that convert from an electrical signal to the power envelope of an optical signal.

**Moore's law** An observation that the number of transistors in an integrated circuit doubles every 18 to 24 months, doubling its performance.

**Multiply-accumulate (MAC)** A common operation that represents a single multiplication followed by an addition:  $a \leftarrow a + (b \times c)$ .

**Neural networks** A wide class of computing models consisting of a distributed set of nodes, called neurons, interconnected with configurable or adaptable strengths, called

weights. Overall neural network behavior can be extremely complex relative to single neuron behavior.

**Neuromorphic computing** Computing approaches based on specialized hardware that formally adheres to one or more neural network models. Algorithms, metrics, and benchmarks can be shared between disparate neuromorphic computers that adhere to a common mathematical model.

**Neuromorphic photonics** An emerging field at the nexus of photonics and neural network processing models, which combines the complementary advantages of optics and electronics to build systems with high efficiency, high interconnectivity, and extremely high bandwidth.

**Optoelectronics** A technology of electronic devices and systems (semiconductor lasers, photodetectors, modulators, photonic integrated circuits) that interact (source, detect, and control).

**Photonic integrated circuits (PICs)** A chip that integrates many photonic components (lasers, modulators, filters, detectors) connected by optical waveguides that guide light; similar to an electronic integrated circuit that consists of transistors, diodes, resistors, capacitors, and inductors, connected by conductive wires.

**Physical cascability** The ability of one neuron to produce an output with the same representational properties as its inputs. For example, photonic-electronicphotonic or 32bit-analog-32bit.

**Recurrent network** A network topology in which each neuron output can reach every other neuron, including itself. Every network is a subset of a recurrent network.

**Reservoir computing** A computational approach in which a complex, nonlinear substrate performs a diversity of functions, from which linear classifiers extract the most useful information to perform a given algorithm. The reservoir can be implemented by a recurrent neural network or a wide variety of other systems, such as time-delayed feedback.

**Semiconductor lasers** Lasers based on semiconductor gain media, where optical gain is

achieved by stimulated emission at an interband transition under conditions of a high carrier density in the conduction band.

**Signal cascability** The ability of one neuron to elicit an equivalent or greater response when driving multiple other neurons. The number of target neurons is called fan-out.

**Silicon photonics** A chip-scale, silicon-on-insulator (SOI) platform for monolithic integration of optics and microelectronics for guiding, modulating, amplifying, and detecting light.

**Spiking neural networks (SNNs)** A biologically realistic neural network model that processes information with spikes or pulses that encode information temporally.

**Wavelength-division multiplexing (WDM)**

One of the most common multiplexing techniques used in optics where different wavelengths (colors) of light are combined, transmitted, and separated again.

**WDM weighted addition** A simultaneous summation of power modulated signals and transduction from multiple optical carriers to one electronic carrier. Occurs when multiplexed optical signals impinge on a standard photodetector.

**Weight matrix** A way to describe all network connection strengths between neurons arranged such that rows are input neuron indices and columns are output neuron indices. The weight matrix can be constrained to be symmetric, block off-diagonal, sparse, etc. to represent particular kinds of neural networks.

**Weighted addition** The operation describing how multiple inputs to a neuron are combined into one variable. Can be implemented in the digital domain by multiply-accumulate (MAC) operations or in the analog domain by various physical processes (e.g., current summing, total optical power detection).

## Definition of the Subject

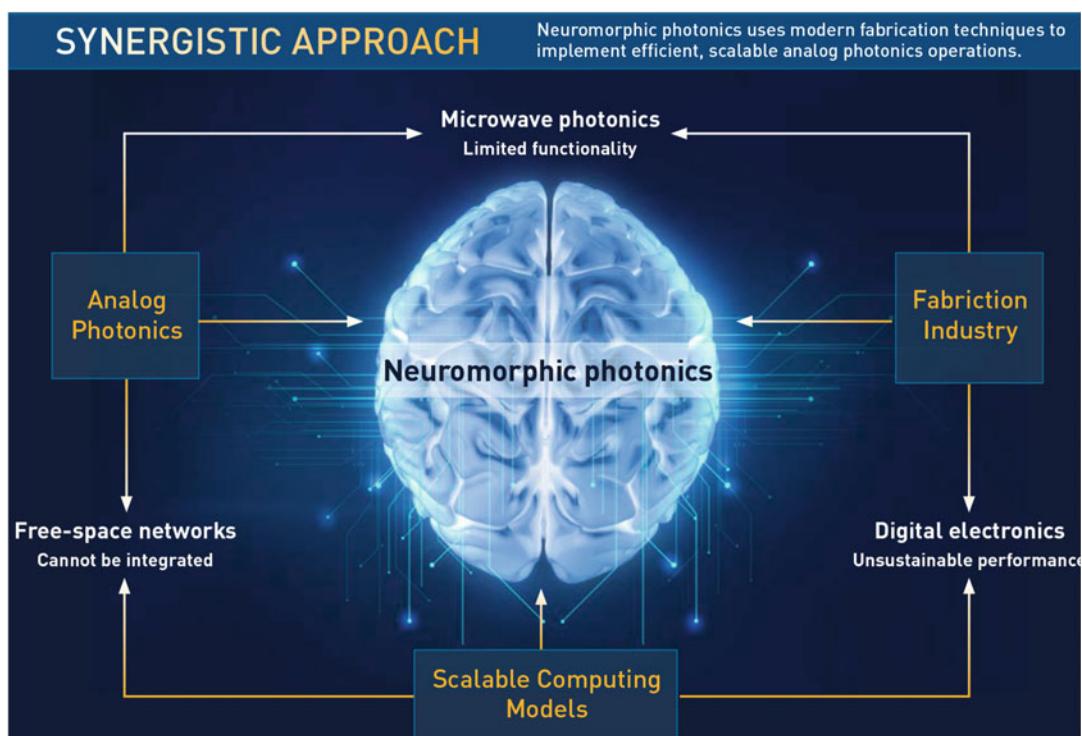
In an age overrun with information, the ability to process reams of data has become crucial. The demand for data will continue to grow as smart gadgets multiply and become increasingly

integrated into our daily lives. Next-generation industries in artificial intelligence services and high-performance computing are so far supported by microelectronic platforms. These data-intensive enterprises rely on continual improvements in hardware. Their prospects are running up against a stark reality: conventional one-size-fits-all solutions offered by digital electronics can no longer satisfy this need, as Moore's law (exponential hardware scaling), interconnection density, and the von Neumann architecture reach their limits.

With its superior speed and reconfigurability, analog photonics can provide some relief to these problems; however, complex applications of analog photonics have remained largely unexplored due to the absence of a robust photonic integration industry. Recently, the landscape for commercially manufacturable photonic chips has been changing rapidly and now promises to achieve economies of scale previously enjoyed solely by microelectronics.

Despite the advent of commercially viable photonic integration platforms, significant challenges still remain before scalable analog photonic processors can be realized. A central challenge is the development of mathematical bridges linking photonic device physics to models of complex analog information processing. Among such models, those of neural networks are perhaps the most widely studied and used by machine learning and engineering fields.

Recently, the scientific community has set out to build bridges between the domains of photonic device physics and neural networks, giving rise to the field of *neuromorphic photonics* (Fig. 1). This entry reviews the recent progress in integrated neuromorphic photonics. We provide an overview of neuromorphic computing, discuss the associated technology (microelectronic and photonic) platforms, and compare their metric performance. We discuss photonic neural network approaches and challenges for integrated neuromorphic



**Principles of Neuromorphic Photonics, Fig. 1** The advent of neuromorphic photonics is due to the convergence of recent advances in photonic integration

technology, resurgence of scalable computing models (e.g., spiking, deep neural networks), and a large-scale silicon industrial ecosystem

photonic processors while providing an in-depth description of photonic neurons and a candidate interconnection architecture. We conclude with a future outlook of neuro-inspired photonic processing.

## Introduction

Complexity manifests in our world in countless ways (Strogatz 2001; Vicsek 2002) ranging from intracellular processes (Crescenzi et al. 1998) and human brain area function (Markram et al. 2011) to climate dynamics (Donges et al. 2009) and world economy (Hidalgo et al. 2007). An understanding of complex systems is a fundamental challenge facing the scientific community. Understanding complexity could impact the progress of our society as a whole, for instance, in fighting diseases, mitigating climate change, or creating economic benefits.

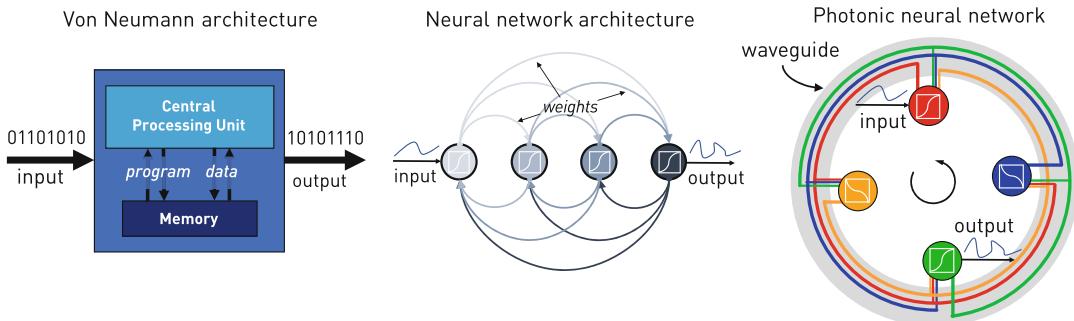
Current approaches to complex systems and big data analysis are based on software, executed on serialized and centralized von Neumann machines. However, the interconnected structure of complex systems (i.e., many elements interacting strongly and with variation (Bhalla and Iyengar 1999)) makes them challenging to reproduce in this conventional computing framework. Memory and data interaction bandwidths constrain the types of informatic systems that are feasible to simulate. The conventional computing approach has persisted thus far due to an exponential performance scaling in digital electronics, most famously embodied in Moore's law. For most of the past 60 years, the density of transistors, clock speed, and power efficiency in microprocessors have approximately doubled every 18 months. These empirical laws are fundamentally unsustainable. The last decade has witnessed a statistically significant (>99.95% likelihood (Marr et al. 2013)) plateau in processor energy efficiency.

This situation suggests that the time is ripe for radically new approaches to information processing, one being *neuromorphic photonics*. An emerging field at the nexus of photonics and neuroscience, neuromorphic photonics combines the

complementary advantages of optics and electronics to build systems with high efficiency, high interconnectivity, and extremely high bandwidth. This entry reviews the recent progress in integrated neuromorphic photonic research. First, we introduce neuromorphic computing and give an overview of current technology platforms in microelectronics and photonics. Next, we discuss the evolution of neuromorphic photonics, the present photonic neural network approaches, and challenges for emerging integrated neuromorphic photonic processors. We introduce the concept of a “photonic neuron” followed by a discussion on its feasibility. We summarize recent research on optical devices that could be used as network-compatible photonic neurons. We discuss a networking architecture that efficiently channelizes the transmission window of an integrated waveguide. We also compare metrics between neuromorphic electronics and neuromorphic photonics. Finally, we offer a glimpse at the future outlook for neuro-inspired photonic processing and discuss potential applications.

## Neuromorphic Computing: Beyond von Neumann and Moore

Conventional digital computers are based on the von Neumann architecture (von Neumann 1993). It consists of a memory that stores both data and instructions, a central processing unit (CPU), and inputs and outputs (Fig. 2 (left)). Instructions and data stored in the memory unit are separated from the CPU by a shared digital bus. This is known as the von Neumann bottleneck (Backus 1978) which fundamentally limits the performance of the system – a problem that is aggravated as CPUs become faster and memory units larger. This computing paradigm has dominated for over 60 years driven in part by the continual progress dictated by Moore's law (Moore 2000) for CPU scaling and Koomey's law (Koomey et al. 2011) for energy efficiency (in multiply-accumulate (MAC) operations per joule) compensating the bottleneck. Over the last several years, such scaling has not followed suit, approaching an asymptote. The computation efficiency levels off



**Principles of Neuromorphic Photonics, Fig. 2** Neural nets: The photonic edge. Von Neumann architectures (left), relying on sequential input-output through a central processor, differ fundamentally from more decentralized

neural network architectures (middle). Photonic neural nets (right) can solve the interconnect bottleneck by using one waveguide to carry signals from many connections (easily  $N_2 \sim 10,000$ ) simultaneously

below 10 GMAC/s/W or 100 pJ per MAC (Hasler and Marr 2013). The reasons behind this trend can be traced to both the representation of information at the physical level and the interaction of processing with memory at the architectural level (Marr et al. 2013).

Breaching the energy efficiency wall of digital computation by orders of magnitude is not fundamentally impossible. In fact, the human brain, believed to be the most complex system in the universe, is estimated to execute an amazing  $10^{18}$  MAC/s using only 20 W of power (Hasler and Marr 2013; Merkle 1989). It does this with  $10^{11}$  neurons with an average of  $10^4$  inputs each. This leads to an estimated total of  $10^{15}$  synaptic connections, all conveying signals up to 1 kHz bandwidth. The calculated computational efficiency for the brain (< aJ/MAC) is therefore eight orders of magnitude beyond that of current supercomputers (100 pJ/MAC). The brain is a natural standard for information processing, one that has been compared to artificial processing systems since their earliest inception. Nevertheless, the brain as a processor differs radically from computers today, both at the physical level and at the architectural level. Its exceptional performance is, at least partly, due to the neuron biochemistry, its underlying architecture, and the biophysics of neuronal computation algorithms.

*Neuromorphic computing* offers hope to building large-scale “bio-inspired” hardware whose computational efficiencies move toward those of

a human brain. In doing so, neuromorphic platforms (Fig. 2 (middle)) could break performance limitations inherent in traditional von Neumann architectures in solving particular classes of problems. Their distributed hardware architectures can most efficiently evaluate models with high data interconnection, among which are real-time complex system assurance and big data awareness.

At the device level, digital CMOS is reaching physical limits (Mathur 2002; Taur et al. 1997). As the CMOS feature sizes scale down below 90 nm to 65 nm, the voltage, capacitance, and delay no longer scale according to a well-defined rate by Dennard’s law (Dennard et al. 1974). This leads to a trade-off between performance (when transistor is on) and subthreshold leakage (when it is off). For example, as the gate oxide (which serves as an insulator between the gate and channel) is made as thin as possible (1.2 nm, around five atoms thick Si) to increase the channel conductivity, a quantum mechanical phenomenon of electron tunneling (Taur 2002; Lee and Hu 2001) occurs between the gate and channel leading to increased power

consumption. The recent shift to multi-core scaling alleviated these constraints, but the breakdown of Dennard scaling has limited the number of cores than can simultaneously be powered on with a fixed power budget and heat extraction rate. Fixed power budgets have necessitated so-called *dark silicon* strategies (Esmaeilzadeh et al. 2012). Projections for the 8 nm node indicate that over 50% of the chip will be *dark* (Esmaeilzadeh et al. 2012),

meaning unused at a given time. This has led to a widening rift between conventional computing capabilities and contemporary computing needs, particularly for the analysis of complex systems.

Computational tools have been revolutionary in hypothesis testing and simulation. They have led to the discovery of innumerable theories in science, and they will be an indispensable aspect of a holistic approach to problems in big data and many body physics; however, huge gaps between information structures in observed systems and standard computing architectures motivate a need for alternative paradigms if computational abilities are to be brought to the growing class of problems associated with complex systems. Brain-inspired computing approaches share the interconnected causal structures and dynamics analogous to the complex systems present in our world. This is in contrast to conventional approaches, where these structures are virtualized at considerable detriment to energy efficiency. From a usual constraint of a fixed power budget, energy efficiency caps overall simulation scale.

Over the years, there has been a deeply committed exploration of unconventional computing techniques (Hasler and Marr 2013; Keyes 1985; Jaeger and Haas 2004; Merolla et al. 2014; Modha et al. 2011; Tucker 2010; Caulfield and Dolev 2010; Woods and Naughton 2012; Benjamin et al. 2014; Pfeil et al. 2013; Furber et al. 2014; Snider 2007; Eliasmith et al. 2012; Indiveri et al. 2011; Brunner et al. 2013a; Tait et al. 2017; Shen et al. 2017; Shainline et al. 2017; Prucnal et al. 2016; Prucnal and Shastri 2017) to alleviate the device level and system/architectural level challenges faced by conventional computing platforms. Specifically, neuromorphic computing is going through an exciting period as it promises to make processors that use low energies while integrating massive amounts of information. Neuromorphic engineering aims to build machines employing basic nervous systems operations by bridging the physics of biology with engineering platforms enhancing performance for applications interacting with natural environments such as vision and speech (Hasler and Marr

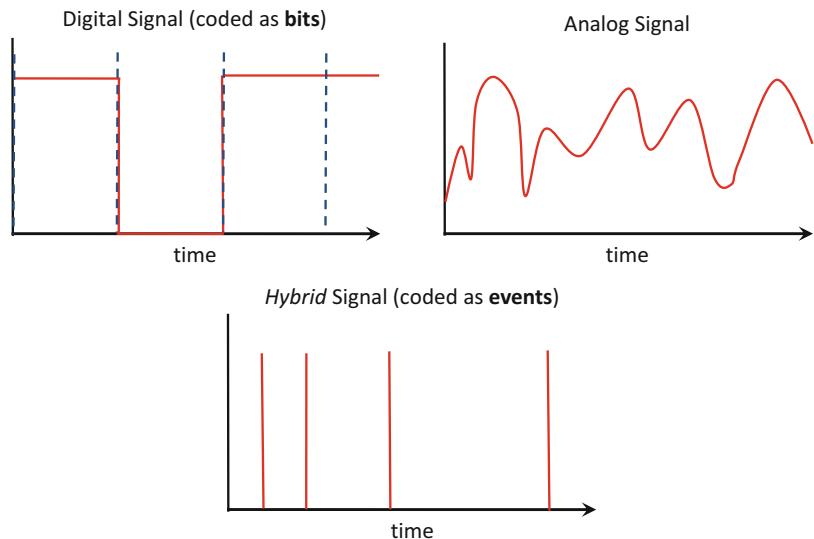
2013). These neural-inspired systems are exemplified by a set of computational principles, including hybrid analog-digital signal representations (discussed next), co-location of memory and processing, unsupervised statistical learning, and distributed representations of information.

## Technology Platforms

Information representation can have a profound effect on information processing. The *spiking* model found in biology is a sparse coding scheme recognized by the neuroscience community as a neural encoding strategy for information processing (Ostojic 2014; Paugam-Moisy and Bohte 2012; Kumar et al. 2010; Izhikevich 2003; Diesmann et al. 1999; Borst and Theunissen 1999) and has code-theoretic justifications (Sarpeshkar 1998; Thorpe et al. 2001; Maass et al. 2002). Spiking approaches promise extreme improvements in computational power efficiency (Hasler and Marr 2013) because they directly exploit the underlying physics of biology (Jaeger and Haas 2004; Sarpeshkar 1998; Maass 1997; Izhikivich 2007), analog electronics, or, in the present case, optoelectronics. Digital in amplitude but temporally analog, spiking naturally interleaves robust, discrete representations for communication with precise, continuous representations for computation in order to reap the benefits of both digital and analog. Spiking has two primary advantages over synchronous analog: (1) its analog variable (time) is less noisy than its digital variable (amplitude), and (2) it is asynchronous, without a global clock. Clock synchronization allows for time-division multiplexing (TDM); however, it is a significant practical problem in many-core systems (Mundy et al. 2015). These advantages may account for the ubiquity of spiking in natural processing systems (Thorpe et al. 2001). It is natural to deepen this distinction to include physical representational aspects, with the important result that optical carrier noise does not accumulate. As will be discussed, when an optical pulse is generated, it is transmitted and routed through a linear optical network with the help of its wavelength identifier (Fig. 3).

### Principles of Neuromorphic Photonics, Fig. 3

Spiking neural networks encode information as events in time rather than bits. Because the time at which a spike occurs is analog while its amplitude is digital, the signals use a mixed-signal or hybrid encoding scheme (Reproduced from Tait et al. 2014a). With permission of Springer)

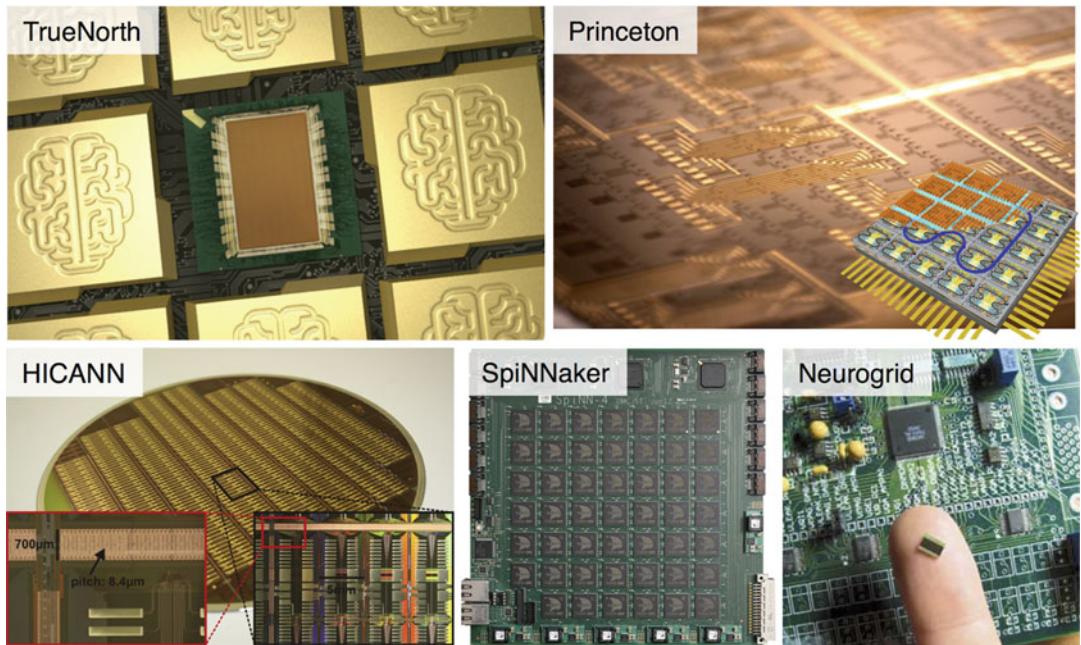


### Neuromorphic Microelectronics

Spiking primitives have been built in CMOS analog circuits, digital *neurosynaptic cores*, and non-CMOS devices. Various technologies (Fig. 4) have demonstrated large-scale spiking neural networks in electronics, including notably Neurogrid as part of Stanford University's Brains in Silicon program (Benjamin et al. 2014), IBM's TrueNorth as part of DARPA's SyNAPSE program (Merolla et al. 2014), HICANN as part of Heidelberg University's FACETS/BrainScaleS project (Schemmel et al. 2010), and University of Manchester's neuromorphic chip as part of the SpiNNaker project (Furber et al. 2014); the latter two are under the flagship of the European Commission's Human Brain Project (The HBP Report 2012). These spiking platforms promise potent advantages in efficiency, fault tolerance, and adaptability over von Neumann architectures to better interact with natural environments by applying the circuit and system principles of neuronal computation, including robust analog signaling, physics-based dynamics, distributed complexity, and learning. Using this neuromorphic hardware to process faster signals (e.g., radio waveforms) is, however, not a simple matter of accelerating the clock. These systems rely on slow timescale operation to accomplish dense interconnection.

Whereas von Neumann processors rely on point-to-point memory processor communication, a

neuromorphic processor typically requires a large number of interconnects (i.e.,  $\sim 100$  s of many-to-one fan-in per processor) (Hasler and Marr 2013). This requires a significant amount of multicasting, which creates a communication burden. This, in turn, introduces fundamental performance challenges that result from RC and radiative physics in electronic links, in addition to the typical bandwidth-distance-energy limits of point-to-point connections (Miller 2000). While some incorporate a dense mesh of wires overlaying the semiconductor substrate as cross-bar arrays, large-scale systems are ultimately forced to adopt some form of TDM or packet switching, notably, address-event representation (AER), which introduces the overhead of representing spike as digital codes instead of physical pulses. This abstraction at the architectural level allows virtual interconnectivities to exceed wire density by a factor related to the sacrificed bandwidth, which can be orders of magnitude (Boahen 2000). Spiking neural networks (SNNs) based on AER are thus effective at targeting biological timescales and the associated application space: real-time applications (object recognition) in the kHz regime (Merolla et al. 2014; Furber et al. 2014) and accelerated simulation in the low MHz regime (Schemmel et al. 2010). However, neuromorphic processing for high-bandwidth applications in



**Principles of Neuromorphic Photonics, Fig. 4** Selected pictures of five different neuromorphic hardware discussed here. They include TrueNorth (Merolla

et al. 2014), HICANN (Schemmel et al. 2010), SpiNNaker (Furber et al. 2014), Neurogrid (Benjamin et al. 2014)

the GHz regime (such as sensing and manipulating the radio spectrum and for hypersonic aircraft control) must take a fundamentally different approach to interconnection.

### Toward Neuromorphic Photonics

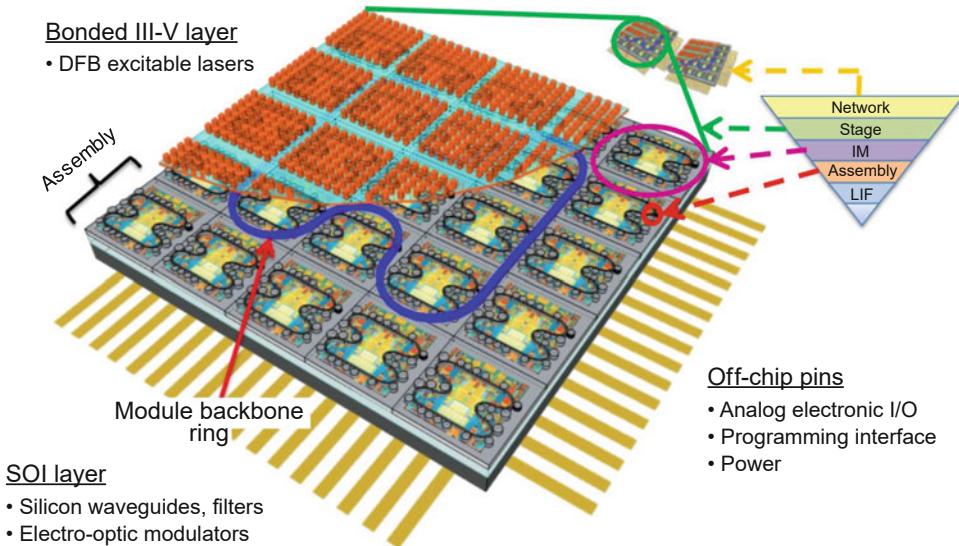
Just as optics and photonics are being employed for interconnection in conventional CPU systems, optical networking principles can be applied to the neuromorphic domain (Fig. 2 (right)). Mapping a processing paradigm to its underlying dynamics, than abstracting the physics away entirely, can significantly streamline efficiency and performance, and mapping a laser's behavior to a neuron's behavior relies on discovering formal mathematical analogies (i.e., isomorphisms) in their respective governing dynamics. Many of the physical processes underlying photonic devices have been shown to have a strong analogy with biological processing models, which can both be described within the framework of non-linear dynamics. Large-scale integrated photonic platforms (see Fig. 5) offer an opportunity for

ultrafast neuromorphic processing that complements neuromorphic microelectronics aimed at biological timescales. The high switching speeds, high communication bandwidth, and low cross talk achievable in photonics are very well-suited for an ultrafast spike-based information scheme with high interconnection densities (Prucnal et al. 2016; Tait et al. 2014b). The efforts in this budding research field aims to synergistically integrate the underlying physics of photonics with spiking neuron-based processing. *Neuromorphic photonics* represents a broad domain of applications where quick, temporally precise, and robust systems are necessary.

Later in this entry, we compare metrics between neuromorphic electronics and neuromorphic photonics.

### Neuromorphic Photonics

Photonics has revolutionized information transmission (communication and interconnects), while electronics, in parallel, has dominated information



**Principles of Neuromorphic Photonics, Fig. 5** Conceptual rendering of a photonic neuromorphic processor. Laser arrays (orange layer) implement pulsed (spiking) dynamics with electro-optic physics, and a photonic

network on-chip (blue and gray) supports complex structures of virtual interconnection among these elements, while electronic circuitry (yellow) controls stability, self-healing, and learning

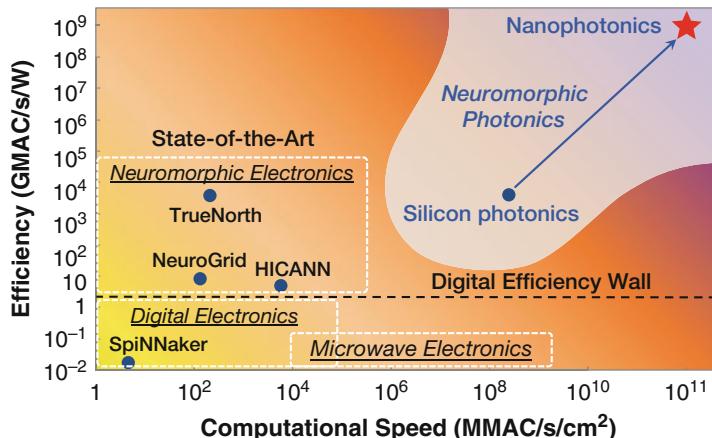
transformation (computations). This leads naturally to the following question: how can the unifying of the boundaries between the two be made as effective as possible? (Keyes 1985; Tucker 2010; Caulfield and Dolev 2010). CMOS gates only draw energy from the rail when and where called upon; however, the energy required to drive an interconnect from one gate to the next dominates CMOS circuit energy use. Relaying a signal from gate to gate, especially using a clocked scheme, induces penalties in latency and bandwidth compared to an optical waveguide passively carrying multiplexed signals.

This suggests that starting up a new architecture from a photonic interconnection fabric supporting nonlinear optoelectronic devices can be uniquely advantageous in terms of energy efficiency, bandwidth, and latency, sidestepping many of the fundamental trade-offs in digital and analog electronics. It may be one of the few practical ways to achieve ultrafast, complex on-chip processing without consuming impractical amounts of power (Prucnal and Shastri 2017).

Complex photonic systems have been largely unexplored due to the absence of a robust photonic integration industry. Recently, however, the

landscape for manufacturable photonic chips has been changing rapidly and now promises to achieve economies of scale previously enjoyed solely by microelectronics. In particular, a new photonic manufacturing hybrid platform that combines in the same chip both active (e.g., lasers and detectors) and passive elements (e.g., waveguides, resonators, modulators) is emerging (Fang et al. 2007; Liang et al. 2010; Liang and Bowers 2010a; Roelkens et al. 2010; Heck et al. 2013). A neuromorphic photonic approach based on this platform could potentially operate six to eight orders of magnitude faster than neuromorphic electronics when accounting for the bandwidth reduction of virtualizing interconnects (Prucnal and Shastri 2017) (cf. Fig. 6; also see Table 1 and related discussion).

In the past, the communication potentials of optical interconnects have received attention for neural networking; however, attempts to realize holographic or matrix-vector multiplication systems have failed to outperform mainstream electronics at relevant problems in computing, which can perhaps be attributed to the immaturity of large-scale integration technologies and manufacturing economics.



**Principles of Neuromorphic Photonics, Fig. 6** Comparison of neuromorphic hardware platforms. Neuromorphic photonic architectures potentially sport better speed-to-efficiency characteristics than state-of-the-art electronic neural nets (such as IBM's TrueNorth, Stanford University's Neurogrid, Heidelberg University's HICANN), as well as advanced digital electronic systems (such as the University of Manchester's SpiNNaker). On

the top right: the photonic neuron platforms studied in Prucnal and Shastri (2017). The regions highlighted in the graph are approximate, based on qualitative trade-offs of each technology (Adapted from Ferreira de Lima et al. 2017). Licensed under Creative Commons Attribution-NonCommercial-NoDerivatives License (CC BY-NC-ND))

**Principles of Neuromorphic Photonics, Table 1** Comparison between different neuromorphic processors

Chip	MAC rate/processor <sup>a</sup>	Energy/MAC (pJ) <sup>b</sup>	Processor fan-in	Area/MAC ( $\mu\text{m}^2$ ) <sup>c</sup>	Synapse precision (bit)
Photonic hybrid III-V/Si <sup>d</sup>	20 GHz	0.26	108	205	5.1
Sub- $\lambda$ photonics <sup>e</sup> (future trend)	200 GHz	0.0007	$\sim 200$	20	8
HICANN (Schemmel et al. 2010)	22.4 MHz	198.4	224	780	4
TrueNorth (Merolla et al. 2014)	2.5 kHz	0.27	256	4.9	5
Neurogrid (Benjamin et al. 2014)	40.1 kHz	119	4096	7.1	13
SpiNNaker <sup>f</sup> (Furber et al. 2014)	3.2 kHz	$6 \times 10^5$	320	217	16

<sup>a</sup>A MAC event occurs each time a spike is integrated by the neuron. Neuron fan-in refers to the number of possible connections to a single neuron

<sup>b</sup>The energy per MAC for HICANN, TrueNorth, Neurogrid, and SpiNNaker were estimated by dividing wall-plug power to number of neurons and to operational MAC rate per processor

<sup>c</sup>The area per MAC was estimated by dividing the chip/board size to the number of MAC units (neuron count times fan-in). All numbers therefore include overheads in terms of footprint and area

<sup>d</sup>III-V/Si hybrid stands for estimated metrics of a spiking neural network in a photonic integrated circuit in a III-V/Si hybrid platform

<sup>e</sup>Sub- $\lambda$  stands for estimated metrics for a platform using optimized sub-wavelength structures, such as photonic crystals

<sup>f</sup>Neurons, synapses, and spikes are digitally encoded in event headers that travel around co-integrated processor cores. So all numbers here are based on a typical application example

Techniques in silicon photonic integrated circuit (PIC) fabrication are driven by a tremendous demand for optical interconnects within

conventional digital computing systems (Smit et al. 2012; Jalali and Fathpour 2006), which means platforms for systems integration of active

photronics are becoming commercial reality (Liang et al. 2010; Roelkens et al. 2010; Heck et al. 2013; Liang and Bowers 2010b; Marpaung et al. 2013). The potential of recent advances in integrated photonics to enable unconventional computing has not yet been investigated. The theme of current research has been on how modern PIC platforms can topple historic technological barriers between large-scale analog networks and photonic neural systems. In this context, there are two complementary areas of investigation by the research community, namely, photonic spike processing and photonic reservoir computing. While the scope of this entry is limited to the former, we briefly introduce both.

# Photonic Spike Processing

An investigation of photonics for information processing based on spikes has taken place alongside the development of electronic spiking architectures. Since the first demonstration of photonic spike processing by Rosenbluth et al. (Rosenbluth et al. 2009), there has been a surge in research related to aspects of spike processing in various photonic devices with a recent bloom of proposed forms of spiking dynamics (Tait et al. 2014b; Kelleher et al. 2010; Kravtsov et al. 2011; Fok et al. 2011; Coomans et al. 2011; Brunstein et al. 2012; Nahmias et al. 2013; Van Vaerenbergh et al. 2013; Aragoneses et al. 2014; Selmi et al. 2014; Hurtado and Javaloyes 2015; Garbin et al. 2015; Shastri et al. 2016; Romeira et al. 2016) – a

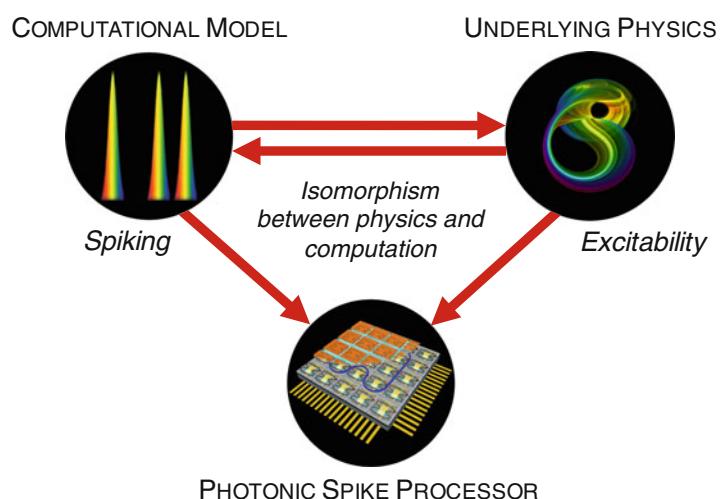
strategy that could lead to combined computation and communication in the same substrate.

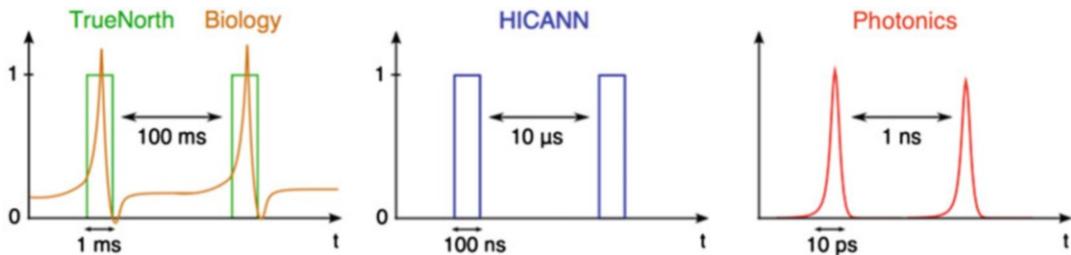
We recently (Prucnal et al. 2016; Prucnal and Shastri 2017) reviewed the recent surge of interest (Tait et al. 2014a; Kelleher et al. 2010; Coomans et al. 2011; Brunstein et al. 2012; Nahmias et al. 2013, 2015; Van Vaerenbergh et al. 2012, 2013; Aragoneses et al. 2014; Selmi et al. 2014; Hurtado and Javaloyes 2015; Garbin et al. 2015; Shastri et al. 2016; Yacomotti et al. 2006a; Goulding et al. 2007; Hurtado et al. 2010, 2012; Coomans 2012; Romeira et al. 2013, 2016; Sorrentino et al. 2015) in the information processing abilities of semiconductor devices that exploit the dynamical isomorphism between semiconductor photocarriers and neuron biophysics. Many of these proposals for “photonic neurons” or “laser neurons” or “optical neurons” for spike processing are based on lasers operating in an *excitable* regime (Fig. 7). Excitability (Hodgkin and Huxley 1952; Krauskopf et al. 2003) is a dynamical system property underlying all-or-none responses.

The difference in physical timescales allows these laser systems to exhibit these properties, except many orders of magnitude faster than their biological counterparts (Nahmias et al. 2013); the temporal resolution (tied to spike widths) and processing speed (tied to refractory period) are accelerated by factors nearing 100 million (Fig. 8). A network of photonic neurons could open computational domains that demand unprecedented temporal precision, power efficiency, and

# Principles of Neuromorphic Photonics,

**Fig. 7** Analogies between spike processing and photonics can be exploited to create a computational paradigm that performs beyond the sum of its parts. By reducing the abstraction between process (spiking) and physics (excitability), there could be a significant advantage on speed, energy usage, scalability (Adapted with permission from Prucnal et al. 2016). Copyright 2016 Optical Society of America



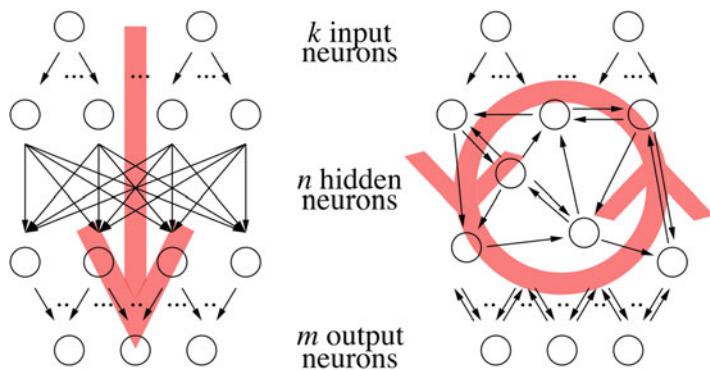


**Principles of Neuromorphic Photonics, Fig. 8** Difference in spike processing timescales (pulse width and refractory period) between biological neurons (left),

electronic spiking neurons (middle), and photonic neurons (right) (Reproduced with permission from Prucnal et al. (2016). Copyright 2016 Optical Society of America)

### Principles of Neuromorphic Photonics,

**Fig. 9** Comparison of the architectures of a feedforward (left-hand side) with a recurrent neural network (right-hand side); the gray arrows sketch the possible direction of computation (Adapted from Burgsteiner (2005))



functional complexity, potentially including applications in wideband radio-frequency (RF) processing, adaptive control of multi-antenna systems, and high-performance scientific computing.

### Photonic Reservoir Computing

Reservoir computing (RC) is another promising approach for neuro-inspired computing. A central tenet of RC is that complex processes are generated in a medium whose behavior is not necessarily understood theoretically. Instead, the “reservoir” (a fixed, recurrent network of non-linear nodes; see Fig. 9) generates a large number of complex processes, and linear combinations of reservoir signals are trained to approximate a desired task (Verstraeten et al. 2007). To arrive at a user-defined behavior, reservoirs do not need to model or program and instead rely on supervised machine learning techniques for simple linear classifiers. This is advantageous in systems whose overall behavior is complex, yet difficult to model or correspond to a theoretical behavior.

There are a wide class of physical systems that fit that description, and the reservoir concept makes them highly likely to apply to widespread information processing tasks (Soriano et al. 2015).

Over the past several years, reservoir computers have been constructed that exploit the incredible bandwidths and speeds available to photonic signals. These “photonic reservoirs” utilize optical multiplexing strategies to form highly complex virtual networks. Photonic RC particularly is attractive when the information to be processed is already in the optical domain, for example, applications in telecommunications and image processing. Recently, there has been a significant development in the hardware realization of RC. Optical reservoirs have been demonstrated with various schemes such as benchtop demonstrations with a fiber with a single nonlinear dynamical node (Brunner et al. 2013a, b; Paquot et al. 2012; Martinenghi et al. 2012; Larger et al. 2012; Duport et al. 2012, 2016; Hicke et al. 2013; Soriano et al. 2013; Ortín et al. 2015), and integrated solutions including microring

resonators (Mesaritakis et al. 2013), a network of coupled semiconductor optical amplifiers (SOAs) (Vandoorne et al. 2011), and a passive silicon photonic chip (Vandoorne et al. 2014).

It has been experimentally demonstrated and verified that some of these photonic RC solutions achieve highly competitive figures of merit at unprecedented data rates often outperforming software-based machine learning techniques for computationally hard tasks such as spoken digit and speaker recognition, chaotic time series prediction, signal classification, or dynamical system modeling. Another significant advantage of photonic-based approaches, as pointed out by Vandoorne et al. (Vandoorne et al. 2014), is the straightforward use of coherent light to exploit both the phase and amplitude of light. The simultaneous exploitation of two physical quantities results in a notable improvement over real-valued networks that are traditionally used in software-based RC – a reservoir operating on complex numbers in essence doubles the internal degrees of freedom in the system, leading to a reservoir size that is roughly twice as large as the same device operated with incoherent light.

Neuromorphic spike processing and reservoir approaches differ fundamentally and possess complementary advantages. Both derive a broad repertoire of behaviors (often referred to as complexity) from a large number of physical degrees of freedom (e.g., intensities) coupled through interaction parameters (e.g., transmissions). Both offer means of selecting a specific, desired behavior from this repertoire using controllable parameters. In neuromorphic systems, network weights are *both* the interaction and controllable parameters, whereas, in reservoir computers, these two groups of parameters are separate. This distinction has two major implications. Firstly, the interaction parameters of a reservoir do not need to be observable or even repeatable from system to system. Reservoirs can thus derive complexity from physical processes that are difficult to model or reproduce. Furthermore, they do not require significant hardware to control the state of the reservoir. Neuromorphic hardware has a burden to correspond physical parameters (e.g., drive voltages) to model parameters (e.g., weights). Secondly,

reservoir computers can only be made to elicit a desired behavior through instance-specific supervised training, whereas neuromorphic computers can be programmed a priori using a known set of weights. Because neuromorphic behavior is determined only by controllable parameters, these parameters can be mapped directly between different system instances, different types of neuromorphic systems, and simulations. Neuromorphic hardware can leverage existing algorithms (e.g., Neural Engineering Framework (NEF) (Stewart and Eliasmith 2014)), map virtual training results to hardware, and particular behaviors are guaranteed to occur. Photonic RCs can of course be simulated; however, they have no corresponding guarantee that a particular hardware instance will reproduce a simulated behavior or that training will be able to converge to this behavior.

## Challenges for Integrated Neuromorphic Photonics

Key criteria for nonlinear elements to enable a scalable computing platform include thresholding, fan-in, and cascability (Keyes 1985; Tucker 2010; Caulfield and Dolev 2010). Past approaches to optical computing have met challenges realizing these criteria. A variety of digital logic gates in photonics that suppress amplitude noise accumulation have been claimed, but many proposed optical logic devices do not meet necessary conditions of cascability. Analog photonic processing has found application in high-bandwidth filtering of microwave signals (Capmany et al. 2006), but the accumulation of phase noise, in addition to amplitude noise, limits the ultimate size and complexity of such systems.

Recent investigations (Prucnal and Shastri 2017; Tait et al. 2014b; Coomans et al. 2011; Brunstein et al. 2012; Nahmias et al. 2013, 2015; Van Vaerenbergh et al. 2013; Aragoneses et al. 2014; Selmi et al. 2014; Hurtado and Javaloyes 2015; Garbin et al. 2015; Shastri et al. 2016; Romeira et al. 2016) have suggested that an alternative approach to exploit the high bandwidth of photonic devices for computing lies not in increasing device performance or fabrication, but instead in examining models of computation that

hybridize techniques from analog and digital processing. These investigations have concluded that a photonic neuromorphic processor could satisfy them by implementing a model of a neuron, i.e., a photonic neuron, as opposed to the model of a logic gate. Early work in neuromorphic photonics involved fiber-based spiking approaches for learning, pattern recognition, and feedback (Rosenbluth et al. 2009; Kravtsov et al. 2011; Fok et al. 2011). Spiking behavior resulted from a combination of SOA together with a highly nonlinear fiber thresholding, but they were neither excitable nor asynchronous and therefore not suitable for scalable, distributed processing in networks.

“Neuromorphism” implies a strict isomorphism between artificial neural networks and optoelectronic devices. There are two research challenges necessary to establish this isomorphism: the nonlinearity (equivalent to thresholding) in individual neurons and the synaptic interconnection (related to fan-in and cascability) between different neurons, as will be discussed in the proceeding sections. Once the isomorphism is established and large networks are fabricated, we anticipate that the computational neuroscience and software engineering will have a new optimized processor for which they can adapt their methods and algorithms.

Photonic neurons address the traditional problem of noise accumulation by interleaving physical representations of information. Representational interleaving, in which a signal is repeatedly transformed between coding schemes (digital-analog) and physical variables (electronic-optical), can grant many advantages to computation and noise properties. From an engineering standpoint, the logical function of a nonlinear neuron can be thought of as increasing signal-to-noise ratio (SNR) that tends to degrade in linear systems, whether that means a continuous nonlinear transfer function suppressing analog noise or spiking dynamics curtailing pulse attenuation and spreading. As a result, we neglect purely linear PNNs as they do not offer mechanisms to maintain signal fidelity in a large network in the presence of noise.

The optical channel is highly expressive and correspondingly very sensitive to phase and frequency noise. For example, the networking

architecture proposed in Tait et al. (2014b) relies on wavelength-division multiplexing (WDM) for interconnecting many points in a photonic substrate together. Any proposal for networking computational primitives must address the issue of practical cascability: transferring information and energy in the optical domain from one neuron to many others and exciting them with the same strength without being sensitive to noise. This is notably achieved, for example, by encoding information in energy pulses that can trigger stereotypical excitation in other neurons regardless of their analog amplitude. In addition, as will be discussed next, schemes which exhibit limitations with regard to wavelength channels may require a large number of wavelength conversion steps, which can be costly, noisy, and inefficient.

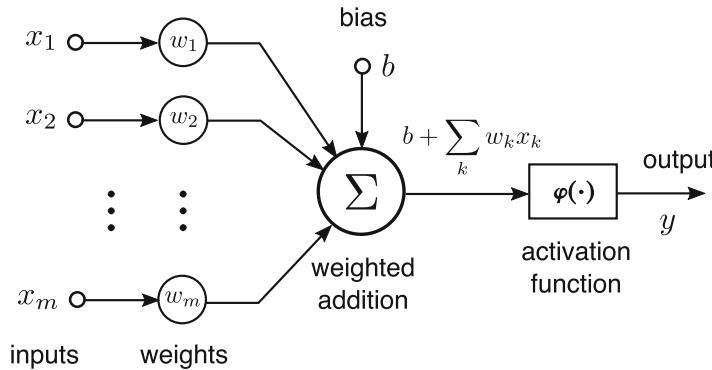
## Photonic Neuron

### What Is an Artificial Neuron?

Neuroscientists research artificial neural networks as an attempt to mimic the *natural processing* capabilities of the brain. These networks of simple nonlinear nodes can be taught (rather than programmed) and reconfigured to best execute a desired task; this is called *learning*. Today, neural nets offer state-of-the-art algorithms for machine intelligence such as speech recognition, natural language processing, and machine vision (Bengio et al. 2013).

Three elements constitute a neural network: a set of nonlinear nodes (neurons), configurable interconnection (network), and information representation (coding scheme). An elementary illustration of a neuron is shown in Fig. 10. The network consists of a weighted directed graph, in which connections are called synapses. The input of a neuron is a linear combination (or weighted addition) of the outputs of the neurons connected to it. Then, the particular neuron integrates the combined signal and produces a nonlinear response, represented by an *activation function*, usually monotonic and bounded.

Three generations of neural networks were historically studied in computational neuroscience (Maass 1997). The first was based on the McCulloch-Pitts neural model, which consists of a



**Principles of Neuromorphic Photonics, Fig. 10** Non-linear model of a neuron. Note the three parts: (i) a set of synapses, or connecting links; (ii) an adder, or linear combiner, performing weighted addition; and (iii) a nonlinear

activation function (Reproduced with permission from Prucnal et al. (2016). Copyright 2016 Optical Society of America)

linear combiner followed by a steplike activation function (binary output). These neural networks are Boolean complete – i.e., they have the ability of simulating any Boolean circuit and are said to be universal for digital computations. The second generation implemented analog outputs, with a continuous activation function instead of a hard threshold. Neural networks of the second generation are universal for analog computations in the sense they can uniformly approximate arbitrarily well any continuous function with a compact domain (Maass 1997). When augmented with the notion of “time,” recurrent connections can be created and be exploited to create attractor states (Eliasmith 2005) and associative memory (Hopfield 1982) in the network.

Physiological neurons communicate with each other using pulses called action potentials or spikes. In traditional neural network models, an analog variable is used to represent the firing rate of these spikes. This coding scheme called *rate coding* was believed to be a major, if not the only, coding scheme used in biology. Surprisingly, there are some fast analog computations in the visual cortex that cannot possibly be explained by rate coding. For example, neuroscientists demonstrated in the 1990s that a single cortical area in macaque monkey is capable of analyzing and classifying visual patterns in just 30 ms, in spite of the fact that these neurons’ firing rates are usually below 100 Hz – i.e., less than three spikes in 30 ms

(Thorpe et al. 2001; Maass 1997; Perrett et al. 1982) which directly challenges the assumptions of rate coding. In parallel, more evidence was found that biological neurons use the precise timing of these spikes to encode information, which led to the investigation of a third generation of neural networks based on a *spiking neuron*.

The simplicity of the models of the previous generations precluded the investigation of the possibilities of using *time* as resource for computation and communication. If the *timing* of individual spikes itself carry analog information (*temporal coding*), then the energy necessary to create such spike is optimally employed to express information. Furthermore, Maass et al. showed that this third generation is a generalization of the first two and, for several concrete examples, can emulate real-valued neural network models while being more robust to noise (Maass 1997).

For example, one of the simplest models of a spiking neuron is called *leaky integrate-and-fire* (LIF), described in Eq. 1. It represents a simplified circuit model of the membrane potential of a biological spiking neuron.

$$C_m \frac{dV_m(t)}{dt} = \frac{1}{R_m} (V_m(t) - V_L) + I_{app}(t);$$

if  $V_m(t) > V_{\text{thresh}}$

then release a spike and set  $V_m(t) \rightarrow V_{\text{reset}}$

(1)

where  $V_m(t)$  is the membrane voltage,  $R_m$  the membrane resistance,  $V_L$  the equilibrium potential, and  $I_{\text{app}}$  the applied current (input). More bio-realistic models, such as the Hodgkin-Huxley model, involve several ordinary differential equations and nonlinear functions.

However, simply simulating neural networks on a conventional computer, be it of any generation, is costly because of the fundamentally serial nature of CPU architectures. Bio-realistic SNN present a particular challenge because of the need for fine-grained time discretization (Izhikevich 2004). Engineers circumvent this challenge by employing an event-driven simulation model which resolves this issue by storing the time and shape of the events expanded in a suitable basis in a simulation queue. Although simplified models do not faithfully reproduce key properties of cortical spiking neurons, it allows for large-scale simulations of SNNs, from which key networking properties can be extracted.

Alternatively, one can build an unconventional, distributed network of nonlinear nodes, which directly use the physics of nonlinear devices or excitable dynamical systems, significantly dropping energetic cost per bit.

Here, we discuss recent advances in neuromorphic photonic hardware and the constraints to which particular implementations must subject, including accuracy, noise, cascability, and thresholding. A successful architecture must tolerate eventual inaccuracies and noise, indefinite propagation of signals, and provide mechanisms to counteract noise accumulation as the signal traverses across the network.

### Basic Requirements for a Photonic Neuron

An artificial neuron described in Fig. 10 must perform three basic mathematical operations: array multiplication (weighting), summation, and a nonlinear transformation (activation function). Moreover, the inputs to be weighted in the first stage must be of the same nature of the output – in the case considered here, photons.

As the size of the network grows, additional mechanisms are required at the hardware level to ensure the integrity of the signals. The neuron must have a scalable number of inputs, referred

to as *maximum fan-in* ( $N_f$ ), which will determine the degree of connectivity of the network. Each neuron's output power must be strong enough to drive at least  $N_f$  others (*cascadability*). This concept is tied closely with that of *thresholding*: the SNR at the output must be higher than at its input. Cascadability, thresholding, and fan-in are particularly challenging to optical systems due to quantum efficiency (photons have finite supply) and amplified spontaneous emission (ASE) noise, which degrades SNR.

### The Processing Network Node

A networkable photonic device with optical I/O, provided that it is capable of emulating an artificial neuron, is named a processing network node (PNN) (Tait et al. 2014b). Formulations of a photonic PNN can be divided into two main categories: all-optical and optical-electrical-optical (O/E/O), respectively, classified according to whether the information is always embedded in the optical domain or switches from optical to electrical and back. We note that the term *all-optical* is sometimes very loosely defined in engineering articles. Physicists reserve it for devices that rely on parametric nonlinear processes, such as four-wave mixing. Here, our definition includes devices that undergo nonparametric processes as well, such as semiconductor lasers with optical feedback, in which optical pulses directly perturb the carrier population, triggering quick energy exchanges with the cavity field that results in the release of another optical pulse.

Silicon waveguides have a relatively enormous transparency window of 7.5 THz (Agrawal 2002) over which they can guide lightwaves with very low attenuation and cross talk, in contrast with electrical wires or radio-frequency transmission lines. With WDM, each input signal exists at a different wavelength but is superimposed with other signals onto the same waveguide. For example, to maximize the information throughput, a single waveguide could carry hundreds of wide-band signals (~20 GHz) simultaneously. As such, it is highly desirable and crucial to design a PNN that is compatible with WDM. All-optical versions of a PNN must have some way to sum multiwavelength signals, and this requires a

population of charge carriers. On the other hand, O/E/O versions could make use of photodetectors (PD) to provide a spatial sum of WDM signals. The PD output could drive an E/O converter, involving a laser or a modulator, whose optical output is a nonlinear result of the electrical input. Instances of both techniques are presented in the next section.

### All-Optical PNNs

Coherent injection models are characterized by input signals directly interacting with cavity modes, such that outputs are at the same wavelength as inputs (Fig. 11a). Since coherent optical systems operate at a single wavelength  $\lambda$ , the signals lack distinguishability from one another in a WDM-encoded framework. As demonstrated in Alexander et al. (2013), the effective weight of coherently injected inputs is also strongly phase dependent. Global optical phase control presents a challenge in synchronized laser systems but also affords an extra degree of freedom to configure weight values.

Incoherent injection models inject light in a wavelength  $\lambda_j$  to selectively modulate an intracavity property that then triggers excitable output pulses in an output wavelength  $\lambda_i$  (Fig. 11b). A number of approaches (Nahmias

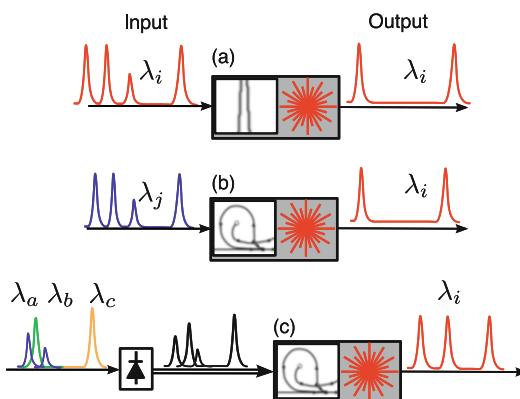
et al. 2013; Selmi et al. 2014, 2015; Hurtado and Javaloyes 2015) – including those based on optical pumping – fall under this category. While distinct, the output wavelength often has a stringent relationship with the input wavelength. For example, excitable micropillar lasers (Selmi et al. 2014; Barbay et al. 2011) are carefully designed to support one input mode with a node coincident with an antinode of the lasing mode. In cases where the input is also used as a pump (Shastri et al. 2016), the input wavelength must be shorter than that of the output in order to achieve carrier population inversion.

WDM networking introduces wavelength constraints that conflict with the ones inherent to optical injection. One approach for networking optically injected devices is to attempt to separate these wavelength constraints. In early work on neuromorphic photonics in fiber, this was accomplished with charge-carrier-mediated cross-gain modulation (XGM) in an SOA (Rosenbluth et al. 2009; Kravtsov et al. 2011; Fok et al. 2011).

### O/E/O PNNs

In this kind of PNN, the O/E subcircuit is responsible for the weighted addition functionality, whereas the E/O is responsible for the nonlinearity (Fig. 11c). Each subcircuit can therefore be analyzed independently. The analysis of an O/E WDM weighted addition circuit is deferred to a later section (photonic neural networks).

The E/O subcircuit of the PNN must take an electronic input representing the complementary weighted sum of optical inputs, perform some dynamical or nonlinear process, and generate a clean optical output on a single wavelength. Figure 12 classifies six different ways nonlinearities can be implemented in an E/O circuit. The type of nonlinearity, corresponding to different neural models, is separated into *dynamical systems* and *continuous nonlinearities*, both of which have a single input  $u$  and output  $y$ . A continuous nonlinearity is described by a differential equation  $\dot{y} = f(y, u)$ . This includes continuous-time recurrent neural networks (CTRNNs) such as Hopfield networks. The derivative of  $y$  introduces a sense of time, which is required to consider recurrent networking, although it does not exclude



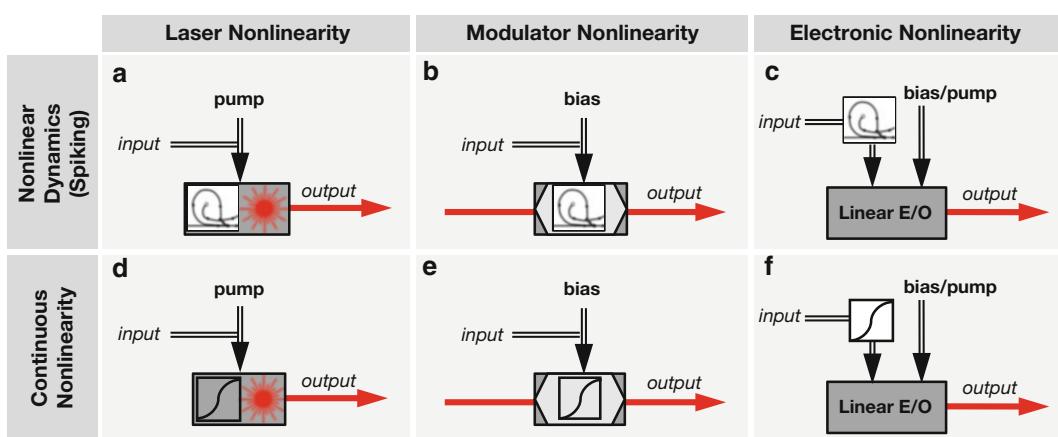
**Principles of Neuromorphic Photonics, Fig. 11** General classification of semiconductor excitable lasers based on (a) coherent optical injection, (b) noncoherent optical injection, and (c) full electrical injection. Each of these lasers can be pumped either electrically or optically (Reproduced from Ferreira de Lima et al. (2017). Licensed under Creative Commons Attribution-NonCommercial-NoDerivatives License. (CC BY-NC-ND))

feedforward models where time plays no role, such as perceptron models. A dynamical system has an internal state  $\vec{x}$  and is described by  $\dot{x} = g(\vec{x}, u)$ ;  $\dot{y} = h(\vec{x}, y, u)$ , where the second differential equation represents the mapping between the internal state  $\vec{x}$  and the output  $y$ . There are a wide variety of spiking models based on excitability, threshold behavior, relaxation oscillations, etc. covered, for example, in Izhikivich (2007).

Physical implementations of these nonlinearities can arise from devices falling into roughly three categories: pure electronics, electro-optic physics in modulators, and active laser behavior (Fig. 12). Figure 12a illustrates spiking lasers, which are detailed in the next section and offer perhaps the most promise in terms of garnering the full advantage of recent theoretical results on spike processing efficiency and expressiveness. Figure 12b is a spiking modulator. The work in Van Vaerenbergh et al. (2012) might be adapted to fit this classification; however, to the authors' knowledge, an ultrafast spiking modulator remains to be experimentally demonstrated. Figure 12c illustrates a purely electronic approach to nonlinear neural behavior. Linear E/O could be

done by either a modulator or directly driven laser. This class could encompass interesting intersections with efficient analog electronic neurons in silicon (Indiveri et al. 2011; Pickett et al. 2013). A limitation of these approaches is the need to operate slow enough to digitize outputs into a form suitable for electronic TDM and/or AER routing.

Figure 12d describes a laser with continuous nonlinearity, an instantiation of which was recently demonstrated in Nahmias et al. (2016). Figure 12e shows a modulator with continuous nonlinearity; the first demonstration of which in a PNN and recurrent network is presented in Tait et al. (2017). The pros and cons between the schemes in Fig. 12d, e are the same ones brought up by the on-chip versus off-chip light source debate, currently underway in the silicon photonic community. On-chip sources could provide advantageous energy scaling (Heck and Bowers 2014), but they require the introduction of exotic materials to the silicon photonic process to provide optical gain. Active research in this area has the goal of making source co-integration feasible (Liang and Bowers 2010a; Roelkens et al. 2010). An opposing school of thought argues that on-chip sources are still a nascent technology



**Principles of Neuromorphic Photonics, Fig. 12** Classification of O/E/O PNN nonlinearities and possible implementations. (a) Spiking laser neuron. (b) Spiking modulator. (c) Spiking or arbitrary electronic system driving a linear electro-optic (E/O) transducer – either modulator or laser. (d) Overdriven continuous laser neuron, as demonstrated in Nahmias et al. (2016).

(e) Continuous modulator neuron, as demonstrated in Tait et al. (2017). (f) Continuous purely electronic nonlinearity with optical output (Reproduced from Ferreira de Lima et al. 2017). Licensed under Creative Commons Attribution-NonCommercial-NoDerivatives License. (CC BY-NC-ND)

(Vlasov 2012). While fiber-to-chip coupling presents practical issues (Barwicz et al. 2016), discrete laser sources are cheap and well understood. Furthermore, on-chip lasers dissipate large amounts of power (Sysak et al. 2011), the full implications of which may complicate system design (Vlasov 2012). In either case, the conception of a PNN module, consisting of a photonic weight bank, detector, and E/O converter, as a participant in a broadcast-and-weight network, could be applied to a broad array of neuron models and technological implementations.

Both discussed all-optical and O/E/O PNN approaches depend on charge-carrier dynamics, whose lifetime eventually limits the bandwidth of the summation operation. The O/E/O strategy, however, has a few advantages: it can be modularized; it uses more standard optoelectronic components; and it is more amenable to integration. Therefore here we give more attention to this strategy. Moreover, although the E/O part of the PNN can involve any kind of nonlinearity (Fig. 12), not necessarily spiking, we are focusing on spiking behavior because of its interesting noise resistance and richness of representation. As such, we study here excitable semiconductor laser physics with the objective of directly producing optical spikes.

In this light, the PNN could be separated into three parts, just like the artificial neuron: weighting, addition, and neural behavior. Weighting and adding define how nonlinear nodes can be *networked* together, whereas the neural behavior dictates the *activation function* shown in Fig. 10. In the next section, we review recent developments of semiconductor excitable lasers that emulate spiking neural behavior, and, following that, we discuss a scalable WDM networking scheme.

## Excitable/Spiking Lasers

In the past few years, there has been a bloom of optoelectronic devices exhibiting excitable dynamics isomorphic to a physiological neuron. Excitable systems can be roughly defined by three criteria: (a) there is only one stable state at which

the system can indefinitely stay at rest; (b) when excited above a certain threshold, the system undergoes a stereotypical excursion, emitting a *spike*; and (c) after the excursion, the system decays back to rest in the course of a *refractory period* during which it is temporarily less likely to emit another spike.

### Analogy to Leaky Integrate-and-Fire Model

Excitable behavior can be realized near the threshold of a passively Q-switched two-section laser with saturable absorber (SA). Figure 13a, b shows an example of integrated design in a hybrid photonic platform. This device comprises a III–V epitaxial structure with multiple quantum well (MQW) region (the gain region) bonded to a low-loss silicon rib waveguide that rests on a silicon-on-insulator (SOI) substrate with sandwiched layers of graphene acting as a saturable absorber region. The laser emits light along the waveguide structure into a passive silicon network. Figure 13c–e shows experimental data from a fiber ring laser prototype, demonstrating key properties of excitability.

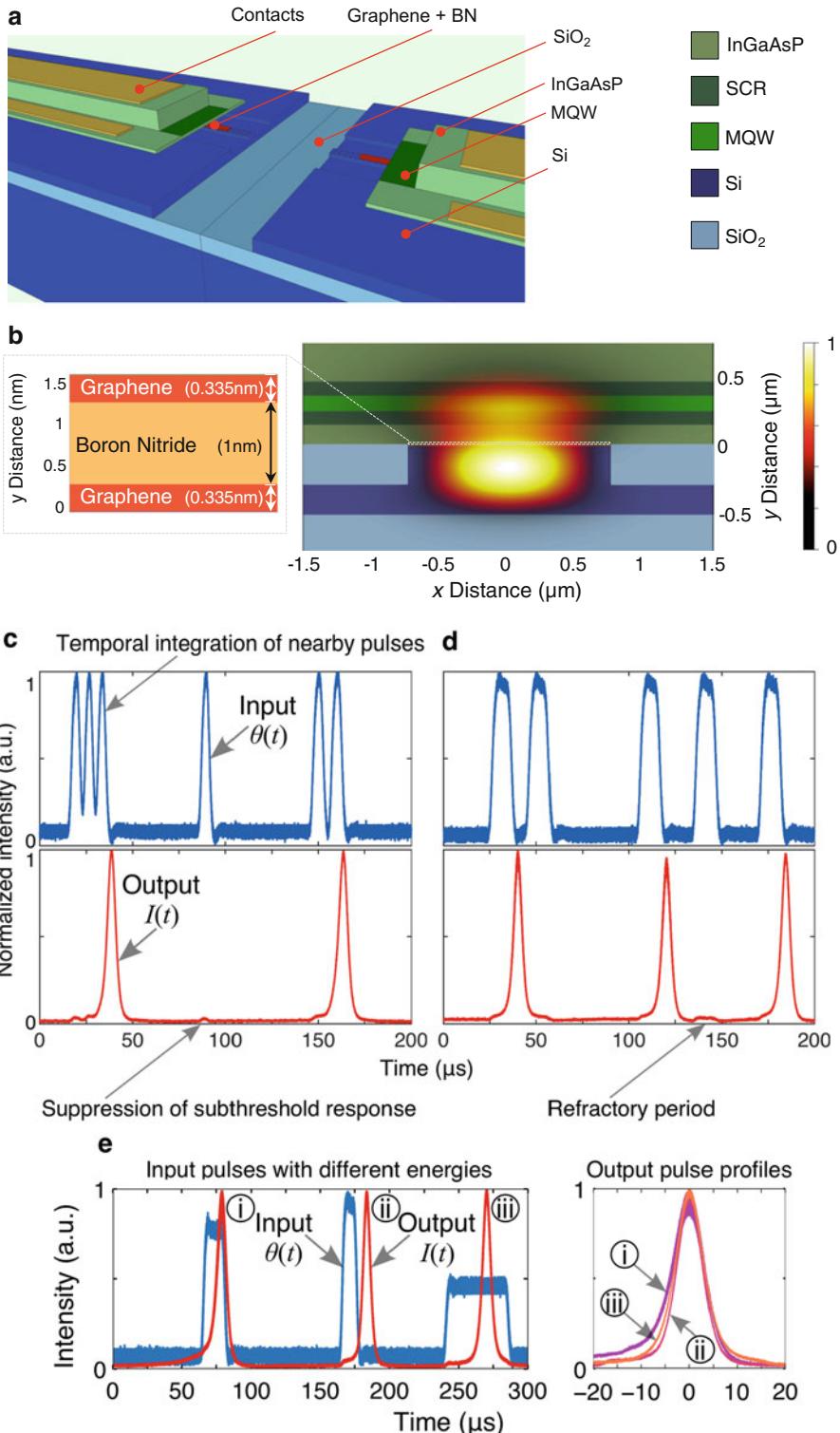
In general, the dynamics of a two-section laser composed of a gain section and a saturable absorber (SA) can be described by the Yamada model (Eqs. 2, 3, and 4) (Yamada 1993). This 3D dynamical system, in its simplest form, can be described with the following undimensionalized equations (Nahmias et al. 2013; Barbay et al. 2011):

$$\frac{dG(t)}{dt} = \gamma_G[A - G(t) - G(t)I(t)] + \theta(t) \quad (2)$$

$$\frac{dQ(t)}{dt} = \gamma_Q[B - Q(t) - aQ(t)I(t)] \quad (3)$$

$$\frac{dI(t)}{dt} = \gamma_I[G(t) - Q(t) - 1]I(t) + ef(G), \quad (4)$$

where  $G(t)$  models the gain,  $Q(t)$  is the absorption,  $I(t)$  is the laser intensity,  $A$  is the bias current of the gain region,  $B$  is the level of absorption,  $a$  describes the differential absorption relative to the differential gain,  $\gamma_G$  is the relaxation rate of the gain,  $\gamma_Q$  is the relaxation rate of the absorber,  $\gamma_I$  is the inverse photon lifetime,  $\theta(t)$  is the



Principles of Neuromorphic Photonics, Fig. 13 (continued)

time-dependent input perturbations, and  $\epsilon f(G)$  is the spontaneous noise contribution to intensity;  $\epsilon$  is a small coefficient.

In simple terms, if we assume electrical pumping at the gain section, the input perturbations are integrated by the gain section according to Eq. 2. An SA effectively becomes transparent as the light intensity builds up in the cavity and bleaches its carriers. It was shown in Nahmias et al. (2013) that the near-threshold dynamics of the laser described can be approximated to Eq. 5:

$$\frac{dG(t)}{dt} = -\gamma_G(G(t) - A) + \theta(t); \quad (5)$$

if  $G(t) > G_{\text{thresh}}$   
release a pulse, and set  $G(t) \rightarrow G_{\text{reset}}$ ,

where  $G(t)$  models the gain,  $\gamma_G$  is the gain carrier relaxation rate, and  $A$  is the gain bias current. The input  $\theta(t)$  can include spike inputs of the form  $\theta(t) = \sum_i \delta_i(t - \tau_i)$  for spike firing times  $\tau_i$ ,  $G_{\text{thresh}}$  is the gain threshold, and  $G_{\text{reset}} \sim 0$  is the gain at transparency.

One can note the striking similarity to the LIF model in Eq. 1: setting the variables  $\gamma_G = 1/R_m C_m$ ,  $A = V_L$ ,  $\theta(t) = I_{app}(t)/R_m C_m$ , and  $G(t) = V_m(t)$  shows their algebraic equivalence. Thus, the gain of the laser  $G(t)$  can be thought of as a virtual *membrane voltage*, the input current  $A$  as a virtual *equilibrium voltage*, etc.

A remarkable difference can be observed between the two systems though: whereas in the neural cell membrane the timescales are governed by an  $R_m C_m$  constant of the order of ms, the carrier dynamics in lasers are as fast as ns. Although this form of excitability was found in two-section lasers, other device morphologies have also shown excitable dynamics. The advantage of

constructing a clear abstraction to the LIF model is that it allows engineers to reuse the same methods developed in the computational neuroscience community for programming a neuromorphic processor.

In the next section, we present recent optical devices with excitable dynamics.

### Semiconductor Excitable Lasers

Optical excitability in semiconductor devices are being widely studied, both theoretically and experimentally. These devices include multisection lasers, ring lasers, photonic crystal nanocavities, tunneling diode attached to laser diodes, and semiconductor lasers with feedback, summarized in Table 2. We group them under the terminology *excitable lasers* for convenience, but exceptions are described in the caption of the table.

Generally speaking, these lasers use III–V quantum wells or quantum dots for efficient light generation. However, they fall into one of three injection categories (illustrated in Fig. 11) and possess very diverse excitability mechanisms. It is difficult to group the rich dynamics of different lasers – which often requires a system of several coupled ordinary differential equations to represent it – using classification keywords. We focus on two fundamental characteristics: the way each laser can be modulated (injection scheme column), and on the physical effect that directly shapes the optical pulse (excitable dynamics column).

The injection scheme of the laser will determine whether it is compatible to all-optical PNNs or O/E/O PNNs. Some of them (B, C, H) operate free of electrical injection, meaning that bits of information remain elegantly encoded in optical carriers. However, as we have pointed out,

**Principles of Neuromorphic Photonics, Fig. 13** Excitable dynamics of the graphene excitable laser. Blue and red curves correspond to input and output pulses, respectively. (a) Cutaway architecture of a hybrid InGaAsP-graphene-silicon evanescent laser (not to scale) showing a terraced view of the center. (b) Cross-sectional profile of the excitable laser with an overlaid electric field (E-field) intensity  $|\vec{E}|^2$  profile. (c–e) Excitable dynamics of the graphene fiber laser. (c) Excitatory activity (temporal

integration of nearby pulses) can push the gain above the threshold, releasing spikes. Depending on the input signal, the system can have a suppressed response due to the presence of either subthreshold input energies (integrated power  $\int |\theta(t)|^2 dt$ ) or (d) a refractory period during which the laser is unable to pulse (regardless of excitation strength). (e) Restorative properties: repeatable pulse shape even when inputs have different energies (Reproduced from Shastri et al. (2016). Licensed under Creative Commons Attribution License (CC BY))

**Principles of Neuromorphic Photonics, Table 2** Characteristics of recent excitable laser devices. Note that this table does not have a one-to-one correspondence to Fig. 12, because some of them are not E/O devices. However, we

observe that devices A, D, and F belong to the category Fig. 12a and device E resembles more closely to the category Fig. 12c

Device	Injection scheme	Pump	Excitable dynamics	References
A. Two-section gain and SA	Electrical	Electrical	Stimulated emission	(Nahmias et al. 2013, 2015; Selmi et al. 2014, 2015; Shastri et al. 2014, 2015, 2016; Barbay et al. 2011; Spühler et al. 1999; Dubbeldam and Krauskopf 1999; Dubbeldam et al. 1999; Larotonda et al. 2002; Elsass et al. 2010)
B. Semiconductor ring laser	Coherent optical	Electrical	Optical interference	(Coomans et al. 2010, 2011, 2013; Van Vaerenbergh et al. 2012; Gelens et al. 2010)
C. Microdisk laser	Coherent optical	Electrical	Optical interference	(Van Vaerenbergh et al. 2013; Alexander et al. 2013)
D. 2D photonic crystal nanocavity <sup>a</sup>	Electrical	Electrical	Thermal	(Brunstein et al. 2012; Yacomotti et al. 2006a, b)
E. Resonant tunneling diode photodetector and laser diode <sup>b</sup>	Electrical or incoherent optical	Electrical	Electrical tunneling	(Romeira et al. 2013; 2016)
F. Injection-locked semiconductor laser with delayed feedback	Electrical	Electrical	Optical interference	(Kelleher et al. 2010, 2011; Garbin et al. 2014, 2015; Goulding et al. 2007; Wieczorek et al. 1999, 2002, 2005; Barland et al. 2003; Marino and Balle 2005; Turconi et al. 2013)
G. Semiconductor lasers with optical feedback	Incoherent optical	Electrical	Stimulated emission	(Aragoneses et al. 2014; Sorrentino et al. 2015; Giudici et al. 1997; Yacomotti et al. 1999; Giacomelli et al. 2000; Heil et al. 2001; Wünsche et al. 2001)
H. Polarization switching VCSELs	Coherent optical	Optical	Optical interference	(Hurtado and Javaloyes 2015; Hurtado et al. 2010, 2012)

<sup>a</sup>Technically this device is not an excitable laser, but an excitable cavity connected to a waveguide

<sup>b</sup>The authors call it *excitable optoelectronic device*, because the excitability mechanism lies entirely in an electronic circuit, rather than the laser itself

avoiding the E/O conversion is much more difficult when you are trying to build a weight-and-sum device compatible with WDM, which is an essential building block for scalable photonic neural networks.

The excitable dynamics determines important properties such as energy efficiency, switching speed, and bandwidth of the nonlinear node. The “optical interference” mechanism typically means that there are two competing modes with a certain phase relationship that can undergo a  $2\pi$  topological excursion and generating an optical pulse in amplitude at the output port. This mechanism is notably different than the others in which it does not require exchange of energy between charge carriers populations and the cavity field. As a result, systems based on this effect are not limited by carrier lifetimes, yet are vulnerable to phase

noise accumulation. Other mechanisms include photon absorption, stimulated emission, thermo-optic effect, and electron tunneling. There, the electronic dynamics of the device governs the population of charge carriers available for stimulated emission, thereby dominating the timescale of the generated pulses. Models of these mechanisms and how they elicit excitability are comprehensively detailed in Prucnal et al. (2016), but a quantitative comparison between performance metrics of lasers in Table 2 is still called for. Qualitatively, however, excitable lasers can simultaneously borrow key properties of electronic transistors, such as thresholding and cascability.

In addition to individual laser excitability, there have been several demonstrations of simple processing circuits. Temporal pattern recognition (Shastri et al. 2016) and stable recurrent memory

(Shastri et al. 2016; Romeira et al. 2016; Garbin et al. 2014) are essential toy circuits that demonstrate basic aspects of network compatibility.

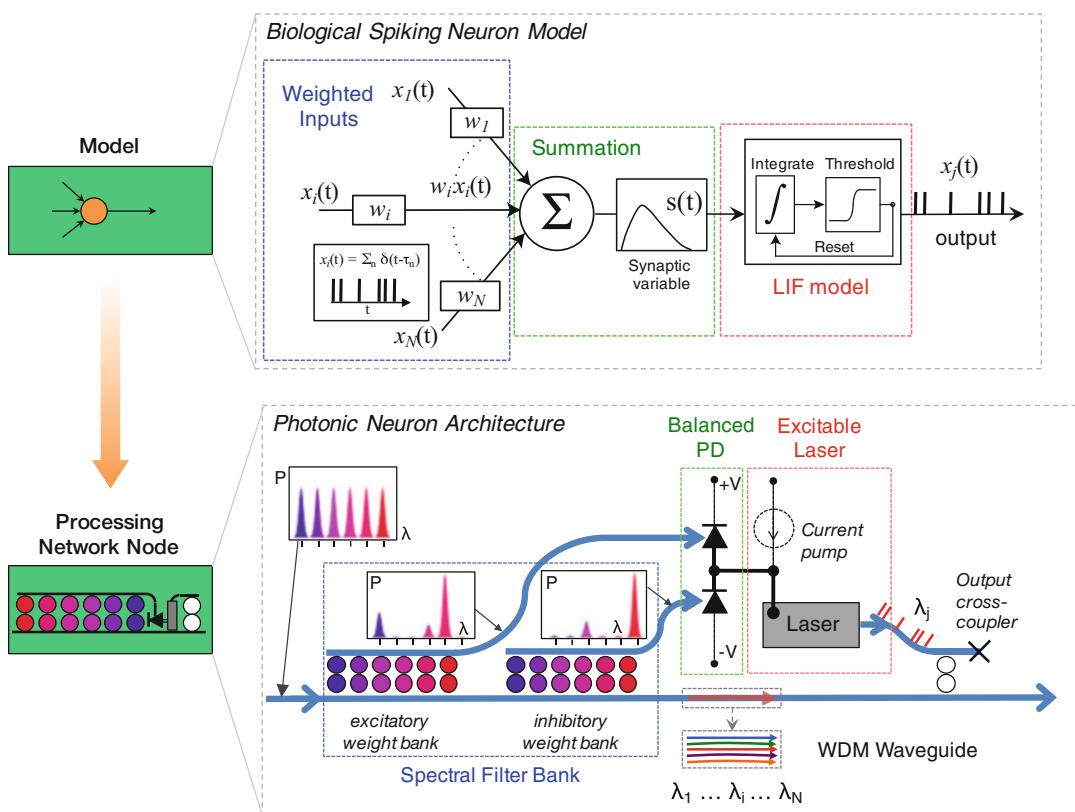
## Photonic Neural Network Architecture

### Isomorphism to Biological Spiking Neuron

Neurons only have computational capabilities if they are in a network. Therefore, an excitable laser (or spiking laser) can only be viewed as a neuron candidate if it is contained in a PNN (Fig. 14). The configurable analog connection strengths between neurons, called weights, are as important to the task of network processing as

the dynamical behavior of individual elements. Earlier, we have discussed several proposed excitable lasers exhibiting neural behavior and cascability between these lasers. In this section, we discuss the challenges involving the creation of a network of neurons using photonic hardware, in particular, the creation of a weighted addition scheme for every PNN. Tait et al. (2014b) proposed an integrated photonic neural networking scheme called *broadcast-and-weight* that uses WDM to support a large number of reconfigurable analog connections using silicon photonic device technology.

A spiking and/or analog photonic network consists of three aspects: a protocol, a node that



**Principles of Neuromorphic Photonics, Fig. 14** Isomorphism between photonic neuron module (i.e., a PNN) to a biological spiking neuron model. Top: depiction of a single unit neural network model. Inputs  $x_i$  are weighted and summed. The result  $\sum_i w_i x_i$  experiences a nonlinear function. Bottom: Possible physical implementation of a PNN. A WDM signal is incident on a bank of filters (excitatory and inhibitory) which are created using a series of microring filters that apply a

series of weights. The resulting signal is incident on a balanced photodetector which applies a summation operation and drives an excitable laser with a current signal. The resulting laser outputs at a specified wavelength which is subsequently coupled back into the broadcast interconnect (see Fig. 15) (Adapted with permission from Tait et al. *J. Lightwave Technol.* **32**, 4029–4041 (2014) (Tait et al. 2014). Copyright 2014 Optical Society of America)

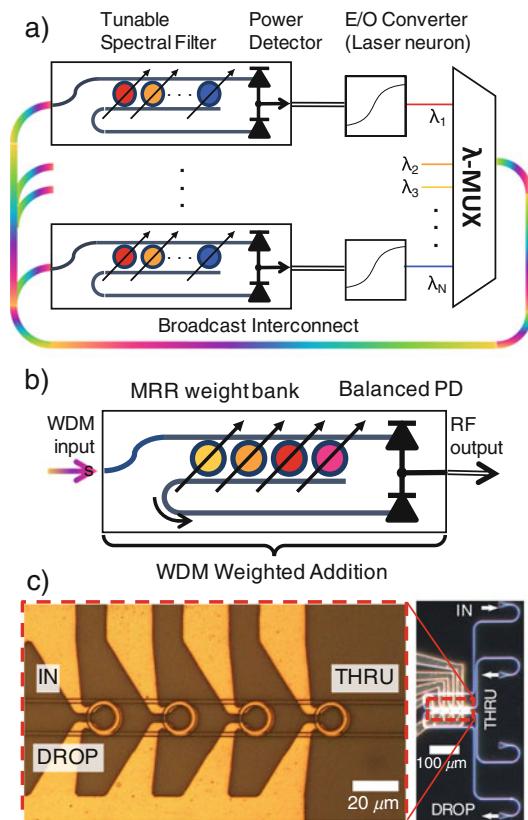
abides by that protocol (the PNN), and a network medium that supports multiple connections between these nodes. This section will begin with broadcast-and-weight as a WDM protocol in which many signals can coexist in a single waveguide and all nodes have access to all the signals. Configurable analog connections are supported by a novel device called a microring resonator (MRR) weight bank (Fig. 15). We will summarize experimental investigations of MRR weight banks.

### Broadcast-and-Weight Protocol

WDM channelization of the spectrum is one way to efficiently use the full capacity of a waveguide, which can have usable transmission windows up to 60 nm (7.5 THz bandwidth) (Preston et al. 2011). In fiber communication networks, a WDM protocol called broadcast-and-select has been used for decades to create many potential connections between communication nodes (Ramaswami 1993). In broadcast-and-select, the active connection is selected, not by altering the intervening medium, but rather by tuning a filter at the receiver to drop the desired wavelength. Broadcast-and-weight is similar but differs by directing multiple inputs simultaneously into each detector (Fig. 15b) and with a continuous range of effective drop strengths between  $-1$  and  $+1$ , corresponding to an analog weighting function.

The ability to control each connection, each weight, independently is a crucial aspect of neural network models. Weighting in a broadcast-and-weight network is accomplished by a tunable spectral filter bank at each node, an operation analogous to a neural weight. The local state of the filters defines the interconnectivity pattern of the network.

A great variety of possible weight profiles allows a group of functionally similar units to instantiate a tremendous variety of neural networks. A reconfigurable filter can be implemented by a MRR – in simple words, a waveguide bent back on itself to create an interference condition. The MRR resonance wavelength can be tuned thermally (as in Fig. 15c) or electronically on timescales much slower than signal bandwidth.



**Principles of Neuromorphic Photonics, Fig. 15** (a) Broadcast-and-weight network. An array of source lasers outputs distinct wavelengths (represented by solid color). These channels are wavelength multiplexed (WDM) in a single waveguide (multicolor). Independent weighting functions are realized by tunable spectral filters at the input of each unit. Demultiplexing does not occur in the network. Instead, the total optical power of each spectrally weighted signal is detected, yielding the sum of the input channels. The electronic signal is transduced to an optical signal after nonlinear transformation. (b) Tunable spectral filter constructed using microring resonator (MRR) weight bank. Tuning MRRs between on- and off-resonance switches a continuous fraction of optical power between drop and through ports. A balanced photodetector (PD) yields the sum and difference of weighted signals. (c) Left: optical micrograph of a silicon MRR weight bank, showing a bank of four thermally tuned MRRs. Right: wide area micrograph, showing fiber-to-chip grating couplers (Reproduced with permission from Tait et al. (2016a). Copyright 2016 Optical Society of America)

Practical, accurate, and scalable MRR control techniques are a critical step toward large-scale analog processing networks based on MRR weight banks.

## Controlling Photonic Weight Banks

Sensitivity to fabrication variations, thermal fluctuations, and thermal cross talk has made MRR control an important topic for WDM demultiplexers (Klein et al. 2005), high-order filters (Mak et al. 2015), modulators (Cox et al. 2014), and delay lines (Cardenas et al. 2010). Commonly, the goal of MRR control is to track a particular point in the resonance relative to the signal carrier wavelength, such as its center or maximum slope point. On the other hand, an MRR weight must be biased at arbitrary points in the filter roll-off region in order to multiply an optical signal by a continuous range of weight values. Feedback control approaches are well-suited to MRR demultiplexer and modulator control (DeRose et al. 2010; Jayatilleka et al. 2015), but these approaches rely on having a reference signal with consistent average power. In analog networks, signal activity can depend strongly on the weight values, so these signals cannot be used as references to estimate weight values. These reasons dictate a feedforward control approach for MRR weight banks.

### Single Channel Control Accuracy and Precision

How accurate can a weight be? The resolution required for effective weighting is a topic of debate within the neuromorphic electronics community, with IBM's TrueNorth selecting four digital bits plus one sign bit (Akopyan et al. 2015). In Tait et al. (2016b), continuous weight control of an MRR weight bank channel was shown using an interpolation-based calibration approach. The goal of the calibration is to have a model of applied current/voltage versus effective weight command. The calibration can be performed once per MRR, and its parameters can be stored in memory. Once calibration is complete, the controller can navigate the MRR transfer function to apply the correct weight value for a given command. However, errors in the calibration, environmental fluctuations, or imprecise actuators cause the weight command to be inaccurate. It is necessary to quantify that accuracy.

Analog weight control accuracy can be characterized in terms of the ratio of weight range (normalized to 1.0) to worst-case weight

inaccuracy over a sweep and stated in terms of bits or a dynamic range. The initial demonstration reported in Tait et al. (2016b) indicates a dynamic range of the weight controller of 9.2 dB, in other words, an equivalent digital resolution of 3.1 bits.

### Multichannel Control Accuracy and Precision

Another crucial feature of an MRR weight bank is simultaneous control of all channels. When sources of cross talk between one weight and another are considered, it is impossible to interpolate the transfer function of each channel independently. Simply extending the single-channel interpolation-based approach of measuring a set of weights over the full range would require a number of calibration measurements that scales exponentially with the channel count, since the dimension of the range grows with channel count. Simultaneous control in the presence of cross talk therefore motivates model-based calibration approaches.

Model-based, as opposed to interpolation-based, calibration involves parameterized models for cross talk inducing effects. The predominant sources of cross talk are thermal leakage between nearby integrated heaters and, in a lab setup, interchannel cross-gain saturation in fiber amplifiers, although optical amplifiers are not a concern for fully integrated systems that do not have fiber-to-chip coupling losses. Thermal cross talk occurs when heat generated at a particular heater affects the temperature of neighboring devices (see Fig. 15c). In principle, the neighboring channel could counter this effect by slightly reducing the amount of heat its heater generates. A calibration model for thermal effects provides two basic functions: forward modeling (given a vector of applied currents, what will the vector of resultant temperatures be?) and reverse modeling (given a desired vector of temperatures, what currents should be applied?). Models such as this must be calibrated to physical devices by fitting parameters to measurements. Calibrating a parameterized model requires at least as many measurements as free parameters. Tait et al. (2016a) describes a method for fitting parameters with  $O(N)$  spectral and oscilloscope measurements, where  $N$  is the number of MRRs. As an example, whereas an interpolation-only approach with 20-point resolution would require  $20^4 = 160,000$  calibration

measurements, the presented calibration routine takes roughly  $4 \times [10(\text{heater}) + 20(\text{filter}) + 4(\text{-amplifier})] = 136$  total calibration measurements. Initial demonstrations achieved simultaneous four-channel MRR weight control with an accuracy of 3.8 bits and precision of 4.0 bits (plus 1.0 sign bit) on each channel. While optimal weight resolution is still a topic of discussion in the neuromorphic electronics community (Hasler and Marr 2013), several state-of-the-art architectures with dedicated weight hardware have settled on 4-bit resolution (Akopyan et al. 2015; Friedmann et al. 2013).

### Scalability with Photonic Weight Banks

Engineering analysis and design relies on quantifiable descriptions of performance called metrics. The natural questions of “how many channels are possible” and, subsequently, “how many more or fewer channels are garnered by a different design” are typically resolved by studying trade-offs. Increasing the channel count performance metric will eventually degrade some other aspect of performance until the minimum specification is violated.

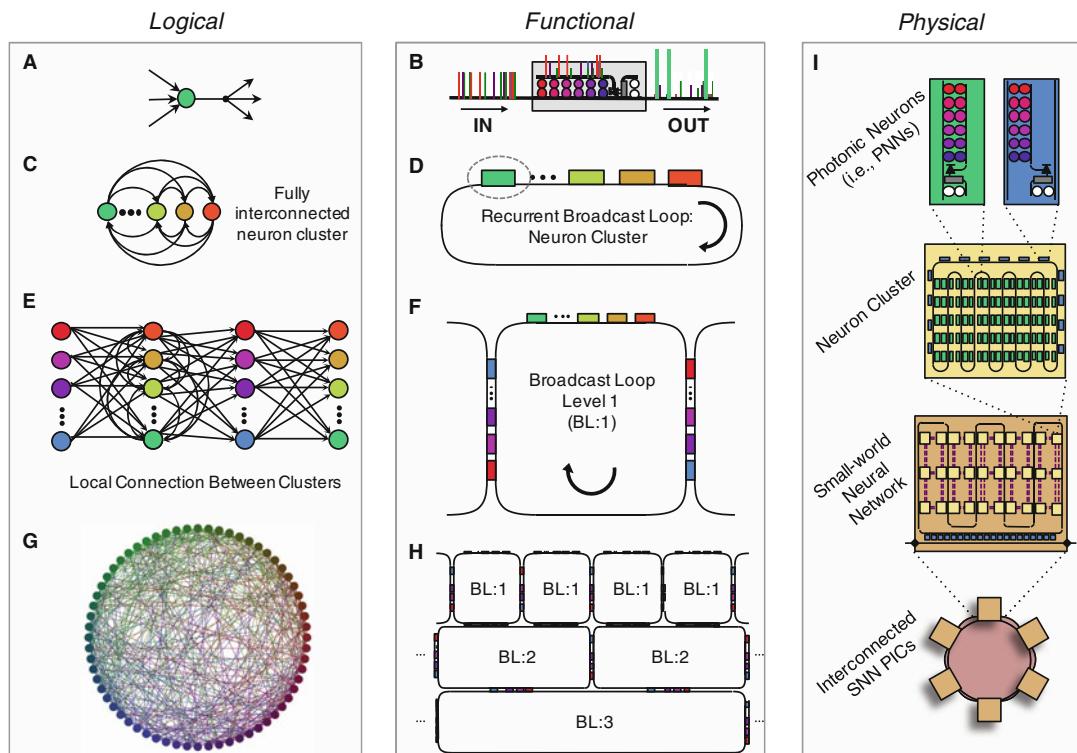
Studying trade-offs between these metrics are important for better designing the network and understanding its limitations. Just as was the case with control methodologies, it was found that quantitative analysis for MRR weight banks must follow an approach significantly different from those developed for MRR demultiplexers and modulators (Tait et al. 2016c).

In conventional analyses of MRR devices for multiplexing, demultiplexing, and modulating WDM signals, the trade-off that limits channel spacing is interchannel cross talk (Preston et al. 2011; Jayatilleka et al. 2016). However, unlike MRR demultiplexers where each channel is coupled to a distinct waveguide output (Klein et al. 2005), MRR weight banks have only two outputs with some portion of every channel coupled to each. All channels are meant to be sent to both detectors in some proportion, so the notion of cross talk between signals breaks down (Fig. 15b). Instead, for dense channel spacing, different filter peaks viewed from the common drop port begin to merge together. This has the effect of reducing the weight bank’s ability to weight neighboring signals independently. As

detailed in Tait et al. (2016c), Tait et al. (1) quantify this effect as a power penalty by including a notion of tuning *range* with the cross-weight penalty metric and (2) perform a channel density analysis by deriving the scalability of weight banks that use microresonators of a particular finesse.

In summary, WDM channel spacing,  $\delta\omega$ , can be used to determine the maximum channel count given a resonator finesse. While finesse can vary significantly with the resonator type, normalized spacing is a property of the circuit (i.e., multiplexer vs. modulators vs. weight bank). Making an assumption that a 3 dB cross-weight penalty is allowed, we find that the minimum channel spacing falls between 3.41 and 4.61 linewidths depending on bus length. High finesse silicon MRRs, such as shown in Xu et al. (2008) (finesse = 368) and (Biberman et al. 2012) (finesse = 540), could support 108 and 148 channels, respectively. Other types of resonators in silicon, such as elliptical microdisks (Xiong et al. 2011) (finesse = 440) and traveling-wave microresonators (Soltani et al. 2010) (finesse = 1140) could reach up to 129 and 334 channels, respectively.

MRR weight banks are an important component of neuromorphic photonics – regardless of PNN implementation because they control the configuration of analog network linking photonic neurons together. In Tait et al. (2016a), it was concluded that ADC resolution, sensitized by biasing conditions, limited the attainable weight accuracy. Controller accuracy is expected to improve by reducing the mismatch between tuning range of interest and driver range. Tait et al. (2016c) arrived at a scaling limit of 148 channels for a MRR weight bank, which is not impressive in the context of neural networks. However, the number of neurons could be extended beyond this limit using spectrum reuse strategies (Fig. 16) proposed in Tait et al. (2014b), by tailoring interference within MRR weight banks as discussed in Tait et al. (2016c), or by packing more dimensions of multiplexing within silicon waveguides, such as mode-division multiplexing. As the modeling requirements for controlling MRR weight banks become more computationally intensive, a feedback control technique would be transformative for both precision and modeling demands. Despite the special



**Principles of Neuromorphic Photonics, Fig. 16** Spectrum reuse strategy. Panels are organized into rows at different scales (core, cluster, chip, and multichip) and into columns at three different views (logical, functional, and physical). (a, b) Depicts the model and photonic implementation of a neuron as a processing network node (PNN) as detailed in Fig. 14. (c, d) Fully interconnected network by attaching PNNs to a broadcast

loop (BL) waveguide. (e, f) A slightly modified PNN can transfer information from one BL to another. (g, h) Hierarchical organization of the waveguide broadcast architecture showing a scalable modular structure. (i) Using this scheme, neuron count in one chip is only limited by footprint, but photonic integrated circuits (PICs) can be further interconnected in an optical fiber network

requirements of photonic weight bank devices making them different from communication-related MRR devices, future research could enable schemes for feedback control.

Many researchers are concentrating their efforts toward the long-term technical potential and functional capability of the hardware compared to standard digital computers. One of the main drivers for the community is computational power efficiency (Hasler and Marr 2013): digital CPUs are reaching a power efficiency wall with the current von Neumann architecture, but hardware neural networks, in which memory and instructions are simplified and colocated, offer to overcome this barrier.

These projects also aimed at simulating large-scale spiking neural networks, with the goal of simulating subcircuits of the human cortex, at a biological timescale (<1 kHz). The HICANN chip, exceptionally, is designed to be accelerated at about 10,000 times respective to biological time-scales and features analog synapses and realistic

## Neuromorphic Platform Comparison

As stated earlier, the neuromorphic computing community has been making vigorous efforts toward large-scale spiking neuromorphic hardware, e.g., Heidelberg HICANN chip via the FACETS/BrainScaleS projects (Schemmel et al. 2010), IBM TrueNorth via the DARPA SyNAPSE program (Merolla et al. 2014), Stanford University's Neurogrid (Benjamin et al. 2014), and SpiNNaker (Furber et al. 2014) (Fig. 4).

neural spiking behaviors (Schemmel et al. 2010). It pays the price of huge power consumption: 800 W for a wafer-scale system containing 180,000 neurons (Benjamin et al. 2014). In contrast, TrueNorth aimed for large-scale, efficient networks optimized for biologically plausible tasks, such as machine vision, but with a simplified neural model (Merolla et al. 2014). Indeed, it contained a total of 1 M neurons and 256 M synapses per chip, consuming only 63 mW of power (Merolla et al. 2014). The Neurogrid board also aimed for scalability and efficiency, consuming 5 W also with 1 M neurons and about four billion synapses, but it kept greater biological fidelity to the mammalian cortex. The SpiNNaker computer is designed to simulate scalably large and versatile networks using arrays of chips with 18 ARM968 processing cores each: unlike the other three technologies, the number of synapses per neuron, or even the number of neurons, is not fixed and can be dynamically reprogrammed (Furber et al. 2014). The demonstrated system has 48 chips interconnected in a PCB, but it can be scaled up to 1200 interconnected PCBs, totalling a 72 kW peak power consumption.

We have recently produced a quantitative comparison between the aforementioned electronic and photonic neuromorphic hardware architectures (Prucnal and Shastri 2017). In order to compare these processors with one another, we reintroduce the multiply-accumulate (MAC) operation that typically bottlenecks complex computations.

The MAC operation takes the following form:  $a \leftarrow a + (w \times x)$ . It includes both a product (i.e.,  $x$  is multiplied by the “weight”  $w$ ) and an addition (the result is accumulated to variable  $a$ ). In neural network models, inputs are combined via a weighted sum of the form  $\sum w_i x_i$ . The result is then input into some nonlinear scalar function  $f\{x\}$  which can range from a simple sigmoid function to a complex nonlinear dynamical system with hysteresis, depending on the complexity of the neuron being modeled. The weighted sum can be broken down into a series of MAC operations of the form  $a_i = a_{i-1} + w_i x_i$  for  $i = 1 \dots M$ . Each neuron requires  $M$  parallel MAC operations per time step  $\Delta t$  (in a given bit period  $\tau$ , determined by the signal bandwidth capacity) or one operation per synapse, where  $M$  refers to the number of

inputs for a given node. Thus, a hardware neural network can be characterized with  $M \times N$  MAC operations per time step  $\Delta t$  (i.e., quadratic scaling of MAC operations), where  $N$  is the number of neurons in the network.

The nonlinear function  $f\{x\}$  also takes up computational resources, but since this operation scales with  $N$  rather than  $M \times N$ , it does not represent the most costly operation. Therefore, as the size of the network  $N$  grows large, MACs – i.e., “synaptic computations” – become the most burdensome hardware bottlenecks in neural networks (Hasler and Marr 2013).

For consistency, we compare architectures that have similar functionality: we limit ourselves to fully reconfigurable systems of spiking neural networks. For the photonically enhanced system, we studied an optoelectronic neural network with PNNs instantiated within the hybrid silicon/III–V platform (Nahmias et al. 2015; Ferreira de Lima et al. 2016). We also consider a future photonic crystal instantiation, based on fundamental physical considerations. Calculated metrics are based on realistic device parameters, derived from the literature. We also refer the reader to a more detailed discussion of spiking electronic neuromorphic hardware in Liu et al. (2015) and an overview of current spiking and nonspiking hardware in Nawrocki et al. (2016).

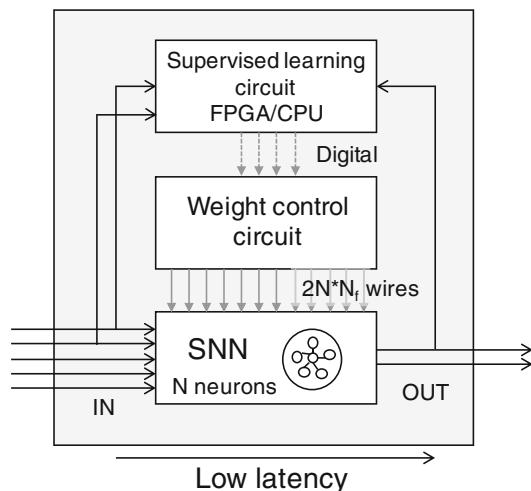
Results are summarized in Table 1. The most striking figure is the number of operations per second, which exceeds electronic platforms by three orders of magnitude, compared to the analog/digital accelerated HICANN and six orders of magnitude compared to the others which are purely digital implementations. This stems from both the high bandwidths and low latencies possible with photonic signals. The optoelectronic approach is able to achieve such energy efficiency at high speeds because power is mainly consumed statically by the lasers, while the passive filters have low leakage current. This contrasts with CMOS digital switches, whose power consumption increases dynamically with clock speed. Processor fan-in is similar in both platforms, despite very differing technologies. The area per MAC is more stringent in a photonically enhanced system, since photonic elements cannot be shrunk beyond

the diffraction limit of light. This is because each data channel requires a weighting filter in the PNN, such as an MRR pair, which adds a footprint penalty. However, this is compensated by the fact that a single waveguide can carry many wideband channels simultaneously, unlike electronic wires. Nonetheless, even though photonically enhanced systems cannot compete with the miniaturization of future nanoelectronics, the estimated footprint of such a system is currently on par with some of the electronics systems presented here.

## Future Directions

After half a century of continuous investment and commercial success, digital CMOS electronics dominates the industry of general-purpose computing. However, with growing demand for connectivity, there is an urgent need for ultrafast coprocessors that could relieve the stress in digital processing circuits. Here, we have presented elements of a reconfigurable photonic hardware that can emulate spiking neural networks operating a billion times faster than the brain. As we identify proper metrics for a neuromorphic photonic processor, research efforts are incipiently transitioning from individual devices to systems design. We are witnessing a fast maturation of standardized photonic foundries in several platforms. Chrostowski and Hochberg say (Chrostowski and Hochberg 2015) we are entering a nascent era of fabless photonics, where users can create computer-assisted chip designs and have it fabricated by these foundries using quality controlled, repeatable processes. It is reasonable to expect that neuromorphic photonic coprocessors (Fig. 17) can be fabricated and packaged using fabless services for near-term research and long-term volume production.

Applications for neuromorphic photonic processors can be grouped into two categories: (1) a front-end stage for RF systems and data centers and (2) ultrafast processing for specialized fast applications (Prucnal and Shastri 2017). The first category utilizes the low-latency, parallelism, and energy efficiency properties of photonics to alleviate the throughput of RF systems, e.g., by



**Principles of Neuromorphic Photonics, Fig. 17** Diagram description of a fully packaged neuromorphic processor. While two layers of electronics provide reconfigurability, the photonic spiking neural network permits low-latency functionality.  $N_f$ : fan-in of each neuron (Reproduced from Ferreira de Lima et al. 2017). Licensed under Creative Commons Attribution-NonCommercial-NoDerivatives License. (CC BY-NC-ND))

executing dimensionality reduction tasks such as principal component analysis or blind-source separation. The second category takes advantage of the raw speed (bandwidth and latency) of the photonic processor to execute iterative algorithms mapped to recurrent neural networks.

Neuromorphic photonic processors join a class of photonic hardware accelerators designed to assist in acquisition, feature extraction, and storage of wideband waveforms (Jalali and Mahjoubfar 2015). These accelerators manipulate the spectrotemporal of a wideband signal, a task difficult to accomplish in analog electronics over broad bandwidth and with low loss.

## Real-Time Radio-Frequency Processing

High-volume data applications, including streaming video and cloud services, will continue to push the telecommunications industry to build better, high-bandwidth systems. Data traffic on some mobile networks alone has increased by over 6000% (Index 2015). This has motivated the exploration of more efficient usage of spectral resources (Akyildiz et al. 2006). Although RF

integrated circuits (RFICs) have been researched for applications such as duplex processing (Lee 2003; Razavi 2000) or control of beamforming antennas (Yu et al. 2011; Razavi 2009), the requirement for impedance-matched transmission lines greatly increases device and interconnect footprint, limiting the overall complexity of each chip. Photonics provides a solution to these fundamental limitations: optical waveguides can support large bandwidths ( $\sim 100$  s of THz) with high information density and low cross talk between multiplexed channels. By using techniques such as WDM, a large number of multi-GHz channels can exist within the same optical waveguide. As a result, the number of virtual channels can greatly exceed the number of physical waveguides, allowing for the formation of complex processing circuits without a significant hardware overhead.

After some initial front-end processing (i.e., heterodyning and amplification), most radio transceiver systems are processed by either digital signal processors (DSP) or field programmable gate arrays (FPGAs) for more complex signal operations. However, the speeds of these processors (i.e.,  $\sim 500$  MHz) limit the overall throughput of RF carrier signals, which can easily be in the  $\sim$ GHz. Clever sampling and parallelization can help alleviate this bottleneck but at the cost of much higher latency and a significant resource/energy overhead. Specialized RF application-specific integrated circuits (ASICs) are another option but are expensive, require significant development time, and have limited re-configurability. Future imagined multiple-in multiple-out (MIMO) systems – which, in the case of massive MIMO, can be on the order of  $\sim 100$  s of input and output channels (Larsson et al. 2014; Gesbert et al. 2003) – are especially susceptible to this bottleneck and may require a radically new solution.

Adding a photonic processing chip to the front of a radio transceiver would allow very complex operations to be performed in real time, which can significantly offload electronic post-processing and provide a technology to make faster, more relevant RF decisions on the fly. Massive MIMO systems based on beamforming in phased array antennas require a processor that can distinguish and operate on hundreds of high-bandwidth

signals simultaneously, a feat that is currently speed limited by current electronic processors (Larsson et al. 2014; Hansen 2009). A photonic neural network model is a perfect fit for addressing this kind of technological challenge: efficient MIMO beamforming relies on MAC operations that are already applied in neural network models via *weighted addition*. In addition, classification algorithms can be built efficiently using the neural network approach, allowing for RF fingerprinting and signal identification.

As spread spectrum, adaptive RF transceivers become more widespread in future telecommunication systems, the scalability of the photonic approach could provide significant processing advantages. Its high bandwidth, low latency, and high throughput would be especially useful in an ultra-wideband (UWB) radio system, in which it could sample from many frequencies and directions simultaneously to scan for spectral opportunities and make a decision quickly and efficiently. Pairing this technology with an FPGA or electronic ASIC controller would enable the implementation of adaptive optimization and learning algorithms in real time for ultrafast cognitive radio applications.

## Nonlinear Programming

Another way of taking advantage of raw speed is via an *iterative* approach. Iterative algorithms find successfully better approximations to a problem of interest and often require many time steps to reach a desired solution. Since one of the most salient advantages of a photonic approach is its low time of flight ( $\sim$ ps) between communicating processors, the convergence rates can be significantly improved by implementing them on a photonic platform. A large class of problems that can be solved iteratively includes *linear and nonlinear programming problems*. These methods seek to minimize some objective function  $E(\vec{x})$  of real variables in  $\vec{x}$  subject to a series of constraints represented by equalities or inequalities, i.e.,  $g(\vec{x}) \leq 0$  and  $h(\vec{x}) = 0$ . Applications in telecommunications, aerospace, and financial industries can be described in this basic framework, including optimal portfolio trading

strategies, control of machinery/actuators, and allocation of resources and jobs in online servers. Using a photonic approach, 100-variable problems could converge in less than  $\sim$ 100 ns, which could be useful in the control of very fast dynamical systems (i.e., actuators) or in the creation of low-latency optimization routines in data-intensive environments.

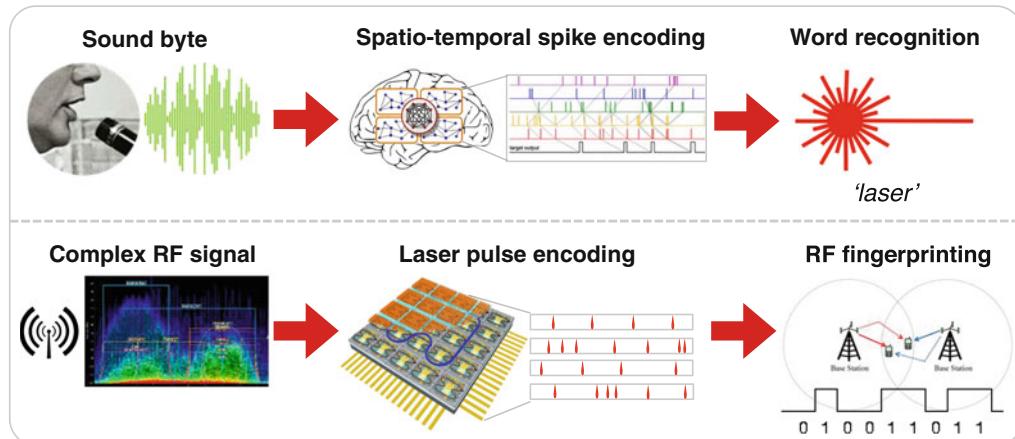
Mathematical optimization problems can be grouped into *linear* and *nonlinear* optimization problems. Nonlinear optimization problems are often difficult to solve and sometimes involve exotic techniques such as genetic algorithms or particle swarm optimization. Nonlinear optimization problems, however, are nonetheless *quadratic* to second order around the local vicinity of the optimum. Therefore, quadratic programming (QP) – which finds the minima/maxima quadratic functions of variables subject to linear constraints (Lendaris et al. 1999) – becomes an effective first pass at such problems and can be applied to a wide array of applications. For example, many machine learning problems, such as support vector machine (SVM) training and least-squares regression, can be reformulated in terms of a QP problem. In addition, computational problems such as model predictive control (MPC), an optimal nonlinear control algorithm, or compressive sampling, a method for sampling at rates below the Nyquist without loss of information via the characterization of sparsity in incoming data, are examples of QP problems. Together, these applications represent some of the most effective yet generalized tools for acquiring and processing information and using the results to control systems. QP is an NP hard problem in the number of variables, which means that conventional digital computers must either be limited to solving quadratic programs of very few variables or to applications where computation time is noncritical. This is reflected in industrial applications of QP solvers. MPC is used in the chemical industry to control chemical processing plants, where reaction timescales can be made very long, and in the finance industry to control long-term portfolio optimization. The application of MPC to faster systems, therefore, relies on new ways of finding faster solutions to

QP problems (Jerez et al. 2011). In machine learning, many algorithms (such as SVM) require offline training because of the computational complexity of QP but would be much more effective if they could be trained online (Fig. 18).

While Hopfield showed that Hopfield networks are able to solve quadratic optimization problems quickly over 25 years ago (Tank and Hopfield 1986), Hopfield quadratic optimizers are uncommon today. This is largely due to the high connectivity between neurons that neural networks require ( $n^2$  connections for  $n$  neurons). In an electronic circuit, as the number of connections increases, the bandwidth at which the system can operate without being subject to cross talk between connections and other issues decreases (Tait et al. 2014b). This creates an undesirable trade-off between neuron speed and neural network size. Photonic neural networks have several advantages over their electrical counterparts. Most importantly, the connectivity concerns prevalent in electronic neurons are significantly ameliorated by using light as a communication medium (Tait et al. 2014b). WDM allows for hundreds of high-bandwidth signals to flow through a single optical waveguide. Moreover, the analog computational bandwidth of a photonic neuron (as designed in Tait et al. (2016b)) lies in the picosecond to femtosecond timescale (Nahmias et al. 2015). For a Hopfield quadratic optimizer, this means that a photonic implementation (Fig. 19) can simultaneously have large dimensionality and a fast convergence time to the minimum. These processors represent some of the most effective yet generalized tools for acquiring and processing information and controlling highly mobile systems, such as a hypersonic aircraft (Keviczky and Balas 2006).

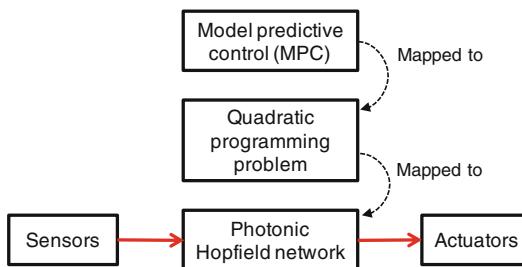
## Summary and Conclusion

Photonics has revolutionized information communication, while electronics, in parallel, has dominated information processing. Recently, there has been a determined exploration of the unifying boundaries between photonics and electronics on the same substrate, driven in part by



**Principles of Neuromorphic Photonics, Fig. 18** A vision for a brain-inspired laser neural network for enhanced RF communication. Top: spoken words are transformed into spatiotemporal “spike” (event) patterns by biological neurons. A pattern recognition neuron is sensitive to a specific spike fingerprint and releases its own spike only if it occurs, shown in “target output.” Bottom: in analogy with audio waveforms, a much faster

system ( $\sim$ GHz) based on excitable lasers could operate directly on RF waveforms. Applying operations at the front-end of RF transceivers could offload complex signal processing operations to a photonic chip and address bandwidth and latency limitations of current FPGA and DSP solutions (The spike-coded pattern is reproduced from Tapson et al. 2013). Licensed under Creative Commons Attribution License (CC BY))



**Principles of Neuromorphic Photonics, Fig. 19** Employing a photonic neural network for a model predictive control problem by solving a quadratic programming problem with a Hopfield network

Moore’s law approaching its long-anticipated end. For example, the computational efficiency for digital processing has leveled off around 100 pJ per MAC. As a result, there has been a widening gap between today’s computational efficiency and the next-generation needs, such as big data applications which require advanced pattern matching and real-time analysis. This, in turn, has led to expeditious advances in (1) emerging devices that are called “beyond CMOS” or “More-than-Moore”; (2) novel processing or unconventional computing architectures called “beyond von Neumann” that are brain-inspired, i.e., neuromorphic;

and (3) CMOS-compatible photonic interconnect technologies. Collectively, these research endeavors have given rise to the field of neuromorphic photonics (Fig. 1). Emerging photonic hardware platforms have the potential to vastly exceed the capabilities of electronics by combining ultrafast operation, moderate complexity, and full programmability, extending the bounds of computing for applications such as navigation control on hypersonic aircrafts and real-time analysis of the RF spectrum.

In this entry, we discussed the current progress and requirements of such a platform. In a photonic spike processor, information is encoded as events in the temporal and spatial domains of spikes (or optical pulses). This hybrid coding scheme is digital in amplitude but analog in time and benefits from the bandwidth efficiency of analog processing and the robustness to noise of digital communication. Optical pulses are received, processed, and generated by certain class of semiconductor devices that exhibit excitability – a nonlinear dynamical mechanism underlying all-or-none responses to small perturbations. Optoelectronic devices operating in the excitable regime are dynamically analogous to a

biophysical neuron, but roughly eight orders of magnitude faster. We dubbed these devices as “photonic neurons” or “laser neurons.” The field is now reaching a critical juncture where there is a shift from studying single photonic neurons to studying interconnected networks of such devices. A recently proposed on-chip networking architecture called broadcast-and-weight could support massively parallel (*all-to-all*) interconnection between excitable devices using wavelength division multiplexing.

A hybrid III–V/Si photonic platform is a candidate for an integrated hardware platform. III–V compound semiconductor technology, such as indium phosphide (InP) and gallium arsenide (GaAs), is at the forefront of providing *active* elements like lasers, amplifiers, and detectors. Silicon, in parallel, brings compatibility with CMOS fabrication processes and low-loss *passive* components like waveguides and resonators. Scalable and fully reconfigurable networks of excitable lasers can be implemented in the silicon photonic layer of modern hybrid integration platforms, in which spiking lasers in a bonded InP layer are densely interconnected through a silicon layer. Such a photonic spike processor will potentially be able to support several thousand interconnected devices. It is predicted that such a chip would have a computational efficiency of 260 fJ per MAC, which surpasses the energy efficiency wall by two orders of magnitude while operating at high speeds (i.e., signal bandwidths 10 GHz). The emerging field of photonic spike processors has received tremendous interest and continues to develop as photonic integrated circuits increase in performance and scale. As novel applications requiring real-time, ultrafast processing – such as the exploitation of the RF spectrum – become more demanding, we expect that these systems will find use in a variety of high-performance, time-critical environments.

Moving forward, we envision a tremendous interest in designing, building, and understanding photonic networks of excitable elements for ultrafast information processing, guided by the latest computational models of the brain. Successful implementation of a small-scale photonic spike

processor could, in principle, provide the fundamental technology to build and study larger-scale brain-inspired networks based on laser excitability. Neuromorphic photonics is poised to usher in exciting new fields of inquiry and impactful enterprises of application.

## Bibliography

- Agrawal GP (2002) Fiber-optic communication systems. Wiley series in microwave and optical engineering (Wiley-interscience). Wiley, New York
- Akopyan F, Sawada J, Cassidy A, Alvarez-Icaza R, Arthur J, Merolla P, Imam N, Nakamura Y, Datta P, Nam GJ, Taba B, Beakes M, Brezzo B, Kuang J, Manohar R, Risk W, Jackson B, Modha D (2015) IEEE Trans Comput Aided Des Integr Circuits Syst. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. 34:1537
- Akyildiz IF, Lee WY, Vuran MC, Mohanty S (2006) Comput Netw 50:2127
- Alexander K, Van Vaerenbergh T, Fiers M, Mechet P, Dambre J, Bienstman P (2013) Opt Express 21:26182
- Aragoneses A, Perrone S, Sorrentino T, Torrent MC, Masoller C (2014) Sci Rep 4:4696 EP
- Backus J (1978) Commun ACM 21:613
- Barbay S, Kuszelewicz R, Yacomotti AM (2011) Opt Lett 36:4476
- Barland S, Piro O, Giudici M, Tredicce JR, Balle S (2003) Phys Rev E 68:036209
- Barwick T, Taira Y, Lichoulas TW, Boyer N, Martin Y, Numata H, Nah JW, Takenobu S, Janta-Polczynski A, Kimbrell EL, Leidy R, Khater MH, Kamlapurkar S, Engelmann S, Vlasov YA, Fortier P (2016) IEEE J Sel Top Quantum Electron 22:455
- Bengio Y, Courville A, Vincent P (2013) IEEE Trans Pattern Anal Mach Intell 35:1798
- Benjamin B, Gao P, McQuinn E, Choudhary S, Chandrasekaran A, Bussat JM, Alvarez-Icaza R, Arthur J, Merolla P, Boahen K (2014) Proc IEEE 102:699
- Bhalla US, Iyengar R (1999) Science 283:381
- Biberman A, Shaw MJ, Timurdogan E, Wright JB, Watts MR (2012) Opt Lett 37:4236
- Boahen K (2000) IEEE Trans Circuits Syst II, Analog Digit Signal Process 47:416
- Borst A, Theunissen FE (1999) Nat Neurosci 2:947
- Brunner D, Soriano MC, Mirasso CR, Fischer I (2013a) Nat Commun 4:1364
- Brunner D, Soriano MC, Fischer I (2013b) IEEE Photon Technol Lett 25:1680
- Brunstein M, Yacomotti AM, Sagnes I, Raineri F, Bigot L, Levenson A (2012) Phys Rev A 85:031803
- Burgsteiner H (2005) On learning with recurrent spiking neural networks and their applications to robot control with real-world devices. PhD thesis, Graz University of Technology

- Capmany J, Ortega B, Pastor D (2006) *J Lightwave Technol* 24:201
- Cardenas J, Foster MA, Sherwood-Droz N, Poitras CB, Lira HLR, Zhang B, Gaeta AL, Khurgin JB, Morton P, Lipson M (2010) *Opt Express* 18:26525
- Caulfield HJ, Dolev S (2010) *Nat Photonics* 4:261
- Chrostowski L, Hochberg M (2015) Silicon photonics design: from devices to systems. Cambridge University Press, Cambridge
- Coomans W (2012) Nonlinear dynamics in semiconductor ring lasers towards an integrated optical neuron. PhD thesis, Vrije Universiteit Brussel
- Coomans W, Beri S, Sande GVD, Gelens L, Danckaert J (2010) *Phys Rev A* 81:033802
- Coomans W, Gelens L, Beri S, Danckaert J, Van Der Sande G (2011) *Phys Rev E* 84:1
- Coomans W, Van der Sande G, Gelens L (2013) *Phys Rev A* 88:033813
- Cox JA, Lentine AL, Trotter DC, Starbuck AL (2014) *Opt Express* 22:11279
- Crescenzi P, Goldman D, Papadimitriou C, Piccolboni A, Yannakakis M (1998) *J Comput Biol* 5:597
- Dennard R, Rideout V, Bassous E, LeBlanc A (1974) *IEEE J Solid State Circuits* 9:256
- DeRose CT, Watts MR, Trotter DC, Luck DL, Nielson GN, Young RW (2010) In: Conference on lasers and electro-optics 2010 (Optical Society of America), p CThJ3
- Diesmann M, Gewaltig MO, Aertsen A (1999) *Nature* 402:529
- Donges JF, Zou Y, Marwan N, Kurths J (2009) *Eur Phys J Special Topics* 174:157
- Dubbeldam JLA, Krauskopf B (1999) *Opt Commun* 159:325
- Dubbeldam JLA, Krauskopf B, Lenstra D (1999) *Phys Rev E* 60:6580
- Duport F, Schneider B, Smerieri A, Haelterman M, Massar S (2012) *Opt Express* 20:22783
- Duport F, Smerieri A, Akroud A, Haelterman M, Massar S (2016) *Sci Rep* 6:22381 EP
- Eliasmith C (2005) *Neural Comput* 17:1276
- Eliasmith C, Stewart TC, Choo X, Bekolay T, DeWolf T, Tang Y, Rasmussen D (2012) *Science* 338:1202
- Elsass T, Gauthron K, Beaudoing G, Sagnes I, Kuszelewicz R, Barbay S (2010) *Eur Phys J D* 59:91
- Esmailzadeh H, Blem E, St. Amant R, Sankaralingam K, Burger D (2012) *IEEE Micro* 32:122
- Fang AW, Park H, Hao Kuo Y, Jones R, Cohen O, Liang D, Raday O, Sysak MN, Paniccia MJ, Bowers JE (2007) *Mater Today* 10:28
- Ferreira de Lima T, Shastri BJ, Nahmias MA, Tait AN, Prucnal PR (2016) In: Summer topicals meeting series (SUM), 2016 (IEEE, 2016)
- Ferreira de Lima T, Shastri BJ, Tait AN, Nahmias MA, Prucnal PR (2017) *Nanophotonics* 6:577
- Fok MP, Deming H, Nahmias M, Rafidi N, Rosenbluth D, Tait A, Tian Y, Prucnal PR (2011) *Opt Lett* 36:19
- Friedmann S, Frémaux N, Schemmel J, Gerstner W, Meier K (2013) *Front Neurosci* 7:160
- Furber S, Galluppi F, Temple S, Plana L (2014) *Proc IEEE* 102:652
- Garbin B, Goulding D, Hegarty SP, Huyet G, Kelleher B, Barland S (2014) *Opt Lett* 39:1254
- Garbin B, Javaloyes J, Tissoni G, Barland S (2015) *Nat Commun* 6:5915 EP, 1409.6350
- Gelens L, Mashai L, Beri S, Coomans W, Van der Sande G, Danckaert J, Verschaffelt G (2010) *Phys Rev A* 82:063841
- Gesbert D, Shafi M, Shan Shiu D, Smith PJ, Naguib A (2003) *IEEE J Sel Areas Commun* 21:281
- Giacomelli G, Giudici M, Balle S, Tredicce JR (2000) *Phys Rev Lett* 84:3298
- Giudici M, Green C, Giacomelli G, Nespolo U, Tredicce JR (1997) *Phys Rev E* 55:6414
- Goulding D, Hegarty SP, Rasskazov O, Melnik S, Hartnett M, Greene G, McInerney JG, Rachinskii D, Huyet G (2007) *Phys Rev Lett* 98:153903
- Hansen RC (2009) Phased array antennas. Wiley, Hoboken, NJ
- Hasler J, Marr B (2013) *Front Neurosci* 7:118
- Heck M, Bowers J (2014) *IEEE J Sel Top Quantum Electron* 20:332
- Heck M, Bauters J, Davenport M, Doylend J, Jain S, Kurezveil G, Srinivasan S, Tang Y, Bowers J (2013) *IEEE J Sel Top Quantum Electron* 19:6100117
- Heil T, Fischer I, Elsäßer W, Gavrielides A (2001) *Phys Rev Lett* 87:243901
- Hicke K, Escalona-Morán MA, Brunner D, Soriano MC, Fischer I, Mirasso CR (2013) *IEEE J Sel Top Quantum Electron* 19:1501610
- Hidalgo CA, Klinger B, Barabási AL, Hausmann R (2007) *Science* 317:482
- Hodgkin AL, Huxley AF (1952) *J Physiol* 117:500
- Hopfield JJ (1982) *Proc Natl Acad Sci* 79:2554
- Hurtado A, Javaloyes J (2015) *Appl Phys Lett* 107:241103
- Hurtado A, Henning ID, Adams MJ (2010) *Opt Express* 18:25170
- Hurtado A, Schires K, Henning ID, Adams MJ (2012) *Appl Phys Lett* 100:103703
- Index CVN (2015) White Paper, February
- Indiveri G, Linares-Barranco B, Hamilton T, van Schaik A, Etienne-Cummings R, Delbrück T, Liu SC, Dudek P, Häfliger P, Renaud S, Schemmel J, Cauwenberghs G, Arthur J, Hyunna K, Folowosele F, SAÏGHI S, Serrano-Gotarredona T, Wijekoon J, Wang Y, Boahen K (2011) *Front Neurosci* 5:73
- Izhikevich E (2003) *IEEE Trans Neural Netw* 14:1569
- Izhikevich EM (2004) *IEEE Trans Neural Netw* 15:1063
- Izhikivich EM (2007) Dynamical systems in neuroscience: the geometry of excitability and bursting. MIT Press, Cambridge
- Jaeger H, Haas H (2004) *Science* 304:78
- Jalali B, Fathpour S (2006) *J Lightwave Technol* 24:4600
- Jalali B, Mahjoubfar A (2015) *Proc IEEE* 103:1071
- Jayatilleka H, Murray K, Ángel Guillén-Torres M, Caverley M, Hu R, Jaeger NAF, Chrostowski L, Shekhar S (2015) *Opt Express* 23:25084

- Jayatilleka H, Murray K, Caverley M, Jaeger N, Chrostowski L, Shekhar S (2016) J Lightwave Technol 34:2886
- Jerez JL, Constantinides GA, Kerrigan EC (2011) ACM/SIGDA international symposium on field programmable gate arrays FPGA, Monterey, CA, p 209
- Kelleher B, Bonatto C, Skoda P, Hegarty SP, Huyet G (2010) Phys Rev E 81:036204
- Kelleher B, Bonatto C, Huyet G, Hegarty SP (2011) Phys Rev E 83:026207
- Keviczky T, Balas GJ (2006) Control Eng Pract 14:1023
- Keyes RW (1985) Opt Acta Int J Optics 32:525
- Klein E, Geuzebroek D, Kelderman H, Sengo G, Baker N, Driessens A (2005) IEEE Photon Technol Lett 17:2358
- Koomey J, Berard S, Sanchez M, Wong H (2011) IEEE Ann Hist Comput 33:46
- Krauskopf B, Schneider K, Sieber J, Wieczorek S, Wolfrum M (2003) Opt Commun 215:367
- Kravtsov K, Fok MP, Rosenbluth D, Prucnal PR (2011) Opt Express 19:2133
- Kumar A, Rotter S, Aertsen A (2010) Nat Rev Neurosci 11:615
- Larger L, Soriano MC, Brunner D, Appeltant L, Gutierrez JM, Pesquera L, Mirasso CR, Fischer I (2012) Opt Express 20:3241
- Larotonda MA, Hnilo A, Mendez JM, Yacomotti AM (2002) Phys Rev A 65:033812
- Larsson E, Edfors O, Tufvesson F, Marzetta T (2014) IEEE Commun Mag 52:186
- Lee TH (2003) The design of CMOS radio-frequency integrated circuits, 2nd edn. Cambridge University Press, New York, NY
- Lee WC, Hu C (2001) IEEE Trans Electron Devices 48:1366
- Lendaris GG, Mathia K, Saeks R (1999) IEEE Trans Syst Man Cybern B (Cybern) 29:114
- Liang D, Bowers JE (2010a) Nat Photonics 4:511
- Liang D, Bowers JE (2010b) Nat Photonics 4:511
- Liang D, Roelkens G, Baets R, Bowers JE (2010) Materials 3:1782
- Liu SC, Delbrück T, Indiveri G, Whatley A, Douglas R (2015) Event-based neuromorphic systems. Wiley, Chichester
- Maass W (1997) Neural Netw 10:1659
- Maass W, Natschläger T, Markram H (2002) Neural Comput 14:2531
- Mak J, Sacher W, Xue T, Mikkelsen J, Yong Z, Poon J (2015) IEEE J Quantum Electron 51:1
- Marino F, Balle S (2005) Phys Rev Lett 94:094101
- Markram H, Meier K, Lippert T, Grillner S, Frackowiak R, Dehaene S, Knoll A, Sompolinsky H, Verstreken K, DeFelipe J, Grant S, Changeux JP, Saria A (2011) Procedia Comput Sci 7:39
- Marpaung D, Roeloffzen C, Heideman R, Leinse A, Sales S, Capmany J (2013) Laser Photonics Rev 7:506
- Marr B, Degnan B, Hasler P, Anderson D (2013) IEEE Trans Very Large Scale Integr (VLSI) Syst 21:147
- Martinenghi R, Rybalko S, Jacquiot M, Chembo YK, Larger L (2012) Phys Rev Lett 108:244101
- Mathur N (2002) Nature 419:573
- Merkle RC (1989) Foresight Update 6
- Merolla PA, Arthur JV, Alvarez-Icaza R, Cassidy AS, Sawada J, Akopyan F, Jackson BL, Imam N, Guo C, Nakamura Y, Brezzo B, Vo I, Esser SK, Appuswamy R, Taba B, Amir A, Flickner MD, Risk WP, Manohar R, Modha DS (2014) Science 345:668
- Mesaritakis C, Papataxiaris V, Syvridis D (2013) J Opt Soc Am B 30:3048
- Miller DAB (2000) Proc IEEE 88:728
- Modha DS, Ananthanarayanan R, Esser SK, Ndirango A, Sherbondy AJ, Singh R (2011) Commun ACM 54:62
- Moore GE (2000) Chapter: cramming more components onto integrated circuits. Morgan Kaufmann Publishers Inc., San Francisco, pp 56–59
- Mundy A, Knight J, Stewart T, Furber S (2015) Neural networks (IJCNN), 2015 international joint conference on (2015), IEEE, pp 1–8
- Nahmias MA, Shastri BJ, Tait AN, Prucnal PR (2013) IEEE J Sel Top Quantum Electron 19:1–12
- Nahmias MA, Tait AN, Shastri BJ, de Lima TF, Prucnal PR (2015) Opt Express 23:26800
- Nahmias MA, Tait AN, Tolias L, Chang MP, Ferreira de Lima T, Shastri BJ, Prucnal PR (2016) Appl Phys Lett 108:151106
- Nawrocki RA, Voyles RM, Shaheen SE (2016) IEEE Trans Electron Devices 63:3819
- Ortíñ S, Soriano MC, Pesquera L, Brunner D, San-Martín D, Fischer I, Mirasso CR, Gutiérrez JM (2015) Sci Rep 5:14945 EP
- Ostojic S (2014) Nat Neurosci 17:594
- Paquot Y, Duport F, Smerieri A, Dambre J, Schrauwen B, Haelterman M, Massar S (2012) Sci Rep 2:287 EP
- Paugam-Moisy H, Bohte S (2012) Computing with spiking neuron networks. In: Rozenberg G, Bäck T, Kok JN (eds) Handbook of natural computing. Springer, Berlin/Heidelberg, pp 335–376
- Perrett DI, Rolls ET, Caan W (1982) Exp Brain Res 47:329
- Pfeil T, Grübl A, Jeltsch S, Müller E, Müller P, Petrovici MA, Schmuker M, Brüderle D, Schemmel J, Meier K (2013) Front Neurosci 7:1–17
- Pickett MD, Medeiros-Ribeiro G, Williams RS (2013) Nat Mater 12:114
- Preston K, Sherwood-Droz N, Levy JS, Lipson M (2011) In: CLEO:2011 laser applications to photonic applications (Optical Society of America), p CThP4
- Prucnal PR, Shastri BJ (2017) Neuromorphic photonics. CRC Press/Taylor & Francis Group, Boca Raton
- Prucnal PR, Shastri BJ, de Lima TF, Nahmias MA, Tait AN (2016) Adv Opt Photon 8:228
- Ramaswami R (1993) IEEE Commun Mag 31:78
- Razavi B (2000) Design of analog CMOS integrated circuits. McGraw-Hill Education, New York, NY
- Razavi B (2009) IEEE Trans Circuits Syst Regul Pap 56:4
- Roelkens G, Liu L, Liang D, Jones R, Fang A, Koch B, Bowers J (2010) Laser Photonics Rev 4:751
- Romeira B, Javaloyes J, Ironside CN, Figueiredo JML, Balle S, Piro O (2013) Opt Express 21:20931

- Romeira B, Avó R, Figueiredo JL, Barland S, Javaloyes J (2016) *Sci Rep* 6:19510 EP
- Rosenbluth D, Kravtsov K, Fok MP, Prucnal PR (2009) *Opt Express* 17:22767
- Sarpeshkar R (1998) *Neural Comput* 10:1601
- Schemmel J, Briiderle D, Gribel A, Hock M, Meier K, Millner S (2010) In: Proceedings of 2010 I.E. international symposium on circuits and systems (IEEE, 2010), pp 1947–1950
- Selmi F, Braive R, Beaudoin G, Sagnes I, Kuszelewicz R, Barbay S (2014) *Phys Rev Lett* 112:183902
- Selmi F, Braive R, Beaudoin G, Sagnes I, Kuszelewicz R, Barbay S (2015) *Opt Lett* 40:5690
- Shainline JM, Buckley SM, Mirin RP, Nam SW (2017) *Phys Rev Appl* 7:034013
- Shastri BJ, Nahmias BJ, Tait AN, Prucnal PR (2014) *Opt Quantum Electron* 46:1353
- Shastri BJ, Nahmias MA, Tait AN, Wu B, Prucnal PR (2015) *Opt Express* 23:8029
- Shastri BJ, Nahmias MA, Tait AN, Rodriguez AW, Wu B, Prucnal PR (2016) *Sci Rep* 6:19126 EP
- Shen Y, Harris NC, Skirlo S, Prabhu M, BaehrJones T, Hochberg M, Sun X, Zhao S, Larochelle H, Englund D, Soljacic M (2017) *Nat Photonics*. arXiv:1610.02365
- Smit M, van der Tol J, Hill M (2012) *Laser Photonics Rev* 6:1
- Snider GS (2007) *Nanotechnology* 18:365202
- Soltani M, Li Q, Yegnanarayanan S, Adibi A (2010) *Opt Express* 18:19541
- Soriano MC, Ortín S, Brunner D, Larger L, Mirasso CR, Fischer I, Pesquera L (2013) *Opt Express* 21:12
- Soriano MC, Brunner D, Escalona-Moran M, Mirasso CR, Fischer I (2015) *Front Comput Neurosci* 9:1–11
- Sorrentino T, Quintero-Quiroz C, Aragoneses A, Torrent MC, Masoller C (2015) *Opt Express* 23:5571
- Spühler GJ, Paschotta R, Fluck R, Braun B, Moser M, Zhang G, Gini E, Keller U (1999) *J Opt Soc Am B* 16:376
- Stewart TC, Eliasmith C (2014) *Proc IEEE* 102:881
- Strogatz SH (2001) *Nature* 410:268
- Sysak M, Liang D, Jones R, Kurczveil G, Piels M, Fiorentino M, Beausoleil R, Bowers J (2011) *IEEE J Sel Top Quantum Electron* 17:1490
- Tait AN, Nahmias MA, Tian Y, Shastri BJ, Prucnal PR (2014a) Photonic Neuromorphic Signal Processing and Computing. In: Naruse M (ed) *Nanophotonic information physics. Nano-optics and nanophotonics*. Springer, Berlin/Heidelberg, pp 183–222
- Tait AN, Nahmias MA, Shastri BJ, Prucnal PR (2014b) *J Lightwave Technol* 32:3427
- Tait AN, Ferreira de Lima T, Nahmias MA, Shastri BJ, Prucnal PR (2016a) *Opt Express* 24:8895
- Tait A, Ferreira de Lima T, Nahmias M, Shastri B, Prucnal P (2016b) *IEEE Photon Technol Lett* 28:887
- Tait AN, Wu AX, de Lima TF, Zhou E, Shastri BJ, Nahmias MA, Prucnal PR (2016c) *IEEE J Sel Top Quantum Electron* 22:312
- Tait AN, de Lima TF, Zhou E, Wu AX, Nahmias MA, Shastri BJ, Prucnal PR (2017) *Sci Rep* arXiv:1611.02272
- Tank D, Hopfield J (1986) *IEEE Trans Circuits Syst* 33:533
- Tapson J, Cohen G, Afshar S, Stiefel K, Buskila Y, Hamilton T, van Schaik A (2013) *Front Neurosci* 7:153
- Taur Y (2002) *IBM J Res Dev* 46:213
- Taur Y, Buchanan D, Chen W, Frank D, Ismail K, Lo SH, Sai-Halasz G, Viswanathan R, Wann HJ, Wind S, Wong HS (1997) *Proc IEEE* 85:486
- The HBP Report (2012) Technical Report (The human brain project)
- Thorpe S, Delorme A, Rullen RV (2001) *Neural Netw* 14:715
- Tucker RS (2010) *Nat Photonics* 4:405
- Turconi M, Garbin B, Feyereisen M, Giudici M, Barland S (2013) *Phys Rev E* 88:022923
- Van Vaerenbergh T, Fiers M, Mechet P, Spuesens T, Kumar R, Morthier G, Schrauwen B, Dambre J, Bienstman P (2012) *Opt Express* 20:20292
- Van Vaerenbergh T, Alexander K, Dambre J, Bienstman P (2013) *Opt Express* 21:28922
- Vandoorne K, Dambre J, Verstraeten D, Schrauwen B, Bienstman P (2011) *IEEE Trans Neural Netw* 22:1469
- Vandoorne K, Mechet P, Van Vaerenbergh T, Fiers M, Morthier G, Verstraeten D, Schrauwen B, Dambre J, Bienstman P (2014) *Nat Commun* 5:3541 EP
- Verstraeten D, Schrauwen B, D'Haene M, Stroobandt D (2007) *Neural Netw* 20:391
- Vicsek T (2002) *Nature* 418:131
- Vlasov Y (2012) *IEEE Commun Mag* 50:s67
- von Neumann J (1993) *IEEE Ann Hist Comput* 15:27
- Wieczorek S, Krauskopf B, Lenstra D (1999) *Opt Commun* 172:279
- Wieczorek S, Krauskopf B, Lenstra D (2002) *Phys Rev Lett* 88:063901
- Wieczorek S, Krauskopf B, Simpson TB, Lenstra D (2005) *Phys Rep* 416:1
- Woods D, Naughton TJ (2012) *Nat Physics* 8:257
- Wünsche HJ, Brox O, Radziunas M, Henneberger F (2001) *Phys Rev Lett* 88:023901
- Xiong K, Xiao X, Hu Y, Li Z, Chu T, Yu Y, Yu J (2011) In: *Photonics and optoelectronics meetings (POEM)*, vol 8333, pp 83330A–83330A-7
- Xu Q, Fattal D, Beausoleil RG (2008) *Opt Express* 16:4309
- Yacomotti AM, Eguia MC, Aliaga J, Martinez OE, Mindlin GB, Lipsitch A (1999) *Phys Rev Lett* 83:292
- Yacomotti AM, Monnier P, Rainieri F, Bakir BB, Seassal C, Raj R, Levenson JA (2006a) *Phys Rev Lett* 97:143904
- Yacomotti AM, Rainieri F, Vecchi G, Monnier P, Raj R, Levenson A, Ben Bakir B, Seassal C, Letartre X, Viktorovitch P, Di Cioccio L, Fedeli JM (2006b) *Appl Phys Lett* 88:231107
- Yamada M (1993) *IEEE J Quantum Electron* 29:1330
- Yu Y, Baltus PG, Van Roermund AH (2011) *Integrated 60GHz RF beamforming in CMOS*. Springer Science & Business Media. Springer Dordrecht Heidelberg London, New York



## Quantum Computing

Viv Kendon  
JQC and Atmol, Department of Physics  
Durham University, Durham, UK

### Article Outline

Glossary  
Definition of the Subject  
Introduction  
Digital Quantum Computing  
What Is a Qubit? (Quantum Mechanics for Dummies)  
Quantum Gates  
How Do Qubits Give Us Cool Computing?  
How Quantum Computing Got Started (A Little Bit of History)  
Error Correction for Quantum Computers  
Programming a Quantum Computer  
Shor's Algorithm  
Quantum Walks  
How Powerful Is Quantum Computing?  
Ultimate Physical Limits to Computation  
Can We Build a Quantum Computer?  
Unconventional Extensions  
Cluster-State Quantum Computing  
Topological Quantum Computing  
What About Analog?  
Continuous-Variable Quantum Computing  
Hybrid Quantum Computing Architectures  
Continuous-Time Quantum Computing  
Adiabatic Quantum Computing and Quantum Annealing  
Quantum Walks (Reprise)  
Quantum Simulation  
Non-digital Implementations of Quantum Computing  
Biological Quantum Computing?  
Future Directions  
Bibliography

## Glossary

**Analog computing** Encoding into a continuous variable and processing in a continuous-time evolution.

**Anyons** A type of quantum particle that can only exist in two dimensions and that has exotic statistics when two identical particles are exchanged.

**Bit** A two-state classical system used to represent a binary digit, zero or one.

**Bus** A communications channel in computer architecture to provide a high-speed link between different elements of memory or processing registers.

**Bose-Einstein particles** Integer spin quantum particles like to be together: any number can occupy the same quantum state.

**Classical computing** How we compute using classical logic and conventional computational devices.

**Computability** What we can in principle compute with given physical resources (some things are uncomputable).

**Computational complexity** How fast we can compute with given physical resources (some things are harder to compute than others).

**Digital computing** Encoding into *bits* or *qubits* and processing using discrete gates.

**Entanglement** Quantum states can be more highly correlated than classical systems: the extra correlations are known as entanglement.

**Error correction** In a quantum context, fixing errors without disturbing the quantum superposition.

**Fermi-Dirac particles** Half-integer spin quantum particles like to be alone: only one such particle can occupy each quantum state at a given time.

**Quantum dense coding** Classical bits can be encoded two for one into *entangled* qubits for communications purposes.

**Quantum key distribution** Quantum mechanics allows for secure key distribution in the presence of eavesdroppers and noisy

environments. The keys can then be used for encrypted communication.

**Quantum teleportation** A method to transmit an unknown quantum state using only classical communications plus shared entanglement.

**Qubit** A two-state quantum system such as the spin of an electron or the polarization of a photon. More complex quantum particles (such as atoms) can be used as qubits if just two of their available states are chosen to represent the qubit.

**Qubus** A quantum version of a computer bus, the fast communications linking memory and processing registers.

**Scalable** A computer architecture designed from modular units that can be efficiently expanded to an arbitrary size.

**Squeezing** With a pair of complementary quantum observables, making one more uncertain so that the other can be measured more precisely.

**Quantum computing** Computation based on the laws of quantum mechanics for the allowed logical operations.

**Threshold result** In the context of quantum computation, this result says that error correction can work if the error rate is low enough.

**Tunneling** Quantum particles can get through barriers that classical particles remain stuck behind. If the barrier is not infinitely high, there is some probability for the quantum particle to be the other side of it even though it doesn't on average have enough energy to jump over the top.

**Unitary operations** How to control quantum systems while preserving quantum properties. Quantum systems evolving without any influence from environmental disturbance follow unitary dynamical evolution.

## Definition of the Subject

*Quantum computing* is computing that follows the logic of quantum mechanics. The quantum part of this subject is well specified after over a hundred years of development of the theory of quantum mechanics. Definitions of computing are fuzzier

and argued over by computer scientists and philosophers. For a physically motivated definition and framework, see Horsman et al. (2014). Here, we leave such discussions to the philosophers in favor of exploring what is possible when the constraints of classical logic are put aside, but the limitations of the physical world are kept firmly in hand.

Quantum computing is not simply computing that involves quantum effects, for that would be too broad to be useful. Since transistors and lasers exploit quantum properties of matter and light, most classical computers would thus be included. Yet it is worth remembering that definitions are never as clear-cut as we would like. Figuring out which quantum systems can be simulated efficiently by classical computers is an active area of current research, and pinning down the boundary between quantum and classical is the objective of ongoing experimental investigation.

## Introduction

As befits an account under the overall title “Unconventional Computing,” this is a somewhat distinctive view of *quantum computing*, adapted to the context in which you are likely to be reading it. To many, quantum computing is already unconventional computing, depending as it does on the elusive properties of matter at the scale of atoms, rather than the simple everyday experience of counting and deduction that underpins classical computation. Yet it is already an established field in which there is a standard approach and the possibility to be unconventional within a quantum computing context. This account is thus arranged into two halves, the first forming a rapid introduction to the standard digital version of quantum computing and the second expanding into a less well-charted territory beside and beyond.

In order to write about such an interdisciplinary subject for a broad audience, I have tried to keep everything at a level a nonspecialist can follow while not oversimplifying to the point of inaccuracy. Please be forgiving about the parts you could have written better yourself and inquisitive about the parts that may have something new for you.

## Digital Quantum Computing

Digital quantum computing developed a more or less parallel structure to classical digital computing, with a quantum version of each component. There are good reasons for this: many of the optimal features of classical digital computing carry over to quantum computing with little or no change. In order to explain why there is a fundamental advantage to using quantum logic for computation, I will explain the components step-by-step and then provide examples of how the parts combine into the whole.

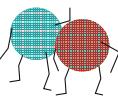
### What Is a Qubit? (Quantum Mechanics for Dummies)

Quantum computing exploits the logic of quantum mechanics, which is notoriously tricky to understand. The mathematical structure is simple and linear, so treating it as “maths we happen to be able to build” is one option for getting to grips with it. Here, I’ve chosen the pictorial intuition approach, with some cartoon qubits to help explain how they behave. Those already familiar with quantum mechanics can safely skip over this section. Those still bemused by the end of it may like to sample from the wide range of available textbooks to find an approach which suits them best.

Classical computers use *bits* as their basic unit; any classical system with two states can represent a single bit. Tossing a coin, with the resulting “heads” or “tails,” is an example of a two-state classical system. In computers, the two states are usually labeled “0” (zero) and “1” (one). Quantum computers use quantum bits, usually called *qubits*, as their basic unit, which are two-state quantum systems, and the two states are also usually labeled zero and one. Examples of physical systems with two quantum states include electron spin, photon polarization, and phosphorus nuclear spin. These have all been used in experiments aimed toward building a quantum computer. In a quantum computer, we usually localize or otherwise separate individual qubits so they are distinguishable from each other, and there are thus no Fermi or Bose statistics to complicate the picture.

We revisit this aspect of quantum mechanics briefly in the section on [Topological Quantum Computing](#).

Here are a couple of cartoon qubits



to help explain their quantum properties. They are wriggly fidgety little things; they move so fast you can’t tell which way up they are just by looking. If you reach down to grab hold of one, the only way you can get a grip is by an arm or a leg. First, we have to choose our basis states, labeled  $|0\rangle$

and  $|1\rangle$ . We write the zero and one in those

funny brackets (“kets”) to remind us they are quantum states rather than classical bits. To simplify notation, we will sometimes write the state of several qubits together inside one ket thus,  $|00\rangle \equiv |0\rangle|0\rangle$ . In terms of the cartoon qubits, this means we decided to label “grabbed by the arm” as the zero state and “grabbed by the leg” as the one state. The key quantum property is that qubits can fidget around anywhere in between the zero and one states, in what is called a superposition state, for example,  $(|0\rangle + |1\rangle)/2$

If you reach down and grab a qubit in one of these superposition states, you are equally likely to grab an arm or a leg, indicating zero or one in a basis state. So you can’t tell the qubit was in a superposition state by measuring it, the outcome is always a basis state. What’s more, grabbing it by the arm hauls it onto its feet, and grabbing it by the leg shoves it into a handstand. So after measuring, what you found is what you now have, regardless of what it was before. Luckily, it is possible to do other things to qubits besides measuring them.

The most general state of a single qubit can be written  $\alpha|0\rangle + \beta|1\rangle$  with  $|\alpha|^2 + |\beta|^2 = 1$ , and  $\alpha, \beta$  are complex numbers. The probability of measuring zero is given by  $|\alpha|^2$  and the probability of measuring one is given by  $|\beta|^2$ . Why complex numbers? Because it makes the maths work conveniently. If you don’t like complex numbers, there are other ways to formulate quantum mechanics, but it isn’t as elegant. To keep things

simple, I use examples that avoid complex numbers as far as possible.

Cartoon qubits with limbs don't quite capture all the subtle quantum effects, but they are a useful image to keep in mind if you are new to these concepts. This account also ignores all of the more fundamental issues to do with the interpretation of quantum mechanics, especially those generally referred to as "the quantum measurement problem."

Now, consider two qubits (traditionally belonging to Alice and Bob) in the state  $(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B)/\sqrt{2}$ . This is an example of an entangled state: the two qubits are completely correlated. Suppose Alice measures her qubit: if she finds  $|0\rangle_A$ , then she knows Bob will find  $|0\rangle_B$  when he measures his qubit. If she finds  $|1\rangle_A$ , then she knows Bob will find  $|1\rangle_B$ . Alice can't communicate anything to Bob this way because she can't control whether she will find  $|0\rangle$  or  $|1\rangle$  when she measures. So after they have both measured their qubits, they now share one random bit, which is a resource they can use for other tasks.

The purely quantum feature of entanglement that cannot be reproduced with suitably correlated classical systems is that Alice and Bob can use any basis states they like, and the perfect correlation still appears. For example,  $(|0\rangle + |1\rangle)/\sqrt{2}$  and  $(|0\rangle - |1\rangle)/\sqrt{2}$  are another possible pair of basis states: the rule is they must be orthogonal to each other. This is the equivalent of changing the orientation of your classical coordinate system. However, Alice and Bob do both have to use the same basis: for a thorough discussion of the consequences of this and other reference frames in quantum mechanics, see Bartlett et al. (2006).

## Quantum Gates

Now that we have qubits to make a quantum register for our quantum computer and measurements to read out the state of the quantum register, we need some quantum gates to perform our calculation with. There are two ways to change a

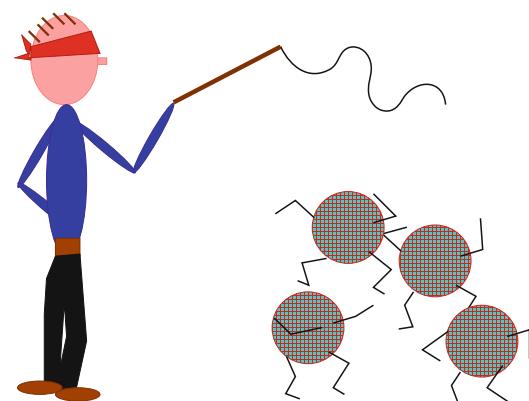
quantum state: measurements, which are what happen when you observe the state of a quantum system, and unitary dynamics, which is what quantum systems do when no one is looking. Happily, it is possible to steer a quantum system without looking at it (see Fig. 1), so we can make it evolve the way we choose through applying *unitary operations*. Unitary operations are reversible; the simplest example, acting on a single qubit, is the Hadamard gate  $H$ . We can define what it does by the effect on basis states:

$$\begin{aligned} H|0\rangle &= (|0\rangle + |1\rangle)/\sqrt{2} \\ H|1\rangle &= (|0\rangle - |1\rangle)/\sqrt{2}, \end{aligned} \quad (1)$$

and of course it can also act on superposition states,

$$\begin{aligned} H(\alpha|0\rangle + \beta|1\rangle) &= \frac{1}{\sqrt{2}}\{(\alpha + \beta)|0\rangle + (\alpha - \beta)|1\rangle\}. \end{aligned} \quad (2)$$

The Hadamard gate rotates the qubit through an angle of  $\pi/2$ , i.e.,  $90^\circ$ , from upright to lying on its side, for example. Notice also that applying the Hadamard gate twice gets you back where you started,  $H(H|\psi\rangle) = |\psi\rangle$ ; the Hadamard gate is its own inverse. The pictorial intuition doesn't quite work here; two rotations through  $90^\circ$  ought to make the qubit upside down. Remember the qubits are wriggly things, they manage to spin



**Quantum Computing, Fig. 1** Unitary evolution: controlling qubits without looking at them

round before you apply the second Hadamard, so it does flip them back where they started. If you look at the maths, it is due to the minus sign when the Hadamard gate acts on the  $|1\rangle$  state. More generally, rotation gates can rotate through any other angle, for example,

$$\begin{aligned} R_\theta|0\rangle &= \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle \\ R_\theta|1\rangle &= \sin(\theta/2)|0\rangle - \cos(\theta/2)|1\rangle \end{aligned} \quad (3)$$

Thanks to that minus sign, these rotation gates are also their own inverse.

To carry out computations, we also need gates that act on 2 qubits at once, for example, the controlled not (CNOT) gate:

$$\begin{aligned} C|0\rangle_c|0\rangle_t &= |0\rangle_c|0\rangle_t \\ C|0\rangle_c|1\rangle_t &= |0\rangle_c|1\rangle_t \\ C|1\rangle_c|0\rangle_t &= |1\rangle_c|1\rangle_t \\ C|1\rangle_c|1\rangle_t &= |1\rangle_c|0\rangle_t \end{aligned} \quad (4)$$

where the first qubit (labeled  $c$ ) is the control and remains unchanged, while the second qubit (labeled  $t$ ) is the target and is flipped if the control qubit is in state  $|1\rangle$ .

Just as with classical computation, small sets of universal quantum gates exist from which any computation can be constructed efficiently (Barenco et al. 1995). A particular set is usually chosen to suit the physical components of the quantum computer. There are many possible choices; the gates described above are just a few of the 1- and 2-qubit gates available.

## How Do Qubits Give Us Cool Computing?

We now have qubits to form a quantum register, quantum gates to perform our computation, and measurements to read out the result; the next step is to put them all together. We begin with an example, which illuminates the basic idea much more clearly than a general description. The Deutsch-Jozsa algorithm (Deutsch and Jozsa 1992) is one of the earliest quantum algorithms; it solves the following promise problem. We are promised that the function  $f(x) : x \mapsto \{0, 1\}$  is either:

- Balanced, i.e., the number of times  $f(x)$  outputs zero is equal to the number of times  $f(x)$  outputs one, compared over all possible input values  $x$  (with  $0 \leq x < N$ ).
- $f(x)$  is constant, i.e., the output is always either zero or one (but we don't know which) for any input  $x$ .

The problem is to find out which, balanced or constant,  $f(x)$  is. The cost of this computation is measured in "queries," in this case, evaluations of the function  $f(x)$ . The evaluation of  $f(x)$  is given to us by an "oracle." Given a number  $x$ , the oracle returns the value, zero or one, of  $f(x)$ . (Oracles are a tool for thought experiments in computer science: they allow us to focus on the problem without having to account for how  $f(x)$  itself is specified.)

To solve this problem classically, you have to use the oracle to examine values of  $f(x)$  for one value of  $x$  at a time and compare them. As soon as you accumulate both a zero and a one in the set of answers, you know (because of the promise) that  $f(x)$  is balanced. But you can't be sure it is constant until you accumulate more than half of the answers and see that they are all the same. So the best possible classical algorithm must take at least two queries and could require up to  $N/2 + 1$  queries.

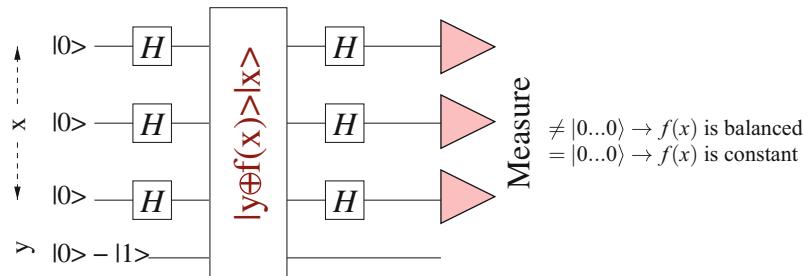
The circuit diagram in Fig. 2 shows how to solve this problem using a single query with a quantum computer. There is one register of  $n = \lceil \log_2 N \rceil$  qubits, i.e., large enough to hold the input  $x$ , and one extra single qubit  $|y\rangle$ . We start with all the qubits in the  $|0\rangle$  state. The first step is the set of Hadamards applied to the  $|x\rangle$  register:

$$\begin{aligned} H^{\otimes n}|0\dots 00\rangle &= (|0\dots 00\rangle + |0\dots 01\rangle \\ &\quad + |0\dots 10\rangle + \dots + |1\dots 11\rangle) \times /2^{-n/2} \end{aligned}$$

which is a superposition of all numbers from 0 to  $2^n - 1$ . We then make our one query to the oracle that computes  $f(x)$  for us, using the superposition of all possible inputs stored in  $|x\rangle$ . We get back a superposition of all the answers, which we store by adding it (bitwise) to the  $|y\rangle$  qubit, giving  $\sum_x |y \oplus f(x)\rangle|x\rangle$ .

**Quantum Computing,**  
**Fig. 2** Circuit diagram for the Deutsch-Jozsa algorithm for  $N < 8$ . See text for details.

Normalization factors have been omitted from the qubit states



Remember, we cannot detect a superposition by measurement alone, so the superposition of all the possible answers doesn't help us yet. We need a trick that will allow us to detect what sort of superposition of answers we got back. Before adding the answer to  $|y\rangle$ , we prepare this qubit in the state  $(|0\rangle - |1\rangle)/\sqrt{2}$ . This can be done by flipping it from  $|0\rangle$  to  $|1\rangle$  then applying a Hadamard gate. Then, when we add the answers to  $|y\rangle$ , there are three possibilities:

1. Constant: answers all  $|0\rangle : |y \oplus f(x)\rangle|x\rangle = (|0\rangle - |1\rangle)|x\rangle/\sqrt{2} = \text{unchanged.}$
2. Constant: answers all  $|1\rangle : |y \oplus f(x)\rangle|x\rangle = (|1\rangle - |0\rangle)|x\rangle/\sqrt{2} = \text{minus what it was.}$
3. Balanced: answers half  $|0\rangle$  and half  $|1\rangle : |y \oplus f(x)\rangle|x\rangle = \text{something else.}$

The second sequence of Hadamard gates then attempts to undo the first set and convert  $|x\rangle$  back to the all-zero state. In the first two cases, this succeeds (we can't detect the minus sign when we measure), but in the third case, the Hadamards don't get us back where we started so there will be some ones in the  $|x\rangle$  qubits which we find when we measure them. So, if we measure all zeros for  $|x\rangle$ , then  $f(x)$  is constant, while if we measure any ones, then  $f(x)$  is balanced. Thus, the quantum algorithm is fundamentally more efficient, requiring only the one evaluation of  $f(x)$ .

Feynman (1982) and independently to Deutsch (1985). They both observed that quantum systems are like a parallel computer (c.f. Feynman paths or many worlds, respectively). It remained a cute idea for the next decade, until a key result showing that *error correction* is theoretically possible implied it might be feasible to build one that really works (Knill et al. 1996; Aharonov and Ben-Or 1996; Steane 1996). Around the same time, Shor (1995) found an algorithm for factoring that could in theory break crypto schemes based on products of large primes being hard to factorize. At this point, the funders sat up and took notice. Not just because factoring could become easier with a quantum computer. There are public key crypto schemes based on other "one-way" functions that are hard to compute given the public key but easy to compute from the private key. By shifting the boundary between what's easy and what's hard, quantum computing threatens to break other classical one-way functions, too.

At the same time, *quantum communications* protocols were being developed. In particular, *quantum key distribution* sidesteps the crypto problem created by Shor's algorithm (Bennett and Brassard 1984), by providing a method to share private keys with the security guaranteed by the laws of quantum mechanics. Quantum resources can double the channel capacity, a technique known as *quantum dense coding* (Bennett and Wiesner 1992), and *quantum teleportation* can transmit a quantum state through a classical channel (Bennett et al. 1993) with the help of some shared entanglement. Commercial quantum communications devices are available now for key distribution (see, e.g., ID Quantique). They work over commercial fiber optics, they work well enough to reach from earth to satellites, and handheld devices are under

## How Quantum Computing Got Started (A Little Bit of History)

The idea that quantum mechanics could give us fundamentally faster computers first occurred to

development (Chun et al. 2016). Countries are starting to invest in quantum communications infrastructure, notably China (Courtland 2016). It remains a niche market, but the successful technology in a closely related quantum information field has helped to keep the funding and optimism for quantum computers buoyant.

## Error Correction for Quantum Computers

The most important result that allowed quantum computing to move from being viewed as an obscure piece of theory to a future technology is that error correction can work to protect the delicate quantum coherences (those signs, or phases, between different components of the superposition). Usually, the ubiquitous environmental disturbances randomize them very rapidly. Quantum systems are very sensitive to noise, which is why we inhabit a largely classical world.

Error correction is not so easy for quantum systems, because we are not allowed to examine their state without disturbing it. Thus we can't see directly whether they have acquired errors. Instead, we can use other quantum systems to check for errors. The main method for doing this is conceptually simple: logical qubits are encoded in a larger physical quantum system, and the extra degrees of freedom are used for error correction. A very simple example: for every qubit in your computation, use three entangled qubits. If an error occurs on a single qubit, the remaining two will give you the correct answer. This is illustrated in Fig. 3. If errors hit two of the three qubits, the majority outcome will be wrong, though, so this code is only good if the error rates are very low.

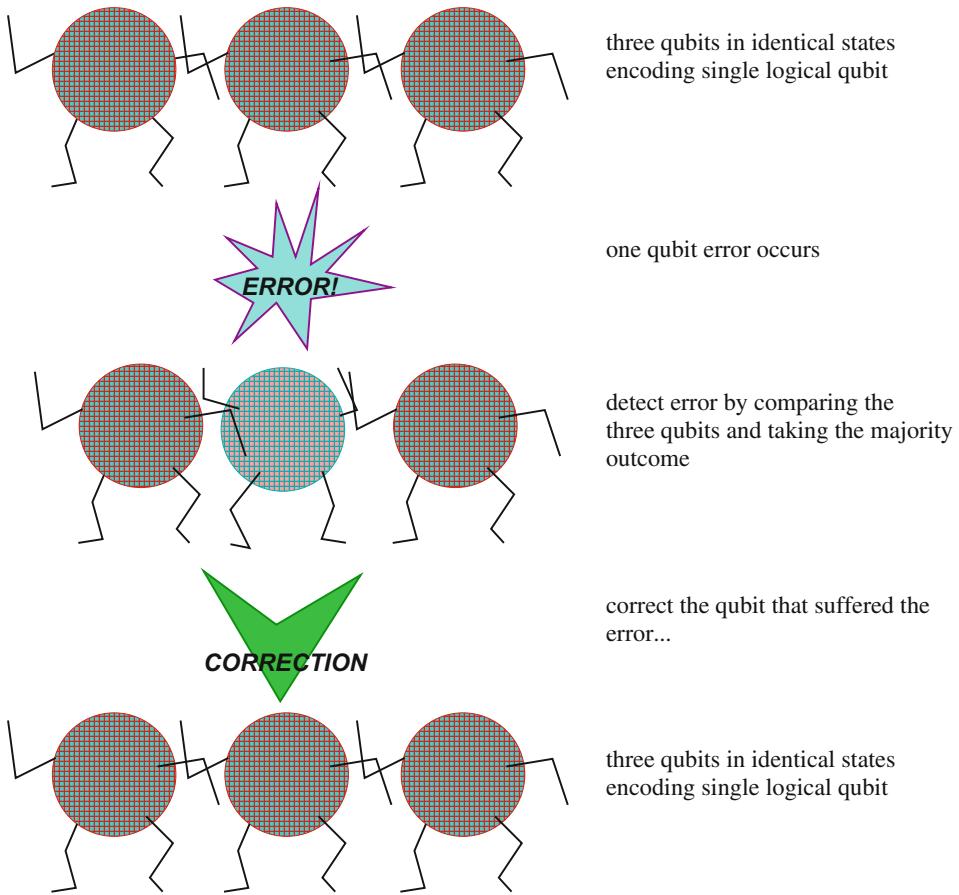
What is not at all obvious is that this has any chance of working. Adding extra degrees of freedom (e.g., more qubits) adds more places where errors can occur. Adding yet more qubits to correct these extra errors brings in yet more errors and so on. Nonetheless, if the error rate is low enough, even the errors arising from the error correction can be corrected fast enough (Knill et al. 1996; Aharonov and Ben-Or 1996). This is known as the *threshold result*. The details of exactly how to

accomplish this are considerable and employ many techniques from classical error-correcting codes (Shor 1995; Steane 1996). For those wanting to know more, there are plenty of good references and tutorials available (e.g., Lidar and Brun 2013; Devitt et al. 2009; Paler and Devitt 2015).

The crucial factor is thus what the actual threshold error rate is. It depends on the hardware, so it can only be estimated based on current experiments. Recent estimates suggest typical values are around  $10^{-2}$ – $10^{-4}$  errors per qubit per quantum operation, depending on the architecture (Nickerson et al. 2014). This is demanding but achievable for most physical systems that have been proposed for quantum computers.

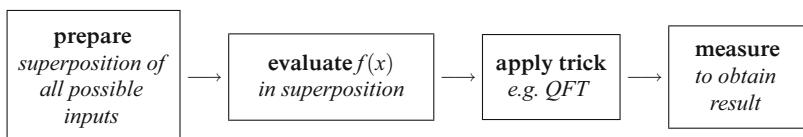
## Programming a Quantum Computer

The general structure of many quantum algorithms follows the same pattern we already saw in the Deutsch-Jozsa algorithm. This is illustrated in Fig. 4. Each algorithm needs to have its own trick to make it work, so progress on expanding the number of quantum algorithms has been relatively slow. Sometimes the same trick will work for a family of similar problems, but the number of distinct types remains limited. We have already seen an example of a promise problem in the Deutsch-Jozsa algorithm, where the Hadamard gates provide the trick. Shor's algorithm and many variants on this method have essentially similar structure based on quantum versions of Fourier transforms, while Grover's search and quantum walks have a distinctly different way of iterating toward the desired result. For a recent overview of quantum algorithms, see Montanaro (2015). Quantum simulation, the earliest suggested application for quantum computation, is in a different category altogether, and we will return to it near the end of the chapter, once we have all the tools necessary to understand how it works. There are also quantum versions of almost anything else computational you can think of, such as quantum game theory (Flitney and Abbott 2002) and quantum digital signatures (Gottesman and Chuang 2001; Collins et al. 2016). Most of these other types of quantum information processing fall into the category of communications protocols, where the



**Quantum Computing, Fig. 3** Error correction using three qubits to encode one logical qubit. Encoding, detecting, and correcting the error is done using CNOT gates without destroying the encoded quantum data

**Quantum Computing,**  
**Fig. 4** General structure of  
many quantum algorithms



advantage is gained through quantum entanglement shared across more than one location. To consolidate our understanding of the functioning of quantum algorithms, we will briefly outline Shor's algorithm and then give an overview of quantum walks.

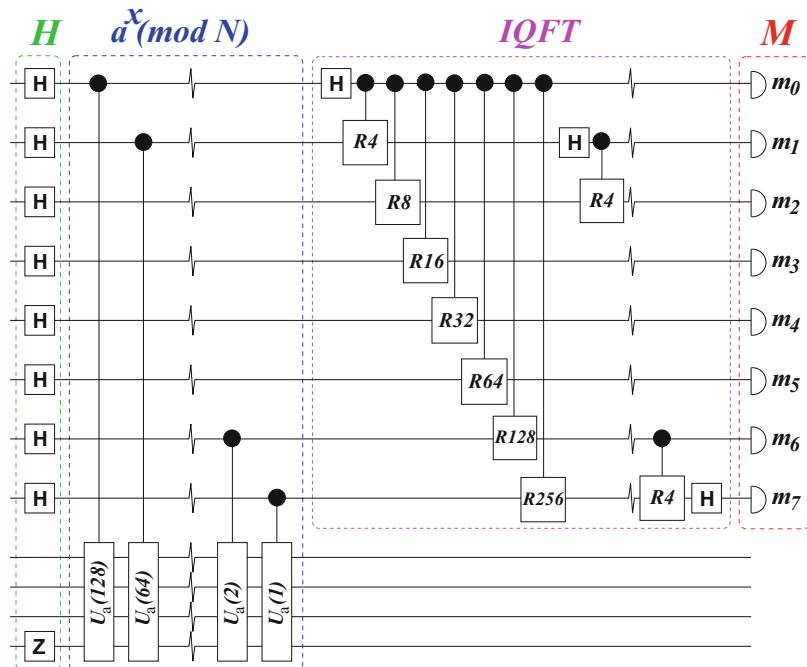
### Shor's Algorithm

The task in Shor's algorithm (Shor 1994, 1997) is to find a factor of a number  $N = pq$  where  $p$  and

$q$  are both prime. First, choose a number  $a < N$  such that  $a$  and  $N$  share no common factors. If  $a$  turns out not to be co-prime to  $N$ , then the task is done. This can be checked efficiently using Euclid's algorithm but only happens in a very small number of cases. Then run the quantum computation shown in Fig. 5 to find  $r$ , the periodicity of  $a^x \pmod{N}$ . The first half of the computation creates a superposition of  $a^x \pmod{N}$  for all possible inputs  $x$  up to  $N^2$ . This upper limit is not essential; it determines the precision

### Quantum Computing,

**Fig. 5** Circuit diagram for Shor's algorithm for factoring shown for the example of factoring 15. **H** is a Hadamard gate,  $R_n$  is a rotation by  $\pi/n$ , and the **U** gates perform the modular exponentiation (see text). **IQFT** is inverse quantum Fourier transform, and **M** is the final measurement



of the output and thus the number of times the computation has to be repeated, on average, to succeed. Shor's first insight was that this modular exponentiation step can be done efficiently. The answer is then contained in the entanglement between qubits in the upper and lower registers (Nielsen and Chuang 1996). Shor's second insight was that a quantum version of the familiar discrete Fourier transform can be used to rearrange the state so that measuring the upper register allows one to calculate the value of  $r$  with high probability. Once we have found  $r$ , then  $a^{r/2} \pm 1$  gives a factor of  $N$  (with high probability). If the first attempt doesn't find a factor, repeating the computation a few times with different values of  $a$  rapidly increases the chances of success.

The classical difficulty of factoring is not known, but the best classical algorithms we have are sub-exponential (exponential to some slower-than-linear function of  $N$ ), whereas Shor's algorithm is polynomial in  $N$ . This is roughly speaking an exponential improvement, and it promises to bring factoring into the set of “easy-to-solve” problems (i.e., polynomial resources) from the “hard-to-solve” set (exponential resources).

There is a whole family of problems using the same method, all based around identifying a hidden subgroup of an abelian group. Some extensions have been made beyond abelian groups, but this is in general much harder (for a review, see Lomont (2004)).

### Quantum Walks

Classical random walks underpin many of the best classical algorithms for problems like factoring and graph isomorphism. Markov chains and Monte Carlo techniques are widely used in simulation and other sampling problems. Finding a faster quantum version of a random walk is thus a good bet for creating a new type of quantum algorithm. A simple recipe for a quantum walk on a line goes as follows:

1. Start at the origin,  $|\psi(0)\rangle = |0\rangle_x$ .
2. Toss a qubit (quantum coin):

$$\begin{aligned}\mathbf{H}|0\rangle &\rightarrow (|0\rangle + |1\rangle)/\sqrt{2} \\ \mathbf{H}|1\rangle &\rightarrow (|0\rangle - |1\rangle)/\sqrt{2}\end{aligned}$$

3. Move *left* and *right* according to qubit state:

$$\begin{aligned} S|x, 0\rangle &\rightarrow |x - 1, 0\rangle \\ S|x, 1\rangle &\rightarrow |x + 1, 1\rangle \end{aligned}$$

4. Repeat steps 2 and 3  $T$  times.

5. Measure position of walker,  $-T \leq x \leq T$

If we repeat steps 1–5 many times, we get a probability distribution  $P(x, T)$  (see Fig. 6), which spreads quadratically faster than a classical random walk.

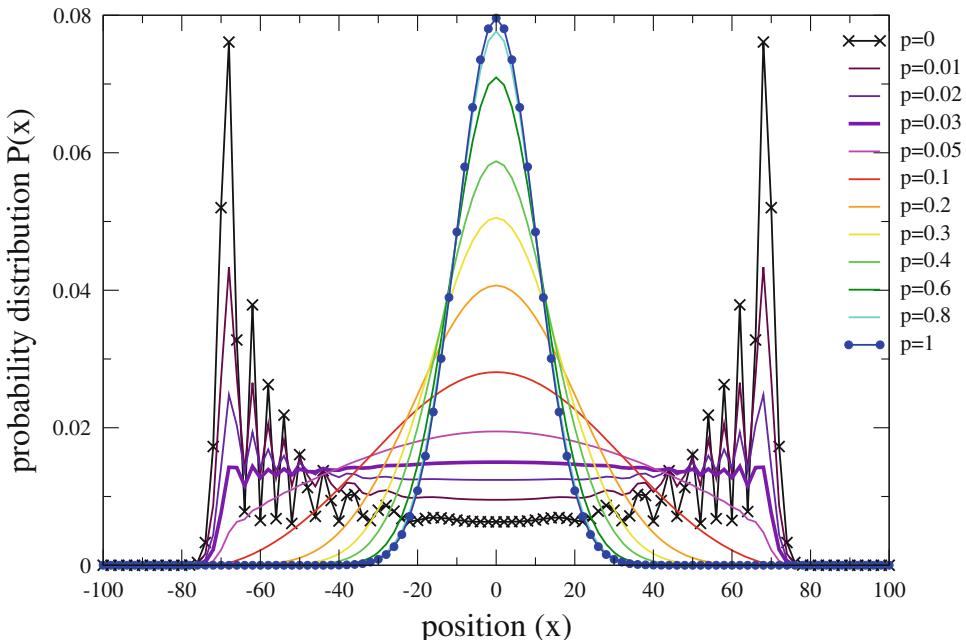
If we add a little decoherence, which is equivalent to measuring the quantum walker with probability  $p$  at each step, then we get a top-hat distribution for just the right amount of noise (Kendon and Tregenna 2003), see Fig. 6. Top hat distributions are useful for uniform random sampling, which has many algorithmic applications. For related results of mixing times on cycles and tori, see Aharonov et al. (2001) and Richter (2007a, b).

Quantum walks can be used to find a marked item in an unsorted database of  $N$  items. In these days of internet searching where everything is indexed, this doesn't seem like a hard problem,

but it is an important computational task from which more complex algorithms can be built. Classical solutions have to check on average  $N/2$  items and in the worst case need to check them all. This problem was investigated by Grover (1996, 1997), who found a quantum algorithm that is quadratically faster than classical searching. His algorithm doesn't use quantum walks, but Shenvi et al. (2003) showed that quantum walks can perform the task just as well. They used a special symmetric coin, called a *Grover coin*, that is like a quantum equivalent of a diffusion operator. The recipe for quantum walk searching is:

1. Start in a uniform distribution over a graph with  $N$  nodes.
2. Use a *Grover coin* everywhere except the marked node (\*).
3. Run for approx  $\frac{\pi}{2} \sqrt{N}/2$  steps.
4. Walker will now be at the marked node with high probability.

(\*) Applying the Grover operator conditionally on the node not being the one we are searching for is



**Quantum Computing, Fig. 6** Comparison of a quantum walk ( $p = 0$ ) of 100 steps with the corresponding classical random walk ( $p = 1$ ). Progressively increasing the

decoherence produces a “top-hat” distribution for the right choice of decoherence probability  $p$

done by using a controlled quantum operation that tests for the marked node and only applies the Grover operator where it doesn't match the test. This is done in superposition, so it doesn't let us find the marked node directly. Quantum walk searching is like the inverse of starting at the origin and trying to get a uniform (top-hat) distribution. A quadratic speedup is the best that can be achieved by any algorithm tackling this problem, as proved by Bennett et al. (1997).

The quantum walk version of Grover's search has been generalized to find more than one item. Magniez et al. (2003, 2005) show how to detect triangles in graphs, Ambainis (2004) applies quantum walks to deciding element distinctness, and Childs and Eisenberg (2005) generalize to finding subsets. These generally obtain a polynomial improvement over classical algorithms, for example, reducing the running time to  $O(N^{2/3})$  compared to  $O(N)$  for classical methods.

There is one known problem for which a quantum walk can provide an exponential speedup. The “glued trees” problem (Childs et al. 2003) is also about finding a marked state, but this time the starting point is also specified, and an oracle is available to give information about the possible paths between the start and finish. For a short review of quantum walk algorithms, see Ambainis (2003), and for a more comprehensive review of quantum walks in general, see Venegas-Andraca (2012).

## How Powerful Is Quantum Computing?

What we've seen so far is quantum computers providing a speedup over classical computation, i.e., new *complexity classes* to add to the zoo (Aaronson 2012), but nothing outside of the classical *computability* limits. The mainstream view is that quantum computing achieves the same *computability* as classical computing, and the quantum advantage may be characterized completely by complexity comparisons. The intuitive way to see that this is likely to be correct is to note that, given the mathematical definition of quantum mechanics, we can always simulate quantum dynamics using classical digital computers, but we generally can't do it efficiently. The chink

in this argument is that quantum mechanics is not digital. Qubits can be in an arbitrary superposition  $\alpha|0\rangle + \beta|1\rangle$ , where  $|\alpha|^2$  can take any value between zero and one. Thus, we can only simulate the quantum dynamics to within some precision dictated by the amount of digital resources we employ. This corresponds to the fact that we cannot measure  $\alpha$  directly in an experiment: we can only infer an approximate value for  $\alpha$  from the various measurements we do make. Furthermore, classical computers also have continuous values for quantities such as voltages, which we can use for analog computation, but again with precision limits to what we can measure. Hence, it isn't clear there could be any fundamental difference between classical and quantum in this respect that could separate their computational ability.

The practical version of the question is, will quantum computers give us an advantage over classical computers? How many qubits do we need to make a useful quantum computer? This depends on what we want to do:

**Simulating a quantum system:** For example,  $N$  two-state particles have  $2^N$  possible different states, and we can map this onto  $N$  qubits. The state of the system could be in superposition of all of these  $2^N$  possible states. Compare this with the memory needed for classical simulations, which use one complex number per state. This needs  $2^{\{N+1\}} \times$  the size of one floating point number, which means 1 Gbyte can store the state of just  $N = 26$  two-state systems. The current record is  $N = 36$  in 1 terabyte of storage (De Raedt et al. 2007) – each additional particle doubles the memory required. Thus, more than 40 or so qubits are beyond current classical resources, to carry out the computation if not store the data. Evolving the quantum state to calculate something useful takes even longer, often scaling as the square or fourth power of the size of the state. In the cases where we don't need to keep track of all the possible superpositions, e.g., if only nearest neighbor interactions are involved, larger classical simulations can be performed; for example, see Verstraete et al. (2004) and Wang et al. (2011). However, such simulations are limited to a subspace no larger than the equivalent of about 36 qubits, no matter how many particles are actually being simulated.

**Shor's factoring algorithm:** The best classical factoring to date (Kleinjung et al. 2010) is of a 232-digit numbers which is approximately 768 bits (RSA-768). Shor's quantum algorithm needs  $2n$  qubits in the QFT register plus  $5n$  qubits for the modular exponentiation (lower register plus ancilla qubits), a total of  $7n$  logical qubits. A 768-bit number therefore needs 5,376 logical qubits. We now need to take account of error correction. Again, this depends on the physical quantum computer and the error rates that have to be corrected. If the error rate is close to the threshold of  $10^{-3}$ – $10^{-4}$ , then more error correction is needed. For low error rates, maybe 20–200 physical qubits per logical qubit are required. For high error rates, the numbers blow up quickly to maybe  $10^5$  physical qubits per logical qubit. This suggests that factoring won't be the first useful application of quantum computing, despite being the most famous; we may need tera-qubit quantum computers to produce useful results here. Although the scaling favors quantum computing, the crossover point is very high.

## Ultimate Physical Limits to Computation

The physical limits to computation is a topic that is revisited regularly. The version by Lloyd (2000) is the best known of these calculations that has an explicitly quantum flavor. Physical limits can be deduced from these fundamental constants:

$$\text{The speed of light : } c = 3 \times 10^8 \text{ ms}^{-1}$$

$$\text{Boltzmann's constant : } k = 1.38 \times 10^{-23} \text{ JK}^{-1}$$

$$\text{The gravitational constant : }$$

$$G = 6.67 \times 10^{-11} \text{ m}^3\text{kg}^{-1}\text{s}^{-2}$$

$$\text{Planck's constant : } \hbar = 1.055 \times 10^{-34} \text{ Js}$$

Consider 1 kg of matter occupying a volume of 1 litre =  $10^{-3} \text{ m}^3$  – comparable with a laptop. How fast can this matter compute? Quantum mechanics tells us that the speed of computation is limited by its average energy  $E$  to  $2E/\pi\hbar = 5 \times 10^{50}$  operations per second with  $E = Mc^2$  given by the mass of 1 kg. This is because the time-energy Heisenberg uncertainty principle,

$\Delta E \Delta t \geq \hbar$ , says that a state with a spread in energy of  $\Delta E$  takes a time  $\Delta t$  to evolve into a different state. This was extended by Margolus and Levitin (1996, 1998) to show that a state with average energy  $E$  takes a time  $\Delta t \geq \pi\hbar/2E$  to evolve to a different state.

The number of bits it can store is limited by its entropy,  $S = k \ln W$ , where  $W$  is the number of states the system can be in. For  $m$  bits,  $W = 2^m$ , so  $S \simeq km$ , and the system can store at most  $m$  bits. Our estimate of  $m$  is thus dependent on what our laptop is made of. It also depends on the energy  $E$ , since not all of the  $W$  possible states have the same energy, so in practice we don't get the maximum possible  $W = 2^m$  number of possible states, since we decided our laptop had a mass energy of 1 kg. In practice, this restriction doesn't change the numbers very much. There are various ways to estimate the number of states, one way is to assume the laptop is made up of stuff much like the early universe: high-energy photons (the left-over ones being the cosmic microwave background radiation). This gives around  $2 \times 10^{31}$  bits as the maximum possible memory.

Having noted that entropy  $S$  is a function of energy  $E$ , we can deduce that the number of operations per second per bit is given by  $2Ek/\pi\hbar S \propto kT/\hbar$ , where  $T = \partial S/\partial E$  is the effective temperature of the laptop. For our 1 kg laptop, this means the number of operations per bit is about  $10^{19}$ , and the temperature is about  $5 \times 10^8 \text{ K}$ , a very hot plasma. This also implies the computer must have a parallel type of operation, since the bit flip time is far smaller than the time it takes for light to travel from one side to the other. If you compress the computer into a smaller volume to reduce the parallelism, it reaches serial operation at the black hole density! The evaporation rate for 1 kg black hole is  $10^{-19} \text{ s}$ , after  $10^{32}$  operations on  $10^{16}$  bits. This is very fast, but otherwise similar in size to conventional computers. Prototype quantum computers already operate near these limits, only with most of their energy locked up as mass.

The salutary thing about these calculations is that once you scale back to a computer in which the mass isn't counted as part of the available energy, we are actually close enough to the physical limits we have to take notice of them.

Computational power has been increasing exponentially for so long we have become used to it doubling every few years and easily forget that it also requires increasing resources to give us this increased computational capacity. Based on current physics, there isn't some vast reservoir of untapped computational power out there waiting for us to harness as our technology advances, and we are already seeing a transition to a significantly different regime, which, one may argue, highlights the importance of unconventional computation, where further increases in speed and efficiency may still be available after the standard model has run out of steam.

## Can We Build a Quantum Computer?

The short answer to the question of whether we can actually build a quantum computer is “yes.” People already have built small ones, but we haven’t yet built one big enough to be useful for something we can’t already calculate more easily with our classical computers, although current experiments are getting very close to this threshold (Bernien et al. 2017). The notion of *scalability* is important in the quest to build a useful quantum computer. We need designs that can be thoroughly tested at small sizes, registers of a few qubits, and then scaled up without fundamentally changing the operations we tested at small scales. The most feasible architecture for achieving this is using small, repeatable, connectable units (Metodi et al. 2005; Nickerson et al. 2014).

Many models for scalable quantum computers are hybrid, for example, combining stationary and flying qubits. Flying qubits are usually photons, while stationary qubits can be almost any other type that can interact with photons, such as atoms or quantum dots. The motivation for this model is that it can be hard to do everything in one type of physical system. For example, 1-qubit gates are easy to apply to photons, while 2-photon gates are impossible without using some other type of matter to create the interaction. Atoms or atomic ions in traps, combined with photons, are currently seen as the best scalable architecture, but this is by no means the only player in the game.

Superconducting qubits are also looking very promising, with demonstrator multi-qubit devices, e.g., at IBM (currently 16 qubits), and silicon-based qubits (quantum dots) are not far behind. The vision is driving beautiful experiments, and the field is moving rapidly; to find out up-to-date information on current progress, visit the websites of the major experimental collaborations.

## Unconventional Extensions

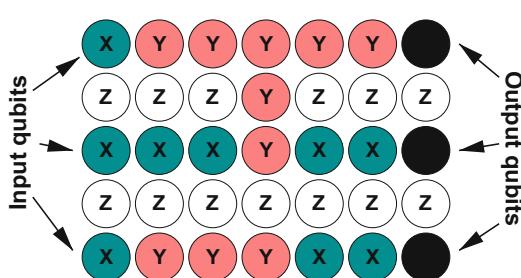
Quantum computing as just described mimics classical digital computing quite closely in terms of bits (qubits), gates, error correction, scalable architectures, and so on. This is partly because the concepts apply across both classical and quantum computing. This part of the picture is also the easiest to explain to those familiar with classical computing. In reality, this view is not justified by either the historical development or the creativity with which theorists and experimentalists are tackling the difficult task of trying to build a working quantum computer. I will next attempt to set the record straight and hint at some of the wilder territory beyond.

## Cluster-State Quantum Computing

First proposed by Raussendorf and Briegel (2001), and also known as “one-way quantum computing” or “measurement-based quantum computing,” cluster-state quantum computing has no direct classical equivalent, yet it has become so popular it is currently almost the preferred architecture for development of quantum computers. Originally designed for atoms in optical lattices, the idea is to entangle a whole array of qubits using global operations and then perform the computation by measuring the qubits in sequence. This architecture trades space resources for a temporal speedup, requiring roughly  $n^2$  physical qubits where the gate model uses  $n$ . The measurement sequence can be compressed to allow a parallel operation for all but a few measurements that are determined by previous measurement outcomes and require

the outcomes to be fed forward as the computation proceeds. One obvious prerequisite for this architecture is qubits that can be measured quickly and easily. The measurement is often slower than other gate operations, so this does limit in practice which types of qubits can be used.

Since the dependent measurements proceed from one side of the lattice to the other (see Fig. 7), instead of making the whole array in one go, the qubits can be added in rows as they are needed (Horsman et al. 2011; Brown et al. 2012). This also allows the use of a probabilistic method for preparing the qubits, which can be repeated until it succeeds in a separate process to the main computation (Nielsen 2004). This idea grew out of schemes for linear optical quantum computation, in which gates are constructed probabilistically and teleported into the main computation once they succeed (Yoran and Reznik 2003). Adding rows of qubits “just in time” does remove one of the main advantages of the original proposal, i.e., making the cluster state in a few simple global operations. However, it gains in another respect in that the data is regularly being moved into fresh qubits, so the whole system doesn’t have to be kept free of decoherence throughout the computation.



**Quantum Computing, Fig. 7** Fragment of a cluster-state quantum computation. The computation can be thought of as proceeding from *left to right*, with the qubit register arranged vertically, though all qubits in these gates can be measured at the same time. *White* qubits do not contribute directly to the computation and are “removed” by measuring in the *Z* basis. *Pink* qubits are measured in the *Y* basis and form the gates, at the *top* a CNOT and at the *bottom* a Hadamard. *Blue* qubits are measured in the *X* basis and transmit the logical qubit through the cluster state. The *black* qubits contain the output state (not yet measured, since their measurements are determined by the next operations) (details in Raussendorf et al. (2003))

Cluster-state quantum computation has also been important on a theoretical level because it made people think harder about the role of measurement in quantum computing (Jozsa 2005). Instead of striving to obtain perfect unitary quantum evolution throughout the computation, cleverly designed measurements can be used to push the quantum system along the desired computational trajectory.

## Topological Quantum Computing

In the section introducing qubits, we learned about qubits as distinguishable quantum two-state systems. But there are more exotic quantum species; indeed, quantum particles are indistinguishable when more than one of them is around in the same place. This means if you swap two of them, you can’t tell anything happened. And there are precisely two ways to do this for particles living in our familiar three spatial dimensions:

$$|\psi_A\rangle_1|\psi_B\rangle_2 \rightarrow |\psi_B\rangle_1|\psi_A\rangle_2 \quad (5)$$

corresponding to *Bose-Einstein particles* (photons, Helium atoms) or

$$|\psi_A\rangle_1|\psi_B\rangle_2 \rightarrow -|\psi_B\rangle_1|\psi_A\rangle_2 \quad (6)$$

corresponding to *Fermi-Dirac particles* (electrons, protons, neutrons). That extra minus sign can’t be measured directly, but it does have observable effects, so we do know the world really works this way. One important consequence is that two or more Fermi-Dirac particles cannot occupy the same quantum state at the same time, whereas Bose-Einstein particles can and frequently do. From this we get, among many other things, lasers (photons, Bose-Einstein particles) and semiconductors (electrons, Fermi-Dirac particles), without which modern technology would be very different indeed.

A simple change of sign when swapping particles won’t make a quantum computer, but something in between one and minus one can. For this, we have to restrict our particles to two spatial dimensions. Then, it is possible to have particles,

known as *anyons*, that acquire more complicated phase factors when exchanged. Restricting quantum particles to less than three spatial dimensions is not as hard as it sounds. A thin layer of conducting material sandwiched between insulating layers can confine electrons to two-dimensional motion, for example, forming what is known as a 2D electron gas.

The first proposal for a quantum computer using anyons was from Kitaev (2003), using a two-dimensional array of interacting spins. The anyons are formed from excitations of the spins. Think of this as similar to the way electronic displays such as those found in trains and stations often work by flipping pixels between black and yellow to form characters. This opened up the field to a host of connections with group theory, gauge theory, and braid (knot) theory and produced a new quantum algorithm to approximate the Jones polynomial (Aharonov et al. 2006; Aharonov and Arad 2006), a fundamental structure in braid theory. If you already know something about any of the above mathematical subjects, then you can jump into the many accounts of topological quantum computation (also going by the names geometric or holonomic quantum computing). An accessible introduction can be found in Brennen and Pachos (2007) and Pachos (2012).

Here is a simple example of how quantum phases work, to further illuminate the idea. The Aharonov-Bohm effect (Aharonov and Bohm 1959) involves two ingredients. First, electrons can be projected through a double slit much like photons can, and they produce an interference pattern when detected on the far side of the slits. If you aren't familiar with Young's double-slit experiment (Young 1804), note that you need to think about it in terms of single electrons going through both slits in superposition. This is an example of the quantum behavior of particles that combines both wave and particle properties. The interference pattern arises because the electron can't land on the detector in places where the path length difference is half a wavelength and prefers to land near where the path difference is a whole number of wavelengths. The second ingredient is a solenoid, a coil of wire with a current

running through it. It generates a magnetic field inside the coil that does not leak outside. If you put the solenoid between the two slits (see Fig. 8), the interference pattern shifts, even though there is no magnetic field where the electrons are traveling (they can't pass through the solenoid). From this, we deduce two things: interference patterns are a way to detect the extra quantum phases that are generated in topological quantum effects, and, even though there was no interaction between the electrons and the magnetic field, i.e., no energy exchanged, the magnetic field still affected the quantum properties of the electrons just by being inside the two paths the electrons take through the slits.

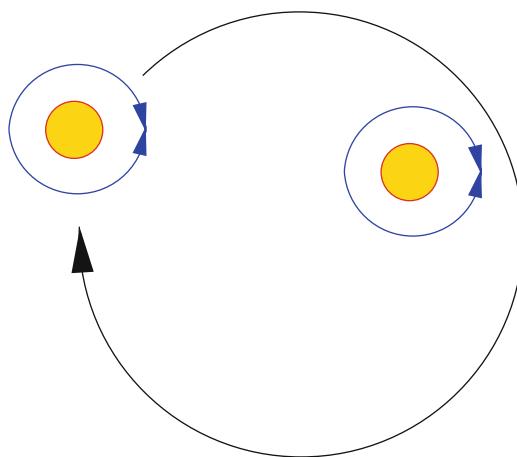
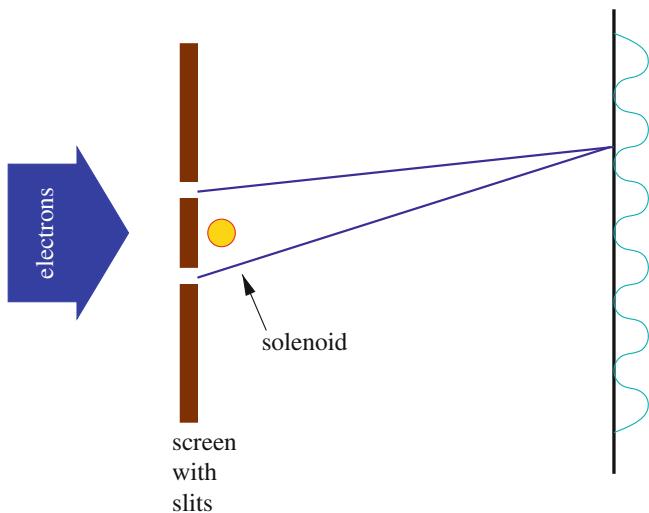
Now, imagine shrinking that experiment down to a magnetic field with a charge (electron) encircling it. This is one way to think of anyons. Confined to two dimensions, the anyons can be moved round each other (Fig. 9), and the charge part of one going round the magnetic field part of the other affects the quantum phase. When you want to measure them, you manipulate them so that they form an interference pattern. The rest of the details of how to make them perform quantum computation comes down to the mathematics of group theory.

Apart from delighting the more mathematically minded quantum computing theorists, this type of quantum computing may have some practical advantages. The effects are generated just by moving the anyons around each other, and the exact path isn't critical, only that they go around the right set of other anyons. So, there is less chance of errors and disturbance spoiling the quantum computation (Preskill 1997). However, constructing a physical system in which anyonic excitations can be manipulated to order is not so easy. Experiments are not as advanced as the theory and are significantly behind the state-of-the-art for standard quantum computing architectures.

## What About Analog?

Both cluster-state and topological quantum computing are more exotic ways to do standard digital

**Quantum Computing,**  
**Fig. 8** Aharonov-Bohm effect



**Quantum Computing, Fig. 9** Anyons: charge (blue) encircling magnetic field (flux, red+yellow), confined to two dimensions

quantum computing; they are different architectures, rather than different models of computation. We can go beyond this and consider different quantum computational models. The obvious place to start is analog. Classical analog computation was once commonplace yet was swept aside by the advance of digital computation. To understand why this happened, we need to review what it means to binary encode our data in a digital computer. Binary encoding has profound implications for the resources required for computation, as Table 1 illustrates.

Binary encoding provides an exponential advantage (reduction) in the amount of memory required to store the data compared to unary encoding. It also means the precision costs are exponentially smaller. To double the precision of a unary representation requires double the resources, while a binary representation achieves this with a single extra bit. This is the main reason why digital computing works so well. It is easy to forget that there was once analog computing in which the data is represented directly as the magnitude of a voltage or water level displacement or, in the slide rule, with logarithms etched onto the stick. Those of you too young to have owned a slide rule should now play with Derek's virtual slide rules, (<http://www.antiquark.com/sliderule/sim/>) to get a feel for what one version of analog computation is like.

It does not, of course, have to be binary encoding; numbers in base three or base ten, for example, gain the exponential advantage just as well. One of the first to discuss the implications of binary encoding in a quantum computing context was Jozsa (1998), and this was refined by Blume-Kohout et al. (2002) and Greentree et al. (2004). Binary encoding matters for quantum computing, too, in the standard digital model. It gains the same exponential advantage from encoding the data into qubits as classical digital computers gain from using bits.

Although the discussion above was framed in terms of discrete systems, the same encoding inefficiencies apply to continuous variables (like length, pressure) that can be used to represent numbers in an analog computer. Since we can only measure up to some fixed precision, any continuous quantity can only produce a discrete set of possible values as outputs of a measurement, which correspond to the  $N$  possible outcomes in Table 1.

## Continuous-Variable Quantum Computing

Despite the binary encoding advantage, there are important uses for analog quantum computing. Although many quantum systems naturally have a fixed number of discrete states (e.g., polarization of a photon or electronic spin), there are also variables like position  $x$  and momentum  $p$  that can take any real value. These can be used to represent data in the same way as voltages or other continuous quantities are used in classical analog computers. This quantum version of analog computation is usually known as continuous-variable or CV quantum computation.

The important difference between quantum and classical continuous variables is that a conjugate pair, such as the position and momentum, are not independent of each other. Instead,  $x$  and  $p$  are related by the commutator,  $[x, p] = xp - px = i\hbar$ .

**Quantum Computing, Table 1** Unary versus binary encoding

Number	Unary	Binary
0	0	
1	•	1
2	..	10
3	...	11
4	....	100
...	...	...
$N$	$N \times •$	$\log_2 N$ bits
<i>Read out:</i>	Distinguish measurements with $N$ outcomes	$\log_2 N$ measurements with 2 outcomes each
<i>Accuracy:</i>	Errors scale linearly in $N$	Errors scale $\propto \log N$

Knowing  $x$  exactly implies  $p$  is unknowable and vice versa; this is called *squeezing*. The most commonly described continuous-variable quantum system uses electromagnetic fields (light, microwaves), but rather than specializing to the case of single photons as one does for digital quantum computing, single *modes* of the field are employed. Braunstein and van Loock (2005) provide a comprehensive review of CV quantum information processing for those interested in further details.

Lloyd and Braunstein (1999) showed that CV quantum computation is universal, i.e., it can compute anything that a digital quantum computer can compute. They did this by showing that it is possible to obtain any polynomial in  $x$  and  $p$  efficiently using operations that are known to be available in physical systems. The universal set of operations has to include a nonlinear operation applied to one of the variables, which means in this context higher than quadratic in  $x$  and  $p$ . Linear operations (beam splitters, mirrors, phase shifters) preserve Gaussian states, roughly, those that are easy to prepare using classical resources. For electromagnetic fields, an example of a suitable nonlinear operation is the Kerr effect, which produces a quartic term in  $x$  and  $p$ . This has the effect of an intensity-dependent refractive index. Unfortunately, although it is observable, in real physical systems, the Kerr effect is too weak to be of practical use in CV quantum computation at optical frequencies. Without the nonlinear term, the operations available are Gaussian preserving, which means they can be simulated efficiently with a classical computer Bartlett et al. (2002). The situation is somewhat reversed at other frequencies, for microwaves a nonlinear operation is easier, but key Gaussian operations, such as beam splitting, are harder.

Some operations are easier in CV than in digital quantum computers. The Fourier transform is a single operation, rather than a long sequence of controlled rotations and Hadamard gates. Hence, for computations requiring many uses of a Fourier transform, there can be advantages for CV quantum computing. Also promising is hybrid digital-analog quantum computing, in which algorithms are structured to use the best method for different tasks. An overview of the experimental efforts is

given by Andersen et al. (2015). Continuous-variable quantum computers are limited in their precision by the degree of squeezing that can be performed on a single mode. An extra bit of precision doubles the required squeezing, so CV quantum computation suffers from the same precision problems as classical analog computing.

Continuous-variable information processing is especially suitable for quantum communications because of the practicality of working with coherent beams of light instead of single photons. Most quantum communications tasks only require Gaussian operations, which are easily achieved with standard optical elements. Quantum repeaters (Ladd et al. 2006) to extend quantum communications channels over arbitrary distances are a good example.

## Hybrid Quantum Computing Architectures

Having noted above that continuous-variable quantum systems are especially suited to quantum communications tasks, one obvious way to use them for quantum computing is to make a high-speed quantum *bus* (qubus) to communicate between qubit registers. Although light and matter interact in simple ways, creating a suitable gate to transmit the quantum information between qubits is a far from trivial task. Coherent light as used for continuous-variable quantum information is also often employed to measure the final state of the matter qubits at the end of the computation.

Unless the required operations are done in a way that reveals only trivial information about the quantum state while the computation is in process, the quantum superpositions will instead be

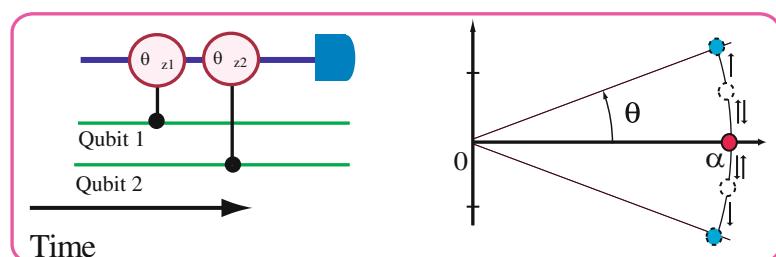
destroyed. One solution is built around an elegant implementation of a parity gate, illustrated in Fig. 10. The left half shows a coherent light beam (blue) interacting with two qubits in turn (green) by means of an interaction that causes a phase shift in the light that depends on the state of the qubit. The right half shows the phase of the light after the interactions. If the 2 qubits are in the same state, the light will have a phase in one of the two blue positions. If the 2 qubits are in different states, the two phase shifts will cancel leaving the phase unaltered (red). Provided the angle of the shift  $\theta$  is large enough, the two cases can be distinguished by measuring only the coordinate of the horizontal axis, not the vertical distance away from it. For a detailed exposition illustrated in a superconducting qubit setting, see Spiller et al. (2006).

Another continuous-variable method that can be used to mediate quantum gates is the vibrational modes of trapped ions. The charges on the ions repel each other, and if one ion is slightly displaced, it will be pushed back and oscillate, like a mass on a spring. The ions also have collective modes, stretching or all swinging together, and these can be used to connect any pair of ions to perform a quantum gate between them (Cirac and Zoller 1995).

## Continuous-Time Quantum Computing

We now turn to a computational model that is distinctly quantum, with no direct classical counterpart. We'd like to use qubits to encode our data, for the efficiency this provides, as explained in the section on [unary versus binary encoding](#). But this doesn't mean we also have to use quantum gates to perform the computation. Recall that qubits can be

**Quantum Computing,**  
**Fig. 10** Qubus parity gate operation



in any superposition of zero and one  $\alpha|0\rangle + \beta|1\rangle$  and that the rotation gate  $R_\theta$  given by Eq. 3 can alter this superposition by any angle or phase. Instead of using discrete quantum gates, we can manipulate our qubits using a continuous-time evolution. This is what quantum systems naturally do; they evolve continuously in time, with the evolution specified by a Hamiltonian  $\hat{H}$  that describes how the quantum system interacts. Schrödinger's equation

$$i\hbar \frac{d|\psi\rangle}{dt} = \hat{H}|\psi\rangle$$

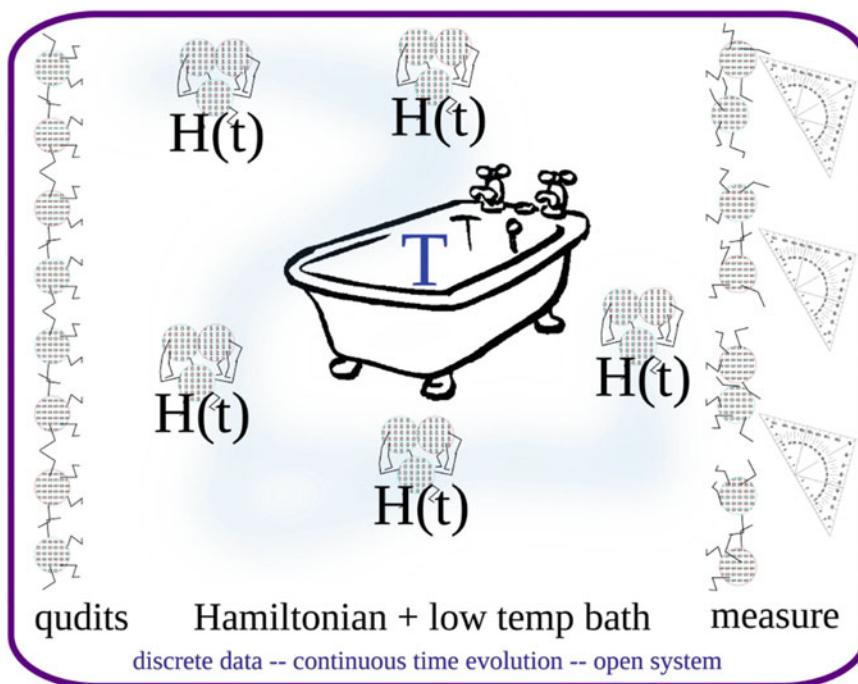
has the formal solution

$$|\psi(t)\rangle = \exp\{-i\hat{H}t/\hbar\}|\psi(0)\rangle \quad (7)$$

for a time-independent  $\hat{H}$ . That exponential of  $\hat{H}$  is a unitary operator, like the ones we already met in the form of quantum gates. Indeed, to implement discrete gates, we actually have to turn on a carefully chosen Hamiltonian and then turn it off

again after the right amount of evolution has taken place for the gate we want. This might be a laser pulse on an atomic qubit, for example. But instead of a sequence of discrete quantum gates, we can design a continuous-time evolution that transforms our qubits from the initial state to the final state that encodes the result of the computation, shown pictorially in Fig. 11. Of course, we are going to have to be clever about this, just like quantum algorithms using gates need some kind of “trick” to ensure the final state tells us the answer with high probability. In the next sections, we'll see several different ways to do this, followed by one of the most important applications of quantum computing, to simulate quantum systems.

But first, a word about why this is a distinctly quantum model of computing. Compare the continuously varying superpositions of qubits with classical bits. Classical bits can take one of just two values, zero or one, and bit flip is the only nontrivial operation. Sure, you may need to



**Quantum Computing, Fig. 11** Continuous-time quantum computing. A register of qubits evolves under a time-varying Hamiltonian  $\hat{H}(t)$  with coupling to a low-temperature

environment (*bath*). After a suitable time, the qubits are measured to obtain the result of the computation

actually turn down the voltage from +5 V to 0 in a continuous manner to change the bit value from one to zero, but there is no logical “in-between” state of the bit as the voltage changes. This has been studied as part of the fundamental difference between quantum and classical theories. Hardy (2001) showed that a continuous evolution between states – as opposed to a discontinuous jump – is enough to distinguish quantum from classical. And it is possible to design simple (nonphysical) models that are partway between the two (Spekkens 2004; Coecke et al. 2010). However, although they reproduce many features of quantum theory – such as entanglement – such model theories don’t provide more powerful computation. Since continuous-time evolution is one of the distinguishing features of quantum mechanics, it makes sense to try to use it to our advantage for computation.

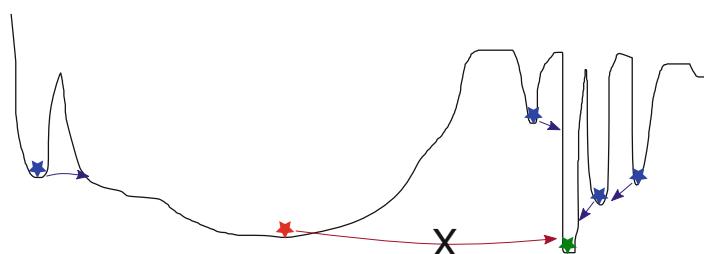
### Adiabatic Quantum Computing and Quantum Annealing

Adiabatic quantum computing and quantum annealing are often conflated in the literature. There are two distinct mechanisms involved that correspond to the two names. They can also be applied in combination, which is part of the origin of the confusion.

**Quantum annealing** is the quantum equivalent of simulated annealing. Simulated annealing is a computational method that finds the solution to a problem that is encoded to be the minimum of some function. It starts in a random initial state

and evolves toward lower values until a minimum is found. In order to avoid being stuck in a local minimum that is higher than the desired solution, some random moves that increase the value are necessary. This can be thought of as starting in a high temperature distribution – in which random jumps are more likely – and slowly “cooling” the simulation until the lowest energy state is found. The slowness of the cooling is to allow time for it to get out of local minima on the way. The same approach can be applied in the quantum case, with the extra help from *quantum tunneling* to escape from local minima. Quantum states don’t get stuck in local minima; they can sneak through barriers to lower-lying states. How fast this happens depends on how broad the barriers are (see Fig. 12). Quantum annealing works best on landscapes with many narrow local minima. For broad barriers, both quantum and simulated annealing will struggle to solve the problem efficiently. There are more efficient hybrid algorithms to deal with such landscapes (see Chancellor 2016a, b).

**Adiabatic quantum computing.** Quantum mechanics provides another effective tool in this setting, thanks to the quantum adiabatic theorem. The evolution starts in a minimum energy state that is easy to prepare and transforms from there into the minimum energy state we are interested in by changing the interaction Hamiltonian for the system. The quantum adiabatic theorem tells us how long it will take to evolve to the desired state with only a very small probability of ending up in some higher energy state by mistake. Suppose we can prepare a system in the ground state  $|\psi_0\rangle$  of



**Quantum Computing, Fig. 12** Schematic of an energy landscape: blue stars are where classical gets stuck, while a quantum process can tunnel; the red star marks where

quantum gets stuck as the barriers are too broad, while the green star is the goal, the lowest energy point in the landscape

the Hamiltonian  $\hat{H}_{\text{simple}}$ , and we want to find the ground state  $|\psi_g\rangle$  of the Hamiltonian  $\hat{H}_{\text{hard}}$ . Then, provided we can evolve the systems under both Hamiltonians together, we apply the following Hamiltonian  $\hat{H}(t)$  to  $|\psi_0\rangle$ :

$$\hat{H}(t) = \{1 - s(t)\}\hat{H}_{\text{simple}} + s(t)\hat{H}_{\text{hard}} \quad (8)$$

where  $s(0) = 0$ ,  $s(T) = 1$ , and  $0 \leq s(t) \leq 1$ . The total time the simulation will take is  $T$ , and  $s(t)$  controls how fast the Hamiltonians switch over. The key parameter is the gap  $\Delta$ , the difference in energy between the ground and first excited states. It varies as the state evolves from  $|\psi_0\rangle$  to  $|\psi_g\rangle$ , and the smaller it is, the slower the state must evolve to stay in the ground state. Let  $\Delta_{\min}$  be the smallest gap that occurs during the adiabatic evolution. Then the quantum adiabatic theorem says that provided  $T \gg \varepsilon/\Delta_{\min}^2$ , the final state will be  $|\phi_0\rangle$  with high probability. Here  $\varepsilon$  is a quantity that relates the rate of change of the Hamiltonian to the ground state and first excited state. It is usually about the same size as the energy of the system. The important parameter is thus  $\Delta_{\min}$ . For hard problems,  $\Delta_{\min}$  will become small, and the required time  $T$  will be large.

Adiabatic quantum computing was introduced by Farhi et al. (2000) as a natural way to solve SAT problems. SAT stands for satisfiability and an example of a Boolean expression to be satisfied is

$$\begin{aligned} B(x_1, x_2, x_3, x_4) = & (x_1 \vee x_3 \vee \tilde{x}_4) \\ & \wedge (\tilde{x}_2 \vee x_3 \vee x_4) \\ & \wedge (\tilde{x}_1 \vee x_2 \vee \tilde{x}_3), \end{aligned} \quad (9)$$

a set of clauses of three variables each combined with “or” that are then combined together with “and.” The tilde over a variable indicates the negation of that variable. The problem is to determine whether there is an assignment of the variables  $x_1, x_2, x_3, x_4$  that makes the expression true (in the above simple example, there are several such assignments, e.g.,  $x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0$ ). The Boolean expression is turned into a Hamiltonian by constructing a cost function for each clause. If the clause is satisfied, the cost is zero; if not, the cost is one unit of energy. The Hamiltonian is the sum of all these cost functions.

If the ground state of the Hamiltonian is zero, the expression can be satisfied, but if it is greater than zero, it means some clauses remain unsatisfied whatever the values of the variables. Real optimization problems can be encoded into qubits in the same way.

The adiabatic quantum computation only works efficiently if it runs in a reasonably short amount of time, which in turn requires the gap  $\Delta_{\min}$  to remain large enough throughout the evolution. Figuring out whether the gap stays large enough is not easy in general and has led to some controversy about the power of quantum computation. The 3SAT problem (clauses with three variables as above) is in the general case a hard problem that requires exponential resources to solve classically. Clearly adiabatic quantum computing can also solve it, but the question is, how long does it take? The best indications are that the gap will become exponentially small for hard problems, and thus the adiabatic quantum computation will require exponential time to solve it, too. For a dissenting view, see Kieu (2006), discussing the traveling salesman problem, another standard “hard” problem. One of the difficulties in analyzing this issue is that adiabatic quantum computation uses the continuous nature of quantum mechanics, and care must be taken when assessing the necessary resources to correctly account for the precision requirements. However, a polynomial speedup for an exponentially hard problem is still useful in practice for real-world problems.

On the other hand, we do know that adiabatic quantum computing is no less powerful than digital computation, i.e., it can solve all the same problems using equivalent resources (Aharonov et al. 2007; Kempe et al. 2004, 2006). It was first implemented in a small proof-of-principle system using nuclear magnetic resonance (NMR) quantum computation (Steffen et al. 2003). There are also indications that it can be resilient to some types of errors (Childs et al. 2002).

Now, how could these two apparently different methods, cooling to the ground state (quantum annealing) and evolving slowly staying in the ground state the whole time (adiabatic quantum computing), be confused? In practice, the same Hamiltonians are used for both, to prepare an

initial state easily using  $\hat{H}_{\text{simple}}$  and to encode the problem,  $\hat{H}_{\text{hard}}$ . So even for quantum annealing it is necessary to switch from  $\hat{H}_{\text{simple}}$  to  $\hat{H}_{\text{hard}}$ , and you may as well do that using something like an adiabatic protocol as in Eq. (8), to give yourself extra chances of finding the ground state easily. The switch between Hamiltonians is too fast to stay in the ground state, but the cooling of the quantum annealing process removes the extra energy, and you find the ground state anyway. A pragmatic “belt-and-braces” approach of using both at the same time is a sensible option for imperfect physical hardware.

## Quantum Walks (Reprise)

We already know that [quantum versions of random walks](#) are a useful quantum computational tool. What I didn’t mention earlier is the continuous-time version of a quantum walk. In fact, the algorithm with an exponential speed up ([Childs et al. 2003](#)) was presented as a continuous-time quantum walk. Only it was to be implemented on a digital quantum computer, with the continuous-time evolution approximated using a quantum form of numerical integration. Can we do better by implementing the quantum walk dynamics directly in continuous-time?

Continuous-time quantum walks are simpler than discrete-time quantum walks, they don’t have a quantum coin, and they just have a hopping rate  $\gamma$  for how likely they are to move to the next position. The walk takes place on a graph with adjacency matrix  $A$ , which has ones for edges and zeros elsewhere. For example, this is the adjacency matrix for a square:

$$A_{\text{square}} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{array}{c|c} 0 & -1 \\ \hline | & | \\ 3 & -2 \end{array}$$

with the corners labeled consecutively corresponding to the columns and rows as shown. This is a symmetric matrix; hence, it can be used to specify a Hamiltonian, which must be a Hermitian matrix, the complex equivalent of

symmetric. The continuous-time quantum walk is just

$$|\psi(t)\rangle = \exp\{-i\gamma At/\hbar\}|\psi(0)\rangle,$$

the solution to Schrödinger’s equation as given in Eq. 7 with  $\hat{H} = \gamma A$ .

This quantum walk Hamiltonian is a good choice for  $\hat{H}_{\text{simple}}$  for adiabatic quantum computing and quantum annealing when  $A$  is the adjacency matrix for a hypercube. Why a hypercube? Because when you encode the quantum walk into qubits – as you must to have an efficient quantum algorithm – a step along one edge of a hypercube corresponds to flipping one of the qubits in the register encoding the label of the position of the walker. And flipping qubits is what you need to do to change the state as you search for the ground state of the problem Hamiltonian  $\hat{H}_{\text{hard}}$ . Can you get an efficient quantum walk algorithm by adding  $\hat{H}_{\text{hard}}$  to  $\hat{H}_{\text{simple}} = \gamma A$ ? For the unordered search problem, yes, you can, as [Childs and Goldstone \(2004\)](#) show. [Childs \(2009\)](#) also proved that quantum walks are universal for quantum computation, so there is every reason to expect that something like this will work for other problems, too, but this is current research ([Callison et al. 2017](#)). What we do know is that, just as hybrid adiabatic and quantum annealing strategies make practical sense, hybrid adiabatic and quantum walk strategies also help when your hardware isn’t perfect ([Morley et al. 2017a, b](#)).

## Quantum Simulation

We have already noted that quantum simulation was one of the original applications that inspired Feynman ([1982](#)) and Deutsch ([1985](#)) to propose the idea of quantum computation. We have also explained why quantum systems cannot generally be simulated efficiently by classical digital computers, because of the need to keep track of the enormous number of superpositions involved, limiting classical simulations to the equivalent of 40 qubits. Ten years later, Lloyd ([1996](#)) proved that a quantum system can simulate another quantum system efficiently, making the suggestions from Feynman and Deutsch concrete.

The first step in quantum simulation is trivially simple; the quantum system to be simulated is mapped onto the quantum simulator by setting up a one-to-one correspondence between their state spaces (which are called Hilbert spaces for quantum systems). The second step is nontrivial, to implement the Hamiltonian interaction in the quantum system. Lloyd proved that the unitary evolution of the quantum system being studied can be efficiently approximated in the quantum simulator by breaking it down into a sequence of small steps that use standard operations. Mathematically this can be written as

$$\exp\{-iHt\} \times \simeq (\exp\{-iH_1 t/n\} \exp\{-iH_2 t/n\} \dots \exp\{-iH_j t/n\})^n + O(t^2/n) [H_j, H_k] \quad (10)$$

where  $\exp\{-iHt\}$  is the unitary evolution of the quantum system and  $H_1 \dots H_j$  are a sequence of standard Hamiltonians available for the quantum simulator. Each  $\exp\{-iH_j t/n\}$  is a unitary quantum gate. The last term says that the errors are small provided  $n$ , the step size, is chosen to be small enough. There are variations and improvements on this that use higher-order corrections to this approximation (e.g., see Berry et al. 2007).

Just as with the quantum algorithms discussed in the first section, evolving the quantum simulation is only half of the job. Extracting the pertinent information from the simulation requires some tricks, depending on what it is you want to find out. A common quantity of interest is the spectral gap, the energy difference between the ground and first excited states, the quantity  $\Delta$  discussed in the section on [adiabatic quantum computing](#). One way to obtain this is to create a superposition of the two energy states and evolve it for a period of time. The phase difference between the ground and excited state will now be proportional to the spectral gap. To measure the phase difference requires a Fourier transform, either quantum or classical. Either way, the simulation has to be repeated or extended to provide enough data points to obtain the spectral gap to the desired precision. Even creating the initial state in superposition of ground and excited states is nontrivial.

It has been shown that this can be done for some systems of interest by using a quasi-adiabatic evolution starting from the ground state. By running the adiabatic evolution a bit too fast, the state of the system doesn't remain perfectly in the ground state, and the required superposition is generated (Abrams and Lloyd 1999).

Simulating quantum systems in this way on a standard digital quantum computer is not the only option. If the Hamiltonian of the system to be simulated can be engineered in another, easier to control, quantum system, then a direct continuous-time quantum simulation can be performed. These are known as “special purpose quantum simulators” because they are designed for specific Hamiltonians, rather than being universal quantum computers, although some quantum Hamiltonians are universal in the sense of being able to simulate all other quantum Hamiltonians (Cubitt et al. 2017). This kind of direct simulation is more efficient, because the approximations in Eq. (10) are avoided, and the natural Hamiltonian of the quantum simulator can run at full speed instead of being turned on and off to create quantum gates. This efficiency comes at the expense of specialization, but a special purpose quantum simulator is also easier to build. Experimentalists are testing machines that are very close to going beyond classical computational power in what they can simulate (e.g., see Neyenhuis et al. (2016) and Bernien et al. (2017)). For a review describing some of the many experimental platforms under development, as well as other applications of quantum simulation, see Brown et al. (2010).

However, because there is no binary encoding, unlike classical simulation on a digital computer, accuracy is a problem for both digital and continuous-time quantum simulation. In particular, accuracy does not scale efficiently with time needed to run the simulation (Brown et al. 2006). Is this going to be a problem in practice? Poor scaling for accuracy didn't stop people building useful analog computers from electrical circuits or water pipes in the days before digital computers were widely available. As we noted, a quantum simulator becomes useful at around 40 qubits in size, which means it may well be the first real

application of quantum computing to calculate something we didn't know already. In the end, the accuracy required depends on the problem we are solving, so we won't know whether it works until we try.

## Non-digital Implementations of Quantum Computing

We have already discussed special purpose quantum simulators that are close to providing useful quantum computation. As well as the systems they are designed to simulate, such hardware can also perform continuous-time quantum computations more generally, where problems are mapped onto the Hamiltonian of the simulator with the solution corresponding to the ground state. The range of useful problems that can be mapped into different simulator Hamiltonians is a topic of current research. Transverse Ising Hamiltonians are known to be universal for classical problems (Bravyi et al. 2006), but other Hamiltonians have not been similarly characterized. This will potentially significantly extend the usefulness of early quantum simulation hardware, such as Fermi gas microscopes (e.g., Kuhr 2016), ion traps (e.g., Neyenhuis et al. 2016), ultracold polar molecules (e.g., Moses et al. 2017), and various superconducting devices. The first commercial device – from D-Wave Systems Inc. – claiming to be a quantum computer uses quantum annealing in superconducting devices. While recent analysis suggests that quantum effects are present, the efficiency of the computations compared to classical computers is still in question (Parekh et al. 2016), due to the high levels of decoherence in the hardware.

Digital types of error correction can't be applied directly to continuous-time evolution (see Bookatz et al. (2015) for alternative approaches). More critically, the precision with which the Hamiltonians must be specified grows with the number of qubits. This means that continuous-time quantum computers are not scalable in the way that digital quantum computers are. Nonetheless, the D-Wave devices suggest that we can make useful hardware at least up to a few thousand qubits, which is far more than the 40 or

so we can simulate classically. Other types of hardware may have better precision, allowing even more qubits. There is thus plenty of room for useful continuous-time quantum computers to provide computational power for us beyond classical limits.

## Biological Quantum Computing?

Beyond the precision quantum engineering required to construct any of the designs described thus far, there is plenty of discussion of quantum computational processes in natural systems. While all natural systems are made up of particles that obey quantum mechanics at a fundamental level, the extent to which their behavior requires quantum logic to describe it is less ubiquitous. Quantum coherences are more usually dissipated into the surrounding environment than marshaled into coordinated activity.

It has been suggested that the brain may exhibit quantum computational capabilities on two different levels. Firstly, Hameroff and Penrose (1996) have argued that brain cells amplify quantum effects that feed into the way the brain functions. This is disputable on physical grounds when the time scales and energies involved are considered in detail. Secondly, a quantum-like logic for brain processes has been proposed (e.g., see Khrennikov (2006)). This does not require actual quantum systems to provide it and is thus not dependent on individual brain cells amplifying quantum effects.

Whether any biological computing is exploiting quantum logic is an open question. Since typical biological temperatures and time scales are generally not a hospitable regime for maintaining quantum coherences, we might expect any such quantum effects to be rare at the level of complex computations. Clearly, quantum effects are biologically important for basic processes such as photosynthesis, for example, and transport properties such as the conductance and coherence of single electrons through biological molecules are the subject of much current study. These are perhaps more likely to feature as problems explored by one of the first quantum simulators, than as examples of natural quantum computers.

Meanwhile, quantum-inspired computation is already making its mark in diverse areas from linguistics to finance to image processing.

## Future Directions

The wide range of creative approaches to quantum computing are largely a consequence of the early stage of development of the field. There is as yet no proven working method to build a useful quantum computer, so new ideas stand a fair chance of becoming a winning combination. We do not even have any concrete evidence that we will be able to build a quantum computer that actually outperforms classical computers, although quantum simulators are now on the cusp of providing this. We also have no evidence to the contrary that quantum computers this powerful are not possible for some fundamental physical reason, so the open question, plus the open questions such a quantum computer may be able to solve, keeps the field vibrant and optimistic.

Classical computation is already undergoing a profound change from an exponential growth in processing power dominated by silicon processors. Moore's law growth has already effectively ended and is being replaced by a diverse range of specialized applications for multiple cores, coprocessors, and embodied and networked computing. This new landscape is well-placed to benefit from unconventional computers of all kinds, especially quantum.

## Bibliography

### Books and Reviews

For those still struggling with the concepts (which probably means most people without a physics degree or other formal study of quantum theory), there are plenty of popular science books and articles. Please dive in: it's the way the world we all live in works, and there is no reason to not dig in deep enough to marvel at the way it fits together and puzzle with the best of us about the bits we can't yet fathom.

For those who want to learn the quantitative details and machinery of quantum computing, this is still the best textbook: Quantum Computation and Quantum Information: (10th Edition). Michael A. Nielsen, Isaac L. Chuang. ISBN 10: 1107002176 ISBN 13: 9781107002173. Publisher: CUP, Cambs., UK

I have cited a number of accessible review articles and books in the primary literature. Especially useful among these are Venegas-Andraca (2012) on quantum versions of random walks; Lidar and Brun (2013), Devitt et al. (2009), and Paler and Devitt (2015) for quantum error correction; Pachos (2012) and Brennen and Pachos (2007) for topological quantum computing; and Brown et al. (2010) for quantum simulation.

For the latest experimental details, the websites of the major academic and commercial players are the best up-to-date source of information. I have highlighted a few already in the main text, notably IBM Q <http://research.ibm.com/ibm-q/> where you can use their demonstrator 5 and 16 qubit transmon quantum computers (current as of July 2017) and D-Wave Inc., <https://www.dwavesys.com/> who build quantum annealers with thousands of superconducting qubits.

Key academic research to watch includes Bristol Centre for Quantum Photonics. <http://www.bristol.ac.uk/physics/research/quantum/> for photonic quantum processors and another online demonstrator; QuTech in Delft <https://qutech.nl/>; Google Santa Barbara John Martinis group <http://web.physics.ucsb.edu/~mart> JILA in Colorado <https://jila.colorado.edu/research/quantum-information> JQI in Maryland <http://jqi.umd.edu/> for ion trap quantum simulators (and much else); and NQIT Oxford <http://nqit.ox.ac.uk/> for modular ion trap quantum computers.

Many of these websites include overviews and tutorials suitable for beginners.

This is a fast-moving area, with major funding in the form of a European Union Quantum Technology Flagship, large national funding programs, and new companies starting up. Exciting developments are promised in the near future.

### Primary Literature

Aaronson S (2012) The complexity zoo. [https://complexityzoo.uwaterloo.ca/Complexity\\_Zoo](https://complexityzoo.uwaterloo.ca/Complexity_Zoo), a comprehensive cross-referenced list of computational complexity classes

Abrams DS, Lloyd S (1999) Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. Phys Rev Lett 83(24):5162. [http://prola.aps.org/abstract/PRL/v83/i24/p5162\\_1](http://prola.aps.org/abstract/PRL/v83/i24/p5162_1)

Aharanov D, Arad I (2006) The bqp-hardness of approximating the jones polynomial. <https://doi.org/10.1088/1367-2630/13/3/035019>, arXiv:quant-ph/0605181

Aharanov D, Ben-Or M (1996) Fault tolerant quantum computation with constant error. In: Proceedings of the 29th ACM STOC, ACM, NY, pp 176–188, arXiv: quantph/9611025

Aharanov D, Ambainis A, Kempe J, Vazirani U (2001) Quantum walks on graphs. In: Proceedings of the 33rd annual ACM STOC, ACM, NY, pp 50–59, quant-ph/0012090

Aharanov D, Jones V, Landau Z (2006) A polynomial quantum algorithm for approximating the Jones polynomial.

- In: STOC'06: Proceedings of the 38th annual ACM symposium on theory of computing, ACM, New York, pp 427–436. <https://doi.org/10.1145/1132516.1132579>
- Aharonov D, van Dam W, Kempe J, Landau Z, Lloyd S, Regev O (2007) Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM J Comput* 37:166, <arXiv:quant-ph/0405098>
- Aharonov Y, Bohm D (1959) Significance of electromagnetic potentials in quantum theory. *Phys Rev* 115:485–491
- Ambainis A (2003) Quantum walks and their algorithmic applications. *Intl J Quantum Inf* 1(4):507–518, <ArXiv:quant-ph/0403120>
- Ambainis A (2004) Quantum walk algorithms for element distinctness. In: 45th annual IEEE symposium on foundations of computer science, Oct 17–19, 2004, IEEE computer society press, Los Alamitos, CA, pp 22–31, <quant-ph/0311001>
- Andersen UL, Neergaard-Nielsen JS, van Loock P, Furusawa A (2014) Hybrid quantum information processing. *Nature Physics* 11, 713–719 (2015). <https://doi.org/10.1038/nphys3410>, <arXiv:1409.3719>
- Barenco A, Bennett CH, Cleve R, DiVincenzo DP, Margolus N, Shor P, Sleator T, Smolin JA, Weinfurter H (1995) Elementary gates for quantum computation. *Phys Rev A* 52(5):3457–3467. <https://doi.org/10.1103/PhysRevA.52.3457>
- Bartlett S, Sanders B, Braunstein SL, Moto KN (2002) Efficient classical simulation of continuous variable quantum information processes. *Phys Rev Lett* 88:097904, <arXiv:quant-ph/0109047>
- Bartlett SD, Rudolph T, Spekkens RW (2006) Reference frames, superselection rules, and quantum information. *Rev Mod Phys* 79:555, <arXiv:quant-ph/0610030>
- Bennett CH, Brassard G (1984) Quantum cryptography: public-key distribution and coin tossing. In: IEEE international conference on computers, systems and signal processing, IEEE Computer Society Press, Los Alamitos, CA, pp 175–179
- Bennett CH, Wiesner SJ (1992) Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Phys Rev Lett* 69(20):2881–2884
- Bennett CH, Brassard G, Crépeau C, Jozsa R, Peres A, Wootters WK (1993) Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys Rev Lett* 70:1895–1899
- Bennett CH, Bernstein E, Brassard G, Vazirani U (1997) Strengths and weaknesses of quantum computing. *SIAM J Comput* 26(5):151–152
- Bernien H, Schwartz S, Keesling A, Levine H, Omran A, Pichler H, Choi S, Zibrov AS, Endres M, Greiner M, Vuletić V, Lukin MD (2017) Probing many-body dynamics on a 51-atom quantum simulator. <arXiv:1707.04344>
- Berry DW, Ahokas G, Cleve R, Sanders BC (2007) Efficient quantum algorithms for simulating sparse hamiltonians. *Commun Math Phys* 270:359–371. <https://doi.org/10.1007/s00220-006-0150-x>, <http://springerlink.com/content/hk748444j37r228/>
- Blume-Kohout R, Caves CM, Deutsch IH (2002) Climbing mount scalable: physical resource requirements for a scalable quantum computer. *Found Phys* 32(11):1641–1670, <ArXiv:quant-ph/0204157>
- Bookatz AD, Farhi E, Zhou L (2014) Error suppression in hamiltonian based quantum computation using energy penalties. *Phys. Rev. A* 92, 022317. <https://doi.org/10.1103/PhysRevA.92.022317>, <arXiv:1407.1485>
- Braunstein SL, van Loock P (2005) Quantum information with continuous variables. *Rev Mod Phys* 77:513–578, <ArXiv:quant-ph/0410100v1>
- Bravyi S, DiVincenzo DP, Oliveira RI, Terhal BM (2006) The complexity of stoquastic local hamiltonian problems <arXiv:quant-ph/0606140>
- Brennen GK, Pachos JK (2007) Why should anyone care about computing with anyons? *Proc Roy Soc Lond A* 464(2089):1–24, <ArXiv:0704.2241v2>
- Brown KR, Clark RJ, Chuang IL (2006) Limitations of quantum simulation examined by a pairing Hamiltonian using nuclear magnetic resonance. *Phys Rev Lett* 97(5):050504, <http://link.aps.org/abstract/PRL/v97/e050504>
- Brown KL, Munro WJ, Kendon VM (2010) Using quantum computers for quantum simulation. *Entropy* 12(11):2268–2307. <https://doi.org/10.3390/e12112268>
- Brown KL, Horsman C, Kendon VM, Munro WJ (2012) Layer by layer generation of cluster states. *Phys Rev A* 85:052305, <http://arxiv.org/abs/1111.1774v1>
- Callison A, Chancellor NC, Kendon VM (2017) Continuous-time quantum walk algorithm for random spin-glass problems. In preparation
- Chancellor N (2016a) Modernizing quantum annealing ii: Genetic algorithms and inference. <arXiv:1609.05875>
- Chancellor N (2016b) Modernizing quantum annealing using local searches. <https://doi.org/10.1088/1367-2630/aa59c4>, <arXiv:1606.06833>
- Childs A, Eisenberg JM (2005) Quantum algorithms for subset finding. *Quantum Inf Comput* 5:593–604, <ArXiv:quant-ph/0311038>
- Childs A, Goldstone J (2004) Spatial search by quantum walk. *Phys Rev A* 70:022314, <quant-ph/0306054>
- Childs AM (2009) Universal computation by quantum walk. *Phys Rev Lett* 102:180501
- Childs AM, Farhi E, Preskill J (2002) Robustness of adiabatic quantum computation. *Phys Rev A* 65:012322, <ArXiv:quant-ph/0108048>
- Childs AM, Cleve R, Deotto E, Farhi E, Gutmann S, Spielman DA (2003) Exponential algorithmic speedup by a quantum walk. In: Proceedings of the 35th annual ACM STOC, ACM, NY, pp 59–68. <arXiv:quant-ph/0209131>
- Chun H, Choi I, Faulkner G, Clarke L, Barber B, George G, Capon C, Niskanen A, Wabnig J, OBrien D, Bitauld D (2016) Motion-compensated handheld quantum key distribution system. <arXiv:1608.07465>
- Cirac JI, Zoller P (1995) Quantum computations with cold trapped ions. *Phys Rev Lett* 74(20):4091. <https://doi.org/10.1103/PhysRevLett.74.4091>, <http://link.aps.org/abstract/PRL/v74/p4091>

- Coecke B, Edwards B, Spekkens RW (2010) Phase groups and the origin of non-locality for qubits. <https://doi.org/10.1016/j.entcs.2011.01.021>, arXiv:1003.5005
- Collins RJ, Amiri R, Fujiwara M, Honjo T, Shimizu K, Tamaki K, Takeoka M, Andersson E, Buller GS, Sasaki M (2016) Experimental transmission of quantum digital signatures over 90-km of installed optical fiber using a differential phase shift quantum key distribution system. Opt Lett 41:4883. <https://doi.org/10.1364/OL.41.004883>, arXiv:1608.04220
- Courtland R (2016) Chinas 2,000-km quantum link is almost complete. IEEE Spectr. <http://spectrum.ieee.org/telecom/security/chinas-2000km-quantum-link-is-almost-complete>. iD Quantique, MagicQ
- Cubitt T, Montanaro A, Piddock S (2017) Universal quantum Hamiltonians. arXiv:1701.05182
- De Raedt K, Michielsen K, De Raedt H, Trieu B, Arnold G, Richter M, Lippert T, Watanabe H, Ito N (2007) Massive parallel quantum computer simulator. Comput Phys Commun 176:127–136, arXiv:quant-ph/0608239v1
- Deutsch D (1985) Quantum-theory, the church-Turing principle and the universal quantum computer. Proc R Soc Lond A 400(1818):97–117
- Deutsch D, Jozsa R (1992) Rapid solutions of problems by quantum computation. Proc Roy Soc Lon A 439:553
- Devitt SJ, Nemoto K, Munro WJ (2009) Quantum error correction for beginners. <https://doi.org/10.1088/0034-4885/76/7/076001>, arXiv:0905.2794
- Farhi E, Goldstone J, Gutmann S, Sipser M (2000) Quantum computation by adiabatic evolution. ArXiv:quant-ph/0001106
- Feynman RP (1982) Simulating physics with computers. Int J Theor Phys 21:467
- Flitney AP, Abott D (2002) An introduction to quantum game theory. Fluctuation Noise Lett 02(04):R175–R187. <https://doi.org/10.1142/S0219477502000981>
- Gottesman D, Chuang IL (2001) Quantum digital signatures. <http://arxiv.org/abs/quant-ph/0105032v2>
- Greentree AD, Schirmer SG, Green F, Hollenberg LCL, Hamilton AR, Clark RG (2004) Maximizing the hilbert space for a finite number of distinguishable quantum states. Phys Rev Lett 92:097901, ArXiv:quant-ph/0304050
- Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th annual ACM STOC, ACM, NY, p 212, ArXiv:quant-ph/9605043
- Grover LK (1997) Quantum mechanics helps in searching for a needle in a haystack. Phys Rev Lett 79:325, ArXiv:quant-ph/9706033
- Hameroff S, Penrose R (1996) Conscious events as orchestrated spacetime selections. J Conscious Stud 3(1): 36–53
- Hardy L (2001) Quantum theory from five reasonable axioms. arXiv:quant-ph/0101012
- Horsman C, Brown KL, Munro WJ, Kendon VM (2011) Reduce, reuse, recycle for robust cluster-state generation. Phys Rev A 83(4):042327. ArXiv:1005.1621[quant-ph]
- Horsman C, Stepney S, Wagner RC, Kendon V (2014) When does a physical system compute? Proc Roy Soc A 470(2169):20140182, arXiv:1309.7979
- Jozsa R (1998) Entanglement and quantum computation. In: Huggett SA, Mason LJ, Tod KP, Tsou S, Woodhouse NMJ (eds) The geometric universe, geometry, and the work of Roger Penrose. Oxford University Press, Oxford, pp 369–379
- Jozsa R (2005) An introduction to measurement based quantum computation. ArXiv:quant-ph/0508124
- Kempe, Kitaev, Regev (2004) The complexity of the local hamiltonian problem. In: Proceedings of the 24th FSTTCS, pp 372–383. ArXiv:quant-ph/0406180
- Kempe, Kitaev, Regev (2006) The complexity of the local hamiltonian problem. SIAM J Comput 35(5): 1070–1097
- Kendon V, Tregenna B (2003) Decoherence can be useful in quantum walks. Phys Rev A 67:042315, ArXiv: quant-ph/0209005
- Khrennikov A (2006) Brain as quantum-like computer. Biosystems 84:225–241, ArXiv:quant-ph/0205092v8
- Kieu TD (2006) Quantum adiabatic computation and the travelling salesman problem, ArXiv:quant-ph/ 0601151v2
- Kitaev AY (2003) Fault-tolerant quantum computation by anyons. Ann Phys 303:2–30, ArXiv:quant-ph/ 9707021v1
- Kleinjung T, Aoki K, Franke J, Lenstra A, Thom E, Bos J, Gaudry P, Kruppa A, Montgomery P, Osvik DA, te Riele H, Timofeev A, Zimmermann P (2010) Factorization of a 768-bit rsa modulus. Cryptology ePrint Archive, Report 2010/006, <http://eprint.iacr.org/2010/006>
- Knill E, Laflamme R, Zurek W (1996) Threshold accuracy for quantum computation. ArXiv:quant-ph/9610011
- Kuhr S (2016) Quantum-gas microscopes – a new tool for cold-atom quantum simulators. Natl Sci Rev. <https://doi.org/10.1093/nsr/nww023>, arXiv:1606.06990
- Ladd TD, van Loock P, Nemoto K, Munro WJ, Yamamoto Y (2006) Hybrid quantum repeater based on dispersive cqed interactions between matter qubits and bright coherent light. New J Phys 8:164. <https://doi.org/10.1088/1367-2630/8/9/184>, ArXiv:quant-ph/ 0610154v1
- Lidar DA, Brun TA (eds) (2013) Quantum error correction. Cambridge University Press, Cambridge, UK
- Lloyd S (1996) Universal quantum simulators. Science 273(5278):1073–1078
- Lloyd S (2000) Ultimate physical limits to computation. Nature 406:1047–1054, ArXiv:quant-ph/9908043
- Lloyd S, Braunstein SL (1999) Quantum computation over continuou variables. Phys Rev Lett 82:1784, ArXiv: quant-ph/9810082v1
- Lomont C (2004) The hidden subgroup problem – review and open problems. arXiv:quant-ph/0411037
- Magniez F, Santha M, Szegedy M (2003) An o(n<sup>1.3</sup>) quantum algorithm for the triangle problem. ArXiv: quant-ph/0310134
- Magniez F, Santha M, Szegedy M (2005) Quantum algorithms for the triangle problem. In: Proceedings of 16th

- ACM-SIAM symposium on discrete algorithms, society for industrial and applied mathematics, Philadelphia, pp 1109–1117
- Margolus N, Levitin LB (1996) The maximum speed of dynamical evolution. In: Toffoli T, Biafore M, Liao J (eds) *Physcomp96*. NECSI, Boston
- Margolus N, Levitin LB (1998) The maximum speed of dynamical evolution. *Physica D* 120:188–195, [ArXiv: quant-ph/9710043v2](https://arxiv.org/abs/quant-ph/9710043v2)
- Metodi TS, Thaker DD, Cross AW, deric T Chong F, Chuang IL (2005) A quantum logic array micro-architecture: scalable quantum data movement and computation. In: 38th annual IEEE/ACM international symposium on microarchitecture (MICRO'05), IEEE Computer Society Press, Los Alamitos, CA, pp 305–318. [ArXiv:quant-ph/0509051v1](https://arxiv.org/abs/quant-ph/0509051v1)
- Montanaro A (2015) Quantum algorithms: an overview. <https://doi.org/10.1038/npjqi.2015.23>, [arXiv:1511.04206](https://arxiv.org/abs/1511.04206)
- Morley JG, Chancellor NC, Kendon VM, Bose S (2017a) Quantum search with hybrid adiabatic quantum-walk algorithms and realistic noise. <https://arxiv.org/abs/1709.00371>
- Morley JG, Chancellor NC, Kendon VM, Bose S (2017b) Quench vs adiabaticity: which is best for quantum search on realistic machines? In preparation
- Moses A, Covey P, Miecnikowski T, Jin DS, Ye J (2017) New frontiers for quantum gases of polar molecules. *Nat Phys* 13:13–20. <http://www.nature.com/doifinder/10.1038/nphys3985>
- Neyenhuis B, Smith J, Lee AC, Zhang J, Richerme P, Hess PW, Gong ZX, Gorshkov AV, Monroe C (2016) Observation of prethermalization in long-range interacting spin chains. [arXiv:1608.00681](https://arxiv.org/abs/1608.00681)
- Nickerson NH, Fitzsimons JF, Benjamin SC (2014) Freely scalable quantum technologies using cells of 5-to-50 qubits with very lossy and noisy photonic links. *Phys Rev X* 4. <https://doi.org/10.1103/PhysRevX.4.041041>, [arXiv:1406.0880](https://arxiv.org/abs/1406.0880)
- Nielsen M, Chuang I (1996) Talk at KITP workshop: quantum coherence and decoherence D. P. DiVincenzo, W. Zurek. <http://www.kitp.ucsb.edu/activities/conferences/past/>
- Nielsen MA (2004) Optical quantum computation using cluster states. *Phys Rev Lett* 93:040503
- Pachos JK (2012) Introduction to topological quantum computation. Cambridge University Press, Cambs., UK. ISBN 9781107005044 1107005043
- Paler A, Devitt SJ (2015) An introduction to fault-tolerant quantum computing. In: DAC'15 Proceedings of the 52nd annual design automation conference, p 60. [arXiv:1508.03695](https://arxiv.org/abs/1508.03695)
- Parekh O, Wendt J, Shulenburger L, Landahl A, Moussa J, Aidun J (2016) Benchmarking adiabatic quantum optimization for complex network analysis. Report number SAND2015-3025. [arXiv:1604.00319](https://arxiv.org/abs/1604.00319)
- Preskill J (1997) Fault-tolerant quantum computation. Check and update reference. [arXiv:quant-ph/9712048](https://arxiv.org/abs/quant-ph/9712048)
- Raussendorf R, Briegel HJ (2001) A one-way quantum computer. *Phys Rev Lett* 86(22):5188–5191. <https://doi.org/10.1103/PhysRevLett.86.5188>
- Raussendorf R, Browne DE, Briegel HJ (2003) Measurement-based quantum computation on cluster states. *Phys Rev A* 68(2):022312. <https://doi.org/10.1103/PhysRevA.68.022312>
- Richter P (2007a) Almost uniform sampling in quantum walks. *New J Phys* 9:72, [ArXiv:quant-ph/0606202](https://arxiv.org/abs/quant-ph/0606202)
- Richter P (2007b) Quantum speedup of classical mixing processes. *Phys Rev A* 76:042306, [ArXiv:quant-ph/0609204](https://arxiv.org/abs/quant-ph/0609204)
- Shenvi N, Kempe J, Birgitta Whaley K (2003) A quantum random walk search algorithm. *Phys Rev A* 67:052307, [ArXiv:quant-ph/0210064](https://arxiv.org/abs/quant-ph/0210064)
- Shor P (1994) Algorithms for quantum computation: discrete logarithms and factoring. In: foundations of computer science, 1994 proceedings., 35th annual symposium on, IEEE Computer Society Press, Los Alamitos, pp 124–134. <https://doi.org/10.1109/SFCS.1994.365700>
- Shor PW (1995) Scheme for reducing decoherence in quantum computer memory. *Phys Rev A* 52:R2493
- Shor PW (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J Sci Statist Comput* 26:1484, [quant-ph/9508027](https://arxiv.org/abs/quant-ph/9508027)
- Spekkens RW (2004) In defense of the epistemic view of quantum states: a toy theory. <https://doi.org/10.1103/PhysRevA.75.032110>, [arXiv:quant-ph/0401052](https://arxiv.org/abs/quant-ph/0401052)
- Spiller TP, Nemoto K, Braunstein SL, Munro WJ, van Loock P, Milburn GJ (2006) Quantum computation by communication. *New J Phys* 8:30, [ArXiv:quant-ph/0509202v3](https://arxiv.org/abs/quant-ph/0509202v3)
- Steane A (1996) Multiple particle interference and quantum error correction. *Proc Roy Soc Lond A* 452:2551, [ArXiv:quant-ph/9601029](https://arxiv.org/abs/quant-ph/9601029)
- Steffen M, van Dam W, Hogg T, Breyta G, Chuang I (2003) Experimental implementation of an adiabatic quantum optimization algorithm. *Phys Rev Lett* 90(6):067903, [ArXiv:quant-ph/0302057](https://arxiv.org/abs/quant-ph/0302057)
- Venegas-Andraca SE (2012) Quantum walks: a comprehensive review. *Quantum Inf Process* 11(5):1015–1106. <https://doi.org/10.1007/s11128-012-0432-5>, [arXiv:1201.4780](https://arxiv.org/abs/1201.4780)
- Verstraete F, Porras D, Cirac JI (2004) Density matrix renormalization group and periodic boundary conditions: a quantum information perspective. *Phys Rev Lett* 93:227205, <https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.93.227205>
- Wang L, Piorn I, Verstraete F (2011) Monte carlo simulation with tensor network states. *Phys Rev B* 83:134421
- Yoran N, Reznik B (2003) Deterministic linear optics quantum computation with single photon qubits. *Phys Rev Lett* 91:037903
- Young T (1804) Experimental demonstration of the general law of the interference of light. *Phil Trans Royal Soc Lon* 94



---

## Computing with Solitons

Darren Rand<sup>1</sup> and Ken Steiglitz<sup>2</sup>

<sup>1</sup>Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, USA

<sup>2</sup>Computer Science Department, Princeton University, Princeton, NJ, USA

### Article Outline

Glossary  
Definition of the Subject  
Introduction  
Scalar Solitons  
Vector Solitons  
Manakov Solitons  
Manakov Soliton Computing  
FANOUT  
NOT and ONE Gates  
Output/Input Converters, Two-Input Gates, and NAND  
Time Gating  
Wiring  
A Second Speed and Final FANOUT and NAND  
Universality  
Discussion  
Multistable Soliton Collision Cycles  
The Basic Three-Cycle and Computational Experiments  
A Tristable Example Using a Four-Cycle  
Discussion  
Application to Noise-Immune Soliton Computing  
Experiments  
Experimental Setup and Design  
Vector Soliton Propagation  
Spatial Soliton Collisions  
Future Directions  
Bibliography

## Glossary

**Integrable** This term is generally used in more than one way and in different contexts. For the purposes of this article, a partial differential equation or system of partial differential equations is integrable if it can be solved explicitly to yield solitons (qv).

**Manakov system** A system of two cubic Schrödinger equations where the self- and cross-phase modulation terms have equal weight.

**Nonlinear Schrodinger equation** A partial differential equation that has the same form as the Schrodinger equation of quantum mechanics, with a term nonlinear in the dependent variable, and, for the purposes of this article, is interpreted classically.

**Self- and cross-phase modulation** Any terms in a nonlinear Schrödinger equation that involve nonlinear functions of the dependent variable of the equation or nonlinear functions of a dependent variable of another (coupled) equation, respectively.

**Solitary wave** A solitary wave is a wave characterized by undistorted propagation. Solitary waves do not in general maintain their shape under perturbations or collisions.

**Soliton** A soliton is a solitary wave which is also robust under perturbations and collisions.

**Turing equivalent** Capable of simulating any Turing machine, and hence by Turing's thesis capable of performing any computation that can be carried out by a sequence of effective instructions on a finite amount of data. A machine that is Turing equivalent is therefore as powerful as any digital computer. Sometimes a device that is Turing equivalent is called "universal."

### Definition of the Subject

Solitons are localized, shape-preserving waves characterized by robust collisions. First observed as water waves by John Scott Russell (1844) in

the Union Canal near Edinburgh and subsequently recreated in the laboratory, solitons arise in a variety of physical systems, as both temporal pulses which counteract dispersion and spatial beams which counteract diffraction.

Solitons with two components, vector solitons, are computationally universal due to their remarkable collision properties. In this article, we describe in detail the characteristics of Manakov solitons, a specific type of vector soliton, and their applications in computing.

## Introduction

In this section, we review the basic principles of soliton theory and spotlight relevant experimental results. Interestingly, the phenomena of soliton propagation and collision occur in many physical systems despite the diversity of mechanisms that bring about their existence. For this reason, the discussion in this article will treat temporal and spatial solitons interchangeably, unless otherwise noted.

## Scalar Solitons

A pulse in optical fiber undergoes dispersion, or temporal spreading, during propagation. This effect arises because the refractive index of the silica glass is not constant, but is rather a function of frequency. The pulse can be decomposed into a frequency range – the shorter the pulse, the broader its spectral width. The frequency dependence of the refractive index will cause the different frequencies of the pulse to propagate at different velocities, giving rise to dispersion. As a result, the pulse develops a chirp, meaning that the individual frequency components are not evenly distributed throughout the pulse. There are two types of dispersion: normal and anomalous. If the longer wavelengths travel faster, the medium is said to have *normal* dispersion. If the opposite is true, the medium has *anomalous* dispersion.

The response of a dielectric such as optical fiber is nonlinear. Most of the nonlinear effects in fiber originate from nonlinear refraction, where the

refractive index  $n$  depends on the intensity of the propagating field according to the relation

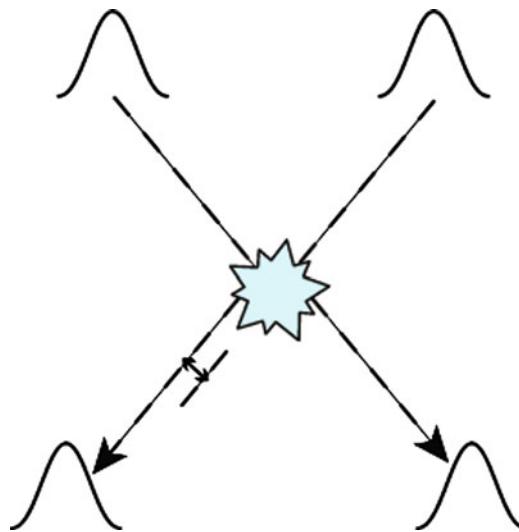
$$n = n_0 + n_2 |E|^2, \quad (1)$$

where  $n_0$  is the linear part of the refractive index,  $|E|^2$  is the optical intensity, and  $n_2$  is the coefficient of nonlinear contribution to the refractive index. Because the material responds almost instantaneously, on the order of femtoseconds, and because the phase shift  $\Delta\phi$  is proportional to  $n$ , each component of an intense optical pulse sees a phase shift proportional to its intensity. Since the frequency shift  $\delta\omega = -(\delta\phi)/(\partial t)$  the leading edge of the pulse is redshifted ( $\delta\omega < 0$ ), while the trailing edge is blueshifted ( $\delta\omega > 0$ ), an effect known as self-phase modulation (SPM). As a result, if the medium exhibits normal dispersion, the pulse is broadened; for anomalous dispersion, the pulse is compressed. Under the proper conditions, this pulse compression can exactly cancel the linear, dispersion-induced broadening, resulting in distortionless soliton propagation. For more details, see the book by Agrawal (2001).

The idealized mathematical model for this pulse propagation is the nonlinear Schrödinger equation (NLSE):

$$i \frac{\partial u}{\partial z} \pm \frac{1}{2} \frac{\partial^2 u}{\partial x^2} + |u|^2 u = 0, \quad (2)$$

where  $u(z, x)$  is the complex-valued field envelope,  $z$  is a normalized propagation distance, and  $x$  is normalized time propagating with the group velocity of the pulse. The second and third terms describe dispersion and the intensity-dependent Kerr nonlinearity, respectively. The coefficient of the dispersion term is positive for anomalous dispersion and negative for normal dispersion. Equation 2, known as the *scalar* NLSE, is integrable, that is, it can be solved analytically, and collisions between solitons are “elastic,” in that no change in amplitude or velocity occurs as a result of a collision. Zakharov and Shabat (1971) first solved this equation analytically using the inverse scattering method. It describes, for example, the propagation of picosecond or longer pulses propagating in lossless optical fiber.



**Computing with Solitons, Fig. 1** Schematic of a scalar soliton collision, in which amplitude and velocities are unchanged. The two soliton collision effects are a position shift (depicted through the translational shift in the soliton path) and phase shift (not pictured)

Two solitons at different wavelengths will collide in an optical fiber due to dispersion-induced velocity differences. A schematic of such a collision is depicted in Fig. 1. The scalar soliton collision is characterized by two phenomena – a position and phase shift – both of which can be understood in the same intuitive way. During collision, there will be a local increase in intensity, causing a local increase in the fiber's refractive index, according to Eq. 1. As a result, both the soliton velocity and phase will be affected during the collision.

From an all-optical signal processing perspective, the phase and position shifts in a soliton collision are not useful. This is because these effects are independent of any soliton properties that are changed by collision, that is, the result of one collision will not affect the result of subsequent collisions. Scalar solitons are therefore not useful for complex logic or computing, which depend on multiple, cascaded interactions.

Despite this setback, it was discovered later that a system similar to the scalar NLSE, the Manakov system (Manakov 1973), possesses very rich collisional properties (Radhakrishnan et al. 1997) and is integrable as well. Manakov solitons are a specific instance of two-component

vector solitons, and it has been shown that collisions of Manakov solitons are capable of transferring information via changes in a complex-valued polarization state (Jakubowski et al. 1998).

## Vector Solitons

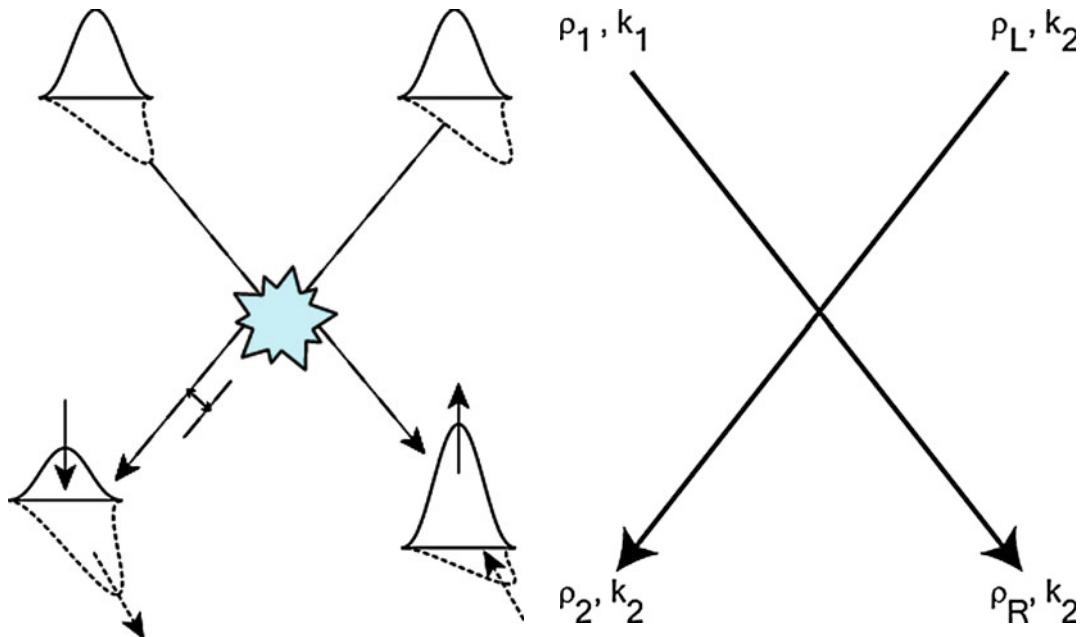
When several field components, distinguished by polarization and/or frequency, propagate in a nonlinear medium, the nonlinear interaction between them must be considered as well. This interaction between field components results in intensity-dependent nonlinear coupling terms analogous to the self-phase modulation term in the scalar case. Such a situation gives rise to a set of coupled nonlinear Schrodinger equations and may allow for propagation of *vector* solitons. For the case of two components propagating in an ideal sl medium with no higher-order effects and only intensity-dependent nonlinear coupling, the equations become

$$i \frac{\partial u_1}{\partial z} + \frac{\partial^2 u_1}{\partial x^2} + 2\mu(|u_1|^2 + \alpha|u_2|^2)u_1 = 0 \quad (3)$$

$$i \frac{\partial u_2}{\partial z} + \frac{\partial^2 u_2}{\partial x^2} + 2\mu(|u_2|^2 + \alpha|u_1|^2)u_2 = 0$$

where  $u_1(z, x)$  and  $u_2(z, x)$  are the complex-valued pulse envelopes for each component,  $\mu$  is a nonlinearity parameter, and  $\alpha$  describes the ratio between self- and cross-phase modulation contributions to the overall nonlinearity. Only for the special case of  $\alpha = 1$  are Eq. 3 integrable. First solved using the method of inverse scattering by Manakov (1973), Eq. 3 admit solutions known as Manakov solitons. For nonintegrable cases ( $\alpha \neq 1$ ), some analytical solitary wave solutions are known for specific cases, although in general a numerical approach is required (Yang 1997). The specific case of  $\alpha = 2/3$ , for example, corresponds to linearly birefringent polarization-maintaining fiber and will be considered in more detail in section “[Experiments](#).”

Due to their multicomponent structure, vector solitons have far richer collision dynamics than their scalar, one-component counterparts. Recall that scalar collisions are characterized by phase and position shifts only. Vector soliton collisions



**Computing with Solitons, Fig. 2** Schematic of a vector soliton collision, which exhibits a position shift and phase shift (not pictured), similar to the scalar soliton collision (cf. Fig. 1). Vector soliton collisions also display an energy redistribution among the component fields, shown here as two orthogonal polarizations. Arrows indicate direction of energy redistribution

also exhibit these effects, with the added feature of possible intensity redistributions between the component fields (Manakov 1973; Radhakrishnan et al. 1997). This process is shown schematically in Fig. 2. In the collision, two conservation relations are satisfied: (i) the energy in each soliton is conserved and (ii) the energy in each component is conserved. It can be seen that when the amplitude of one component in a soliton increases as a result of the collision, the other component decreases, with the opposite exchange in the second soliton. The experimental observation of this effect will be discussed in section “[Experiments](#).” In addition to fundamental interest in such solitons, collisions of vector solitons make possible unique applications, including collision-based logic and universal computation (Jakubowski et al. 1998; Rand et al. 2005; Steiglitz 2000, 2001), as discussed in section “[Manakov Soliton Computing](#).”

**Computing with Solitons, Fig. 3** Schematic of a general two-soliton collision. Each soliton is characterized by a complex-valued polarization state  $p$  and complex parameter  $k$  (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

## Manakov Solitons

As mentioned in section “[Introduction](#),” computation is possible using vector solitons because of an energy redistribution that occurs in a collision. In this section, we provide the mathematic background of Manakov soliton theory, in order to understand soliton computing and a remarkable way to achieve bistability using soliton collisions as described in section “[Manakov Soliton Computing](#)” and “[Multistable Soliton Collision Cycles](#),” respectively.

The Manakov system consists of two coupled NLSEs (Manakov 1973):

$$\begin{aligned} i \frac{\partial q_1}{\partial z} + \frac{\partial^2 q_1}{\partial x^2} + 2\mu(|q_1|^2 + |q_2|^2)q_1 &= 0, \\ i \frac{\partial q_2}{\partial z} + \frac{\partial^2 q_2}{\partial x^2} + 2\mu(|q_1|^2 + |q_2|^2)q_2 &= 0, \end{aligned} \quad (4)$$

where  $q_1(x, z)$  and  $q_2(x, z)$  are two interacting optical components,  $\mu$  is a positive parameter

representing the strength of the nonlinearity, and  $x$  and  $z$  are normalized space and propagation distance, respectively. As mentioned in section “[Vector Solitons](#),” the Manakov system is a special case of Eq. 3 with  $\alpha = 1$ . The two components can be thought of as components in two polarizations or, as in the case of a photorefractive crystal, two uncorrelated beams (Christodoulides et al. 1996).

Manakov first solved Eq. 4 by the method of inverse scattering (Manakov 1973). The system admits single-soliton, two-component solutions that can be characterized by a complex number  $k \equiv k_R + ik_I$  where  $k_R$  represents the energy of the soliton and  $k_I$  the velocity, all in normalized units. The additional soliton parameter is the complex-valued polarization state  $\rho \equiv \frac{q_1}{q_2}$  defined as the ( $z$ - and  $x$ -independent) ratio between the  $q_1$  and  $q_2$  components.

Figure 3 shows the schematic for a general two-soliton collision, with initial parameters  $\rho_1$ ,  $k_1$  and  $k_2$ , corresponding to the right-moving and left-moving solitons, respectively. The values of  $k_1$  and  $k_2$  remain constant during collision, but in general the polarization state changes. Let  $\rho_1$  and  $\rho_L$  denote the respective soliton states before impact and suppose the collision transforms  $\rho_1$  into  $\rho_R$  and  $\rho_L$  into  $\rho_2$ . It turns out that the state change undergone by each colliding soliton takes on the very simple form of a linear fractional transformation (also called a bilinear or Möbius transformation). Explicitly, the state of the emerging left-moving soliton is given by (Jakubowski et al. 1998)

$$\rho_2 = \frac{[(1-g)/\rho_1^* + \rho_1]\rho_L + g\rho_1/\rho_1^*}{g\rho_L + (1-g)\rho_1 + 1/\rho_1^*} \quad (5)$$

where

$$g = \frac{k_1 + k_1^*}{k_2 + k_1^*} \quad (6)$$

The state of the right-moving soliton is obtained similarly and is

$$\rho_R = \frac{[(1-h^*)/\rho_L^* + \rho_L]\rho_1 + h^*\rho_L/\rho_L^*}{h^*\rho_1 + (1-h^*)\rho_L + 1/\rho_L^*} \quad (7)$$

where

$$h \equiv \frac{k_2 + k_2^*}{k_1 + k_2^*} \quad (8)$$

We assume here, without loss of generality, that  $k_{1R}, k_{2R} > 0$ .

Several properties of the linear fractional transformations in Eqs. 5 and 7 are derived in Jakubowski et al. (1998), including the characterization of inverse operators, fixed points, and implicit forms. In particular, when viewed as an operator, every soliton has an *inverse*, which will undo the effect of the operator on the state. Note that this requires that the inverse operator have the same  $k$  parameter as the original, a condition that will hold in our application of computing in the next section.

These state transformations were first used by Jakubowski et al. (1998) to describe logical operations such as NOT. Later, Steiglitz (2000) established that arbitrary computation was possible through time gating of Manakov (1 + 1)-dimensional spatial solitons. We will describe this in section “[Manakov Soliton Computing](#).”

There exist several candidates for the physical realization of Manakov solitons, including photorefractive crystals (Anastassiou et al. 1999, 2001; Chen et al. 1996; Christodoulides et al. 1996; Shih and Segev 1996), semiconductor waveguides (Kang et al. 1996), quadratic media (Steblina et al. 2000), and optical fiber (Menyuk 1989; Rand et al. 2007). In section “[Experiments](#),” we discuss in detail an experiment with vector solitons in linearly birefringent optical fiber.

## Manakov Soliton Computing

We described in the previous section how collisions of Manakov solitons can be described by transformations of a complex-valued state which is the ratio between the two Manakov components. We show in this section that general computation is possible if we use (1 + 1)-dimensional spatial solitons that are governed by the Manakov equations and if we are allowed to time gate the

beams input to the medium. The result is a dynamic computer without spatially fixed gates or wires, which is unlike most present-day conceptions of a computer that involve integrated circuits, in which information travels between logical elements that are fixed spatially through fabrication on a silicon wafer. We can call such a scheme “nonlithographic,” in the sense that there is no architecture imprinted on the medium.

The requirements for computation include cascadability, fanout, and Boolean completeness. The first, cascadability, requires that the output of one device can serve as input to another. Since any useful computation consists of many stages of logic, this condition is essential. The second, fanout, refers to the ability of a logic gate to drive at least two similar gates. Finally, Boolean completeness makes it possible to perform arbitrary computation.

We should emphasize that although the model we use is meant to reflect known physical phenomena, at least in the limit of ideal behavior, the result is a mathematical one. Practical considerations of size and speed are not considered here, nor are questions of error propagation. In this sense the program of this article is analogous to Fredkin and Toffoli (1982) for ideal billiard balls and Shor (1994) for quantum mechanics. There are however several candidates for physical instantiation of the basic ideas in this paper, as noted in the previous section.

Although we are describing computation embedded in a homogeneous medium, and not interconnected *gates* in the usual sense of the word, we will nevertheless use the term *gates* to describe prearranged sequences of soliton collisions that affect logical operations. We will in fact adopt other computer terms to our purpose, such as *wiring* to represent the means of moving information from one place to another and *memory* to store it in certain ways for future use.

We will proceed in the construction of what amounts to a complete computer in the following stages: first we will describe a basic gate that can be used for FANOUT. Then we will show how the same basic configuration can be used for NOT and finally, NAND. Then we will describe ways to use time gating of the input beams to interconnect signals. The NAND gate, FANOUT, and interconnect signals are sufficient to

implement any computer, and we conclude with a layout scheme for a general-purpose and hence Turing-equivalent computer. The general picture of the physical arrangement is shown in Fig. 4.

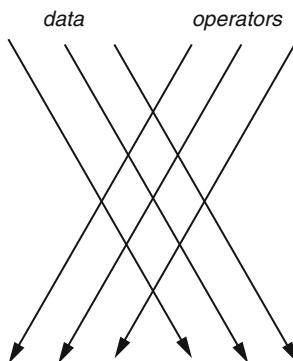
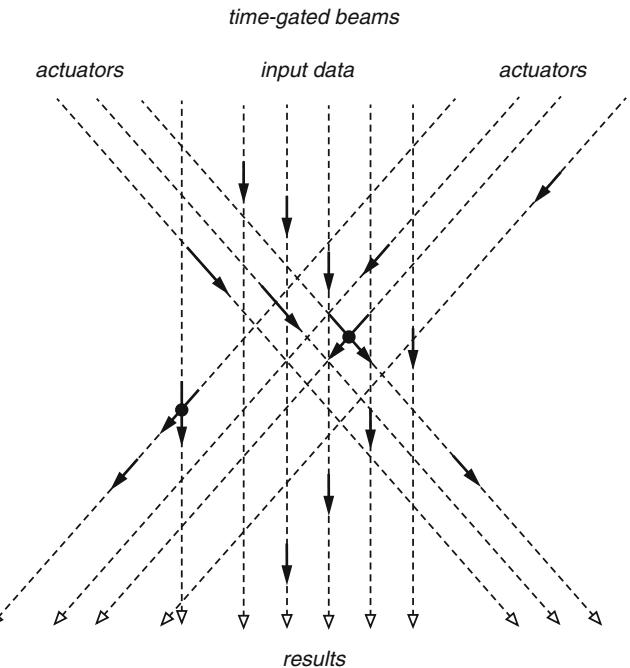
Figure 5 shows the usual picture of colliding solitons, which can work interchangeably for the case of temporal or spatial solitons. It is convenient for visualization purposes to turn the picture and adjust the scale so the axes are horizontal and vertical, as in Fig. 6. We will use binary logic, with two distinguished, distinct complex numbers representing TRUE and FALSE, called 1 and 0, respectively. In fact, it turns out to be possible to use complex 1 and 0 for these two state values, and we will do that throughout this paper, but this is a convenience and not at all a necessity. We will thus use complex polarization states 1 and 0 and logical 1 and 0 interchangeably.

## FANOUT

We construct the FANOUT gate by starting with a COPY gate, implemented with collisions between three down-moving, vertical solitons and one left-moving horizontal soliton. Figure 7 shows the arrangement. The soliton state labeled *in* will carry a logical value, and so be in one of the two states 0 or 1. The left-moving soliton labeled *actuator* will be in the fixed state 0, as will be the case throughout this paper. The plan is to adjust the (so far) arbitrary states *z* and *y* so that *out* = *in*, justifying the name COPY. It is reasonable to expect that this might be possible, because there are four degrees of freedom in the two complex numbers *z* and *y* and two complex equations to satisfy that *out* be 1 and 0 when *in* is 1 and 0, respectively. Values that satisfy these four equations in four unknowns were obtained numerically. We will call them *z<sub>c</sub>* and *y<sub>c</sub>*. It is not always possible to solve these equations; Ablowitz et al. (2004) showed that a unique solution is guaranteed to exist in certain parameter regimes. However, explicit solutions have been found for all the cases used in this section and are given in Table 1.

To be more specific about the design problem, write Eq. 5 as the left-moving product  $\rho_2 = L(\rho_1, \rho_L)$  and similarly write Eq. 7 as  $\rho_R = R(\rho_1, \rho_L)$ . The successive left-moving

**Computing with Solitons, Fig. 4** The general physical arrangement considered in this paper. Time-gated beams of spatial Manakov solitons enter at the top of the medium, and their collisions result in state changes that reflect computation. Each solid arrow represents a beam segment in a particular state (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

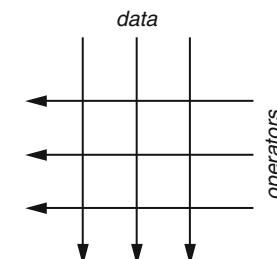


**Computing with Solitons, Fig. 5** Colliding spatial solitons (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

products in Fig. 7 are  $L(in, 0)$  and  $L(y, L(in, 0))$ . The *out* state is then  $R(z, L(y, L(in, 0)))$ . The stipulation that 0 maps to 0 and 1 maps to 1 is expressed by the following two simultaneous complex equations in two complex unknowns

$$\begin{aligned} R(z, L(y, L(0, 0))) &= 0, \\ R(z, L(y, L(1, 0))) &= 1. \end{aligned}$$

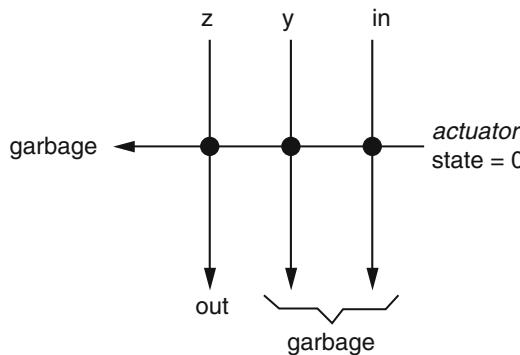
It is possible to solve for  $z$  as a function of  $y$  and then eliminate  $z$  from the equations, yielding one complex equation in one complex



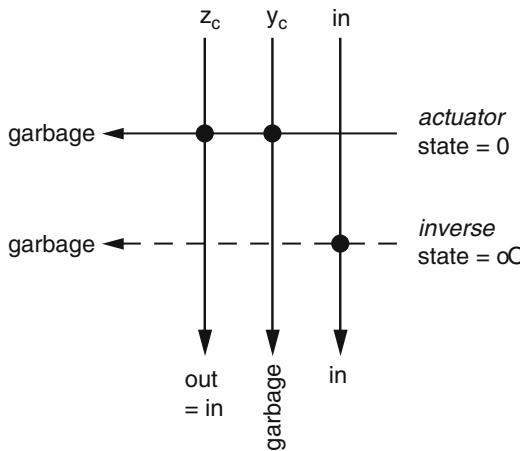
**Computing with Solitons, Fig. 6** Convenient representation of colliding spatial solitons (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

unknown  $y$ . This is then solved numerically by grid search and successive refinement. There is no need for efficiency here, since we will require solutions in only a small number of cases.

To make a FANOUT gate, we need to recover the input, which we can do using a collision with a soliton in the state which is the inverse of 0, namely  $\infty$  (Jakubowski et al. 1998). Figure 8 shows the complete FANOUT gate. Notice that we indicate collisions with a dot at the intersection of paths and require that the continuation of the inverse soliton do not intersect the continuation of  $z$  that it meets. We indicate that by a broken line and postpone the explanation of how this



**Computing with Solitons, Fig. 7** COPY gate (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)



**Computing with Solitons, Fig. 8** FANOUT gate (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

“wire crossing” is accomplished. It is immaterial whether the continuation of the inverse operator hits the continuation of  $y$ , because it is not used later. We call such solitons *garbage* solitons.

### NOT and ONE Gates

In the same way we designed the complex pair of states  $(z_c, y_c)$  to produce a COPY and FANOUT gate, we can find a pair  $(z_n, y_n)$  to get a NOT gate, mapping 0–1, and 1–0; and a pair  $(z_1, y_1)$  to get a ONE gate, mapping both 0 and 1–1. These  $(z, y)$  values are given in Table 1.

We should point out that the ONE gate in itself considered as a one-input, one-output gate is not invertible and could never be achieved by using the continuation of one particular soliton through one, or even many collisions. This is because such transformations are always nonsingular linear fractional transformations, which are invertible (Jakubowski et al. 1998). The transformation of state from the input to the continuation of  $z$  is, however, much more complicated and provides the flexibility we need to get the ONE gate. It turns out that this ONE gate will give us a row in the truth table of a NAND and is critical for realizing general logic.

### Output/Input Converters, Two-Input Gates, and NAND

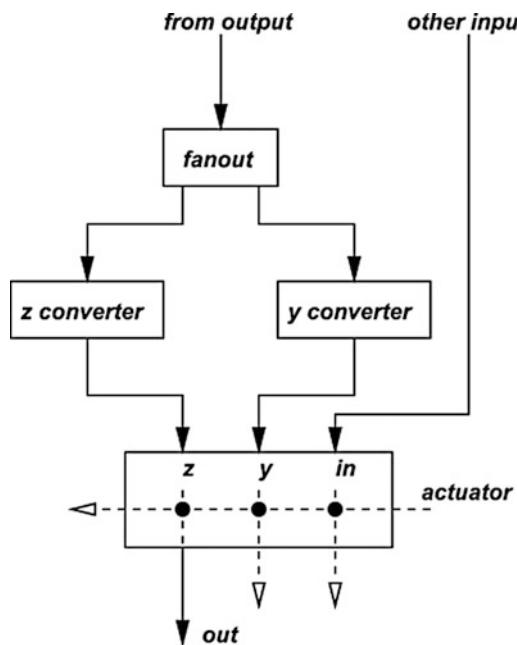
To perform logic of any generality, we must of course be able to use the output of one operation as the input to another. To do this we need to convert logic (0/1) values to some predetermined  $z$  and  $y$  values, the choice depending on the type of gate we want. This results in a two-input, one-output gate.

As an important example, here is how a NAND gate can be constructed. We design a  $z$  converter that converts 0/1 values to appropriate values of  $z$ , using the basic three-collision arrangement shown in Fig. 7. For a NAND gate, we map 0 to  $z_1$ , the  $z$  value for the ONE gate, and map 1 to  $z_n$ , the  $z$  value for the NOT gate. Similarly, we construct a  $y$  converter that maps 0 to  $y_1$  and 1 to  $y_n$ . These  $z$  and  $y$  converters are used on the fanout of one of the inputs, and the resulting two-input gate is shown in Fig. 9. Of course these  $z$  and  $y$  converters require  $z$  and  $y$  values themselves, which are again determined by numerical search (see Table 1).

The net effect is that when the left input is 0, the other input is mapped by a ONE gate, and when it is 1 the other input is mapped by a NOT gate. The only way the output can be 0 is if both inputs are 1, thus showing that this is a NAND gate. Another way of looking at this construction is that the  $2 \times 2$  truth table of (left input)  $\times$  (right input) has as its 0 row a ONE gate of the

**Computing with Solitons, Table 1** Parameters for gates when soliton speeds are 1

Gate	$z$	$y$
COPY	$-0.24896731 - 0.62158212 \cdot I$	$2.28774210 + 0.01318152 \cdot I$
NOT	$-0.17620885 + 0.38170630 \cdot I$	$0.07888703 - 1.26450654 \cdot I$
ONE	$-0.45501471 - 1.37634227 \cdot I$	$1.43987094 + 0.64061349 \cdot I$
Z-CONV	$0.31838068 - 0.43078735 \cdot I$	$-0.04232340 + 2.17536612 \cdot I$
Y-CONV	$1.37286955 + 0.88495501 \cdot I$	$-0.58835758 - 0.18026939 \cdot I$

**Computing with Solitons, Fig. 9** A NAND gate, using converter gates to couple copies of one of its inputs to its  $z$  and  $y$  parameters (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

columns (1 1) and as its 1 row a NOT gate of the columns (1 0).

The importance of the NAND gate is that it is *universal* (Mano 1972). That is, it can be used with interconnects and fanouts to construct any other logical function. Thus we have shown that with the ability to “wire,” we can implement any logic using the Manakov model.

We note that other choices of input converters result in direct realizations of other gates. Using input converters that convert 0 and 1 to  $(z_c, y_c)$  and  $(z_n, y_n)$ , respectively, results in a truth table with first row (0 1) and second row (1 0), an XOR

gate. Converting 0 and 1 to  $(z_c, y_c)$  and  $(z_1, y_1)$ , respectively, results in an OR gate and so on.

### Time Gating

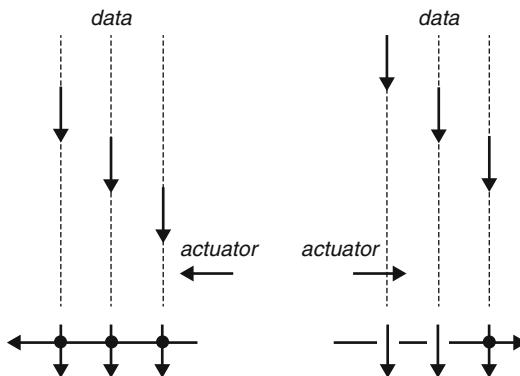
We next take up the question of interconnecting the gates described above and begin by showing how the continuation of the input in the COPY gate can be restored without affecting the other signals. In other words, we show how a simple “wire crossing” can be accomplished in this case.

For spatial solitons, the key flexibility in the model is provided by assuming that input beams can be time gated, that is, turned on and off. When a beam is thus gated, a finite segment of light is created that travels through the medium. We can think of these finite segments as finite light pulses, and we will call them simply *pulses* in the remainder of this article.

Figure 10a shows the basic three-collision gate implemented with pulses. Assuming that the actuator and data pulses are appropriately timed, the actuator pulse hits all three data pulses, as indicated in the projection below the space-space diagram. The problem is that if we want a later actuator pulse to hit the rightmost data pulse (to invert the state, for example, as in the FANOUT gate), it will also hit the remaining two data pulses because of the way they must be spaced for the earlier three collisions.

We can overcome this difficulty by sending the actuator pulse from the left instead of the right. Timing it appropriately early, it can be made to miss the first two data pulses and hit the third, as shown in Fig. 10b. It is easy to check that if the velocity of the right-moving actuator solitons is algebraically above that of the data solitons by the same amount that the velocity of

the data solitons is algebraically above that of the left-moving actuator solitons, the same state transformations will result. For example, if we choose the velocities of the data and left-moving actuator solitons to be +1 and  $-1$ , we should choose the velocity of the right-moving actuator solitons to be  $+3$ . This is really a consequence of the fact that the  $g$  and  $h$  parameters of Eqs. 6 and 8 in the linear fractional transformation depend only on the difference in the velocities of the colliding solitons.



**Computing with Solitons, Fig. 10** (a) When entered from the *right* and properly timed, the actuator pulse hits all three data pulses, as indicated in the projection at the *bottom*; (b) When entered from the *left* and properly timed, the actuator pulse misses two data pulses and hits only the rightmost data pulse, as indicated in the projection at the *bottom* (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

#### Computing with Solitons,

**Fig. 11** The frame of this figure is moving down with the data pulses on the *left*. A data pulse in memory is operated on with a three-collision gate actuated from the *left* and the result deposited to the *upper right* (Reprinted with permission from Steiglitz (2000)).

Copyright by the American Physical Society

## Wiring

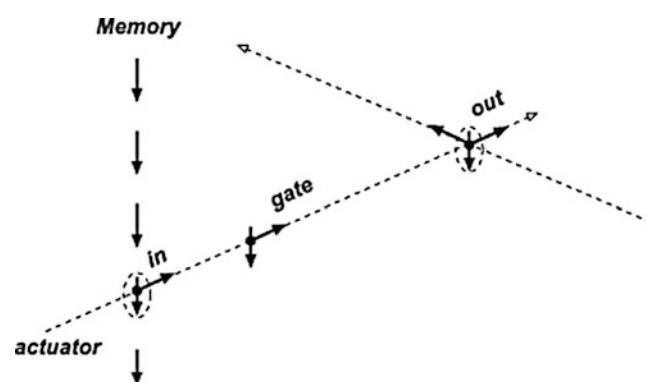
Having shown that we can perform FANOUT and NAND, it remains only to show that we can “wire” gates so that any outputs can be fed to any inputs. The basic method for doing this is illustrated in Fig. 11. We think of data as stored in the down-moving pulses in a column, which we can think of as “memory.” The observer moves with this frame, so the data appears stationary.

Pulses that are horizontal in the three-collision gates shown in previous figures will then appear to the observer to move upward at inclined angles. It is important to notice that these upward diagonally moving pulses are evanescent in our picture (and hence their paths are shown dashed in the figure). That is, once they are used, they do not remain in the picture with a moving frame and hence cannot interfere with later computations.

However, all vertically moving pulses remain stationary in this picture.

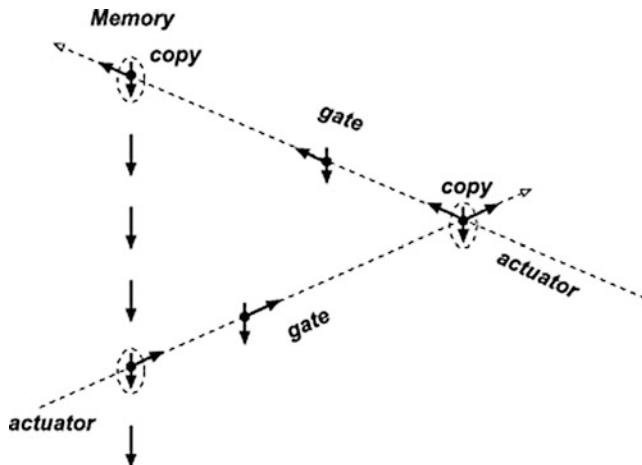
Once a diagonal trajectory is used for a three-collision gate, reusing it will in general corrupt the states of all the stationary pulses along that diagonal. However, the original data pulse (gate input) can be restored with a pulse in the state inverse to the actuator, either along the same diagonal as the actuator, provided we allow enough time for the result (the gate output, a stationary  $z$  pulse) to be used, or along the other diagonal.

Suppose we want to start with a given data pulse in the memory column and create two



### Computing with Solitons, Fig. 12

A data pulse is copied to the *upper right*; this copy is copied to the *upper left* and the result put at the *top* of memory. The original data pulse can then be restored with an inverse pulse and copied to the *left* in the same way (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)



copies above it in the memory column. Figure 12 shows a data pulse at the lower left being copied to the upper right with a three- collision COPY gate, initiated with an actuator pulse from the left. This copy is then copied again to the upper left, back to a waiting  $z$  pulse in the memory column. After the first copy is used, an inverse pulse can be used along the lower left to upper right diagonal to restore the original data pulse. The restored data pulse can then be copied to the left in the same way, to a height above the first copy, say, and thus two copies can be created and deposited in memory above the original.

### A Second Speed and Final FANOUT and NAND

There is one problem still remaining with a true FANOUT: when an original data pulse in memory is used in a COPY operation for FANOUT, two diagonals are available, one from the lower left to the upper right and the other from the lower right to the upper left. Thus, two copies can be made, as was just illustrated. However, when a data pulse is deposited in the memory column as a result of a logic operation, the logical operation itself uses at least one diagonal, which leaves at most one free. This makes a FANOUT of the *output* of a gate impossible with the current scheme.

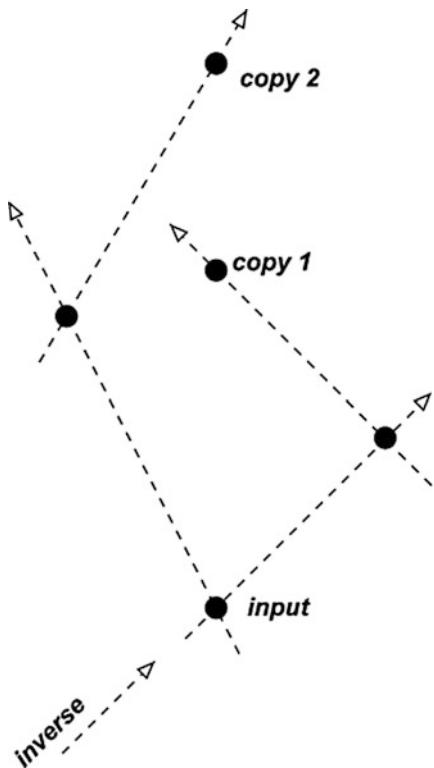
A simple solution to this problem is to introduce another speed, using velocities  $\pm 0.5$ , say, in addition to  $\pm 1$ .

This effectively provides four rather than two directions in which a pulse can be operated on and allows true FANOUT and general interconnections. Figure 13 shows such a FANOUT; the data pulse at the lower left is copied to a position above it using one speed and to another position, above that, using another.

Finally, a complete NAND gate is shown in Fig. 14. The gate can be thought of as composed of the following steps:

- Input 2 is copied to the *upper left* and that copy transformed by a  $z$  converter to the *upper right*, placing the  $z$  pulse for the NAND gate at the *top* of the figure.
- After the copy of input 2 is used, input 2 is restored with an inverse pulse to the *upper left*.
- Input 2 is then transformed to the *upper right* by a  $y$  converter.
- Input 1 is copied to the *upper right*, to a position collinear with the  $z$ - and  $y$ -converted versions of the other input.
- A final actuator pulse converts the  $z$  pulse at the *top* to the output of the NAND gate.

Note that the output of the NAND has used two diagonals, which again shows why a second speed is needed if we are to use the NAND output



**Computing with Solitons, Fig. 13** The introduction of a second speed makes true FANOUT possible. For simplicity, in this and the next figure, data and operator pulses are indicated by solid dots, and the  $y$  operator pulses are not shown. The paths of actuator pulses are indicated by dashed lines (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

as an input to subsequent logical operations. The  $y$  operator pulses, middle components in the three-collision COPY and converter gates, are not shown in the figure, but room can always be made for them to avoid accidental collisions by adding only a constant amount of space (Fig. 14).

## Universality

It should be clear now that any sequence of three-collision gates can be implemented in this way, copying data out of the memory column to the upper left, or right, and performing NAND operations on any two at a time in the way shown in the previous section. The computation can

proceed in a breadth-first manner, with the results of each successive stage being stored above the earlier results. Each additional gate can add only a constant amount of height and width to the medium, so the total area required is no more than proportional to the square of the number of gates.

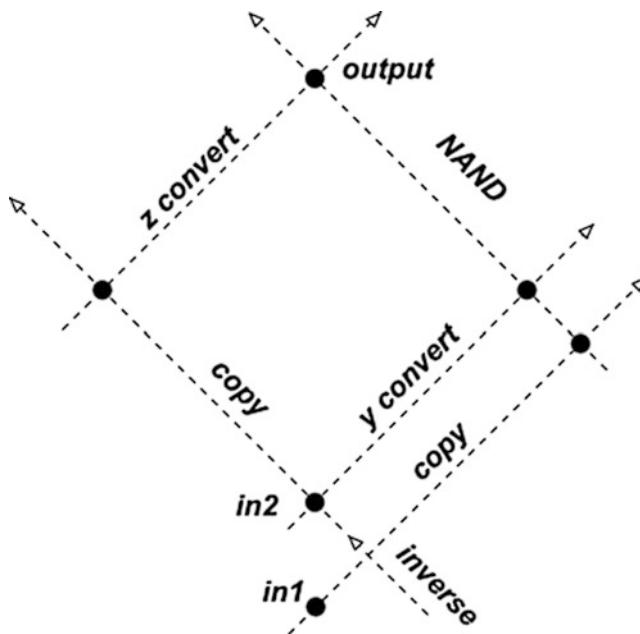
The “program” consists of down-moving  $y$  and  $z$  operator pulses, entering at the top with the down-moving data, and actuator pulses that enter from the left or right at two different speeds. In the frame moving with the data, the data and operator pulses are stationary, and new results are deposited at the top of the memory column. In the laboratory frame, the data pulses leave the medium downward, and new results appear in the medium at positions above the old data, at the positions of newly entering  $z$  pulses.

## Discussion

We have shown that in principle, any computation can be performed by shining time-gated lasers into a completely homogeneous nonlinear optical medium. This result should be viewed as mathematical, and whether the physics of vector soliton collisions can lead to practical computational devices is a subject for future study. With regard to the economy of the model, the question of whether time gating is necessary, or even whether two speeds are necessary, is open.

We note that the result described here differs from the universality results for the ideal billiard ball model (Fredkin and Toffoli 1982) the Game of Life (Berlekamp et al. 1982), and Lattice Gases (Squier and Steiglitz 1993), for example, in that no internal mirrors or structures of any kind are used inside the medium. To the authors’ knowledge, whether internal structure is necessary in these other cases is open.

Finally, we remark that the model used is reversible and dissipationless. The fact that some of the gate operations realized are not in themselves reversible is not a contradiction, since extra, “garbage” solitons (Fredkin and Toffoli 1982) are produced that save enough state to run the computation backward.



**Computing with Solitons, Fig. 14** Implementation of a NAND gate. A second speed will be necessary to use the output (Reprinted with permission from Steiglitz (2000). Copyright by the American Physical Society)

### Multistable Soliton Collision Cycles

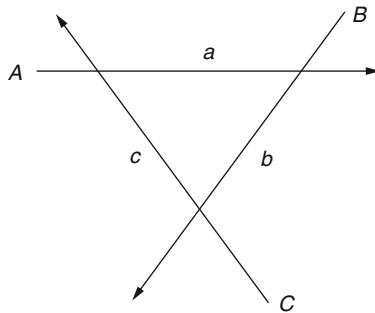
Bistable and multistable optical systems, besides being of some theoretical interest, are of practical importance in offering a natural “flip-flop” for noise immune storage and logic. We show in this section that simple cycles of collisions of solitons governed by the Manakov equations can have more than one distinct stable set of polarization states, and therefore these distinct equilibria can, in theory, be used to store and process information. The multistability occurs in the polarization states of the beams; the solitons themselves do not change shape and remain the usual sech-shaped solutions of the Manakov equations. This phenomenon is dependent only on simple soliton collisions in a completely homogeneous medium.

The basic configuration considered requires only that the beams form a closed cycle and can thus be realized in any nonlinear optical medium that supports spatial Manakov solitons. The possibility of using multistable systems of beam collisions broadens the possibilities

for practical application of the surprisingly strong interactions that Manakov solitons can exhibit, a phenomenon originally described in Radhakrishnan et al. (1997). We show here by example that a cycle of three collisions can have two distinct foci surrounded by basins of attractions and that a cycle of four collisions can have three.

### The Basic Three-Cycle and Computational Experiments

Figure 15 shows the simplest example of the basic scheme, a cycle of three beams, entering in states  $A$ ,  $B$ , and  $C$ , with intermediate beams  $a$ ,  $b$ , and  $c$ . For convenience, we will refer to the beams themselves, as well as their states, as  $A$ ,  $B$ ,  $C$ , etc. Suppose we start with beam  $C$  initially turned off, so that  $A = a$ . Beam  $a$  then hits  $B$ , thereby transforming it to state  $b$ . If beam  $C$  is then turned on, it will hit  $A$ , closing the cycle. Beam  $a$  is the changed, changing  $b$ , etc., and the cycle of state changes propagates clockwise. The question we

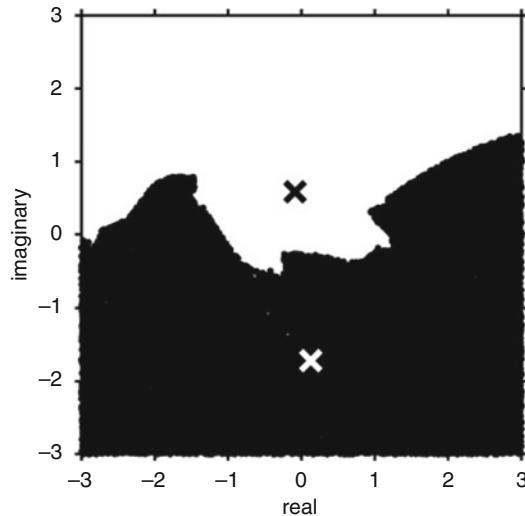


**Computing with Solitons, Fig. 15** The basic cycle of three collisions (Reprinted with permission from Steiglitz (2001). Copyright by the American Physical Society)

ask is whether this cycle converges and, if so, whether it will converge with any particular choice of complex parameters to exactly zero, one, two, or more foci. We answer the question with numerical simulations of this cycle.

A typical computational experiment was designed by fixing the input beams  $A$ ,  $B$ ,  $C$ , and the parameters  $k_1$  and  $k_2$  and then choosing points  $a$  randomly and independently with real and imaginary coordinates uniformly distributed in squares of a given size in the complex plane. The cycle described above was then carried out until convergence in the complex numbers  $a$ ,  $b$ , and  $c$  was obtained to within  $10^{-12}$  in norm. Distinct foci of convergence were stored, and the initial starting points  $a$  were categorized by which focus they converged to, thus generating the usual picture of basins of attraction for the parameter  $a$ . Typically this was done for 50,000 random initial values of  $a$ , effectively filling in the square, for a variety of parameter choices  $A$ ,  $B$ , and  $C$ . The following results were observed:

- In cases with one or two clear foci, convergence was obtained in every iteration, almost always within 1 or 200 iterations.
- Each experiment yielded exactly one or two foci.
- The bistable cases (two foci) are somewhat less common than the cases with a unique focus and are characterized by values of  $k_r$  between about 3 and 5 when the velocity difference  $A$  was fixed at 2.

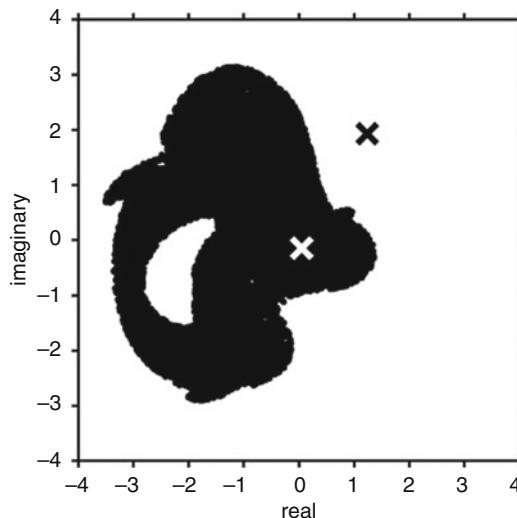


**Computing with Solitons, Fig. 16** The two foci and their corresponding basins of attraction in the first example, which uses a cycle of three collisions. The states of the input beams are  $A = -0.8 - i \cdot 0.13$ ,  $B = 0.4 - i \cdot 0.13$ ,  $C = 0.5 + i \cdot 1.6$ ; and  $k = 4 \pm i$  (Reprinted with permission from Steiglitz (2001). Copyright by the American Physical Society)

Figure 16 shows a bistable example, with the two foci and their corresponding basins of attraction. The parameter  $k$  is fixed in this and all subsequent examples at  $4 \pm i$  for the right- and left-moving beams of any given collision, respectively. The second example, shown in Fig. 17, shows that the basins are not always simply connected; a sizable island that maps to the upper focus appears within the basin of the lower focus.

## A Tristable Example Using a Four-Cycle

Collision cycles of length four seems to exhibit more complex behavior than those of length three, although it is difficult to draw any definite conclusions because the parameter spaces are too large to be explored exhaustively, and there is at present no theory to predict such highly nonlinear behavior. If one real degree of freedom is varied as a control parameter, we can move from bistable to tristable solutions, with a regime between in which one basin of attraction disintegrates into many small separated fragments. Clearly, this model is complex enough to exhibit many of the well-known features of nonlinear systems.

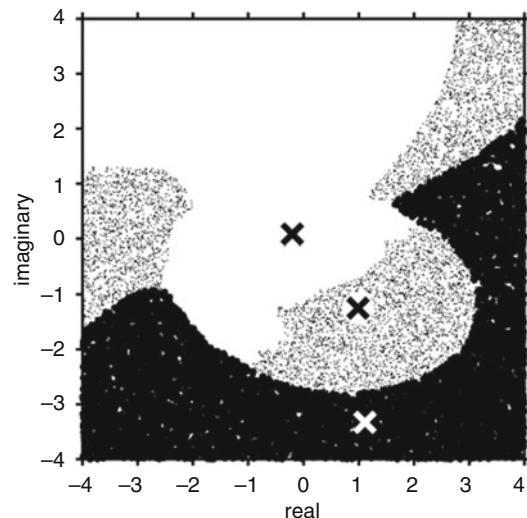


**Computing with Solitons, Fig. 17** A second example using a cycle of three collisions, showing that the basins need not be simply connected. The states of the input beams are  $A = 0.7 - i \cdot 0.3$ ,  $B = -1.1 - i \cdot 0.5$ ,  $C = 0.4 + i \cdot 0.81$ ; and  $k = 4 \pm i$  (Reprinted with permission from Steiglitz (2001). Copyright by the American Physical Society)

Fortunately, it is not difficult to find choices of parameters that result in very well-behaved multistable solutions. For example, Fig. 18 shows such a tristable case. The smallest distance from a focus to a neighboring basin is on the order of 25% of the interfocus distance, indicating that these equilibria will be stable under reasonable noise perturbations.

## Discussion

The general phenomenon discussed in this section raises many questions, both of a theoretical and practical nature. The fact that there are simple polarization-multistable cycles of collisions in a Manakov system suggests that similar situations occur in other vector systems, such as photorefractive crystals or birefringent fiber. Any vector system with the possibility of a closed cycle of soliton collisions becomes a candidate for multistability, and there is at this point really no compelling reason to restrict attention to the Manakov case, except for the fact that the explicit



**Computing with Solitons, Fig. 18** A case with three stable foci, for a collision cycle of length four. The states of the input beams are  $A = -0.39 - i \cdot 0.45$ ,  $B = 0.22 - i \cdot 0.25$ ,  $C = 0.0 + i \cdot 0.25$ ,  $D = -0.51 + i \cdot 0.48$ , and  $k = 4 \pm i$  (Reprinted with permission from Steiglitz (2001). Copyright by the American Physical Society)

state-change relations make numerical study much easier.

The simplified picture we used of information traveling clockwise after we begin with a given beam  $a$  gives us stable polarization states when it converges, plus an idea of the size of their basins of attractions. It is remarkable that in all cases in our computational experience, except for borderline transitional cases in going from two to three foci in a four cycle, this circular process converges consistently and quickly. But understanding the actual dynamics and convergence characteristics in a real material requires careful physical modeling. This modeling will depend on the nature of the medium used to approximate the Manakov system and is left for future work. The implementation of a practical way to switch from one stable state to another is likewise critically dependent on the dynamics of soliton formation and perturbation in the particular material at hand and must be studied with reference to a particular physical realization.

We remark also that no ironclad conclusions can be drawn from computational experiments about the numbers of foci in any particular case or the

number possible for a given size cycle – despite the fact that we regularly used 50,000 random starting points. On the other hand, the clear cases that have been found, such as those used as examples, are very characteristic of universal behavior in other nonlinear iterated maps and are sufficient to establish that bi- and tristability, and perhaps higher-mode multistability is a genuine mathematical characteristic and possibly also physically realizable. It strongly suggests experimental exploration.

We restricted discussion in this section to the simplest possible structure of a single closed cycle, with three or four collisions. The stable solutions of more complicated configurations are the subject of continuing study. A general theory that predicts this behavior is lacking and it seems at this point unlikely to be forthcoming. This forces us to rely on numerical studies, from which, as we point out above, only certain kinds of conclusions can be drawn. We are fortunate, however, in being able to find cases that look familiar and which are potentially useful, like the bistable three cycles with well-separated foci and simply connected basins of attraction.

It is not clear, however, just what algorithms might be used to find equilibria in collision topologies with more than one cycle. It is also intriguing to speculate about how collision configurations with particular characteristics can be designed, how they can be made to interact, and how they might be controlled by pulsed beams. There is promise that when the ramifications of complexes of vector soliton collisions are more fully understood, they might be useful for real computation in certain situations.

## Application to Noise-Immune Soliton Computing

Any physical instantiation of a computing technology must be designed to be immune from the effects of noise buildup from logic stage to logic stage. In the familiar computers of today, built with solid-state transistors, the noise immunity is provided by physical state restoration, so that voltage levels representing logical “0” and “1” are restored by bistable circuit mechanisms at

successive logic stages. This is state restoration at the physical level.

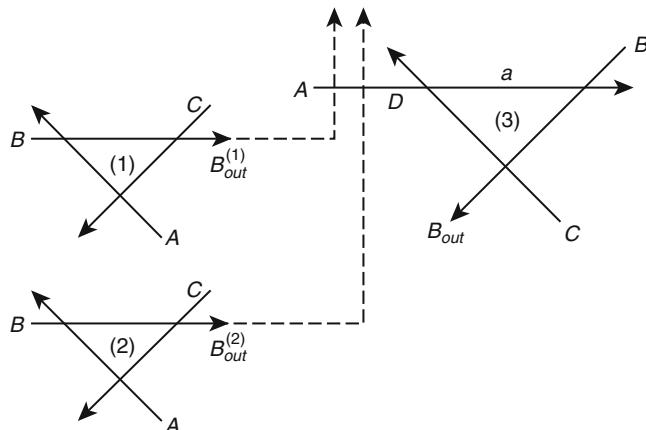
For another example, proposed schemes for quantum computing would be impractical without some means of protecting information stored in qubits from inevitable corruption by the rest of the world. The most common method proposed for accomplishing this is error correction at the software level and state restoration at the logical level.

In the collision-based scheme for computing with Manakov solitons described in section “[Manakov Soliton Computing](#),” there is no protection against buildup of error from stage to stage, and some sort of logical state restoration would be necessary in a practical realization. The bistable collision cycles of Manakov solitons described in this section, however, offer a natural computational building block for soliton computation with physical state restoration. This idea is explored in Rand et al. (2005). Figure 19 illustrates the approach with a schematic diagram of a NAND gate, implemented with bistable cycles to represent bits. The input bits are stored in the collision cycles (1) and (2), which have output beams that can be made to collide with input beam A of cycle (3), which represents the output bit of the gate. These inputs to the gate, shown as dashed lines, change the state of beam A of the ordinarily bistable cycle (3) so that it becomes *monostable*. The state of cycle (3) is then steered to a known state. When the input beams are turned off, cycle (3) returns to its normal bistable condition, but with a known input state. Its state then evolves to 1 of 2 bits, and the whole system of three collision cycles can be engineered so that the final state of cycle (3) is the NAND of the two bits represented by input cycles (1) and (2) (see Rand et al. 2005 for details).

A computer based on such bistable collision cycles is closer in spirit to present-day ordinary transistor-based computers, with a natural noise immunity and state restoration based on physical bistability. As mentioned in the previous subsection, however, the basic bistable cycle phenomenon awaits laboratory verification, and much remains to be learned about the dynamics, and eventual speed and reliability of such systems.

**Computing  
with Solitons,**

**Fig. 19** Schematic of NAMED gate using bistable collision cycles (Reprinted with permission from Rand et al. (2005). Copyright by Old City Publishing)



## Experiments

The computation schemes described in the previous sections obviously rely on the correct mathematical modeling of the physics proposed for realization. We next describe experiments that verify some of the required soliton phenomenology in optical fibers. Specifically, we highlight the experimental observation of temporal vector soliton propagation and collision in a birefringent optical fiber (Rand et al. 2007). This is both the first demonstration of temporal vector solitons with two mutually incoherent component fields and of vector soliton collisions in a Kerr nonlinear medium.

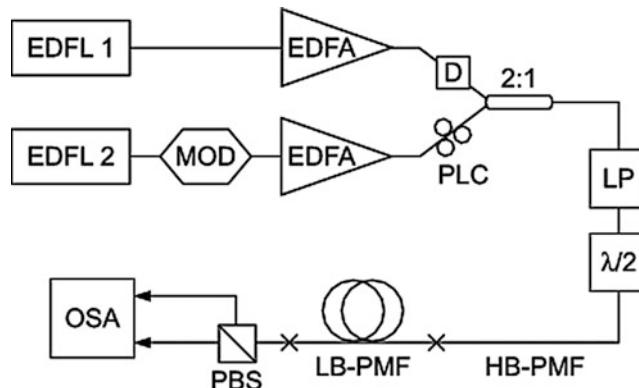
Temporal soliton pulses in optical fiber were first predicted by Hasegawa and Tappert (1973), followed by the first experimental observation by Mollenauer et al. (1980). In subsequent work, Menyuk accounted for the birefringence in polarization-maintaining fiber (PMF) and predicted that vector solitons, in which two orthogonally polarized components trap each other, are stable under the proper operating conditions (Rand et al. 2005; Menyuk 1988). For birefringent fibers, self-trapping of two orthogonally polarized pulses can occur when XPM-induced nonlinearity compensates the birefringence-induced group velocity difference, causing the pulse in the fiber's fast axis to slow down and the pulse in the slow axis to speed up. The first demonstration of temporal soliton trapping was performed in the

sub-picosecond regime (Islam et al. 1989), in which additional ultrashort pulse effects such as Raman scattering are present. In particular, this effect results in a redshift that is linearly proportional to the propagation distance, as observed in a later temporal soliton trapping experiment (Nishizawa and Goto 2002). Recently, soliton trapping in the picosecond regime was observed with equal amplitude pulses (Korolev et al. 2005); however, vector soliton propagation could not be shown, because the pulses propagated for less than 1.5 dispersion lengths. In other work, phase-locked vector solitons in a weakly birefringent fiber laser cavity with nonlinear coherent coupling between components were observed (Cundiff et al. 1999).

The theoretical model for linearly birefringent fiber is the following coupled nonlinear Schrodinger equation (CNLSE):

$$\begin{aligned} i\left(\frac{\partial A_x}{\partial z} + \beta_{1x}\frac{\partial A_x}{\partial t}\right) - \frac{\beta_2}{2}\frac{\partial^2 A_x}{\partial t^2} \\ + \gamma\left(|A_x|^2 + \frac{2}{3}|A_y|^2\right)A_x = 0, \\ i\left(\frac{\partial A_y}{\partial z} + \beta_{1y}\frac{\partial A_y}{\partial t}\right) - \frac{\beta_2}{2}\frac{\partial^2 A_y}{\partial t^2} \\ + \gamma\left(|A_y|^2 + \frac{2}{3}|A_x|^2\right)A_y = 0, \end{aligned} \quad (9)$$

where  $t$  is the local time of the pulse,  $z$  is propagation distance along the fiber, and  $A_{x,y}$  is the



**Computing with Solitons, Fig. 20** Experimental setup. *EDFL* erbium-doped fiber laser, *EDFA* erbium-doped fiber amplifier, *MOD* modulator, *D* tunable delay line, *PLC* polarization loop controller, 2:1 fiber coupler, *LP* linear polarizer,  $\lambda/2$  half-wave plate, *HB-PMF* and *LB-PMF*

*LB-PMF* high- and low-birefringence polarization-maintaining fiber, *PBS* polarization beam splitter, *OSA* optical spectrum analyzer (Reprinted with permission from Rand et al. (2007). Copyright by the American Physical Society)

slowly varying pulse envelope for each polarization component. The parameter  $\beta_{1x, y}$  is the group velocity associated with each fiber axis, and  $\beta_2$  represents the group velocity dispersion, assumed equal for both polarizations. In addition, we neglect higher-order dispersion and assume a loss less medium with an instantaneous electronic response, valid for picosecond pulses propagating in optical fiber.

The last two terms of Eq. 9 account for the nonlinearity due to SPM and XPM, respectively. In linearly birefringent optical fiber, a ratio of 2/3 exists between these two terms. When this ratio is equal to unity, the CNLSE becomes the integrable Manakov system of Eq. 4. On the other hand, solutions of Eq. 9 are, strictly speaking, solitary waves, not solitons. However, it was found in Yang (1997) that the family of symmetric, single-humped (fundamental or first-order) solutions, to which the current investigation in this section belongs, are all stable. Higher-order solitons, characterized by multiple humps, are unstable. Furthermore, it was shown in Yang (1999) that collisions of solitary waves in Eq. 9 can be described by application of perturbation theory to the integrable Manakov equations, indicating the similarities between the characteristics of these two systems.

## Experimental Setup and Design

The experimental setup is shown in Fig. 20. We synchronized two actively mode-locked, erbium-doped fiber lasers (EDFLs)-EDFL1 at 1.25 GHz repetition rate and EDFL2 at 5 GHz; EDFL2 was modulated to match with the lower repetition rate of EDFL1. Each pulse train, consisting of 2 ps pulses, was amplified in an erbium-doped fiber amplifier (EDFA) and combined in a fiber coupler. To align polarizations, a polarization loop controller (PLC) was used in one arm, and a tunable delay line (TDL) was employed to temporally align the pulses for collision. Once combined, both pulse trains passed through a linear polarizer (LP) and a half-wave plate to control the input polarization to the PMF. Approximately 2 m of high-birefringence (HB) PMF preceded the specially designed 500 m of low-birefringence (LB) PMF used to propagate vector solitons. Although this short length of HB-PMF will introduce some pulse splitting (on the order of 2–3 ps), the birefringent axes of the HB- and LB-PMF were swapped in order to counteract this effect. At the output, each component of the vector soliton was then split at a polarization beam splitter, followed by an optical spectrum analyzer (OSA) for measurement.

The design of the LB-PMF required careful control over three characteristic length scales: the (polarization) beat length, dispersion length  $L_d$ , and nonlinear length  $L_{nl}$ . A beat length  $L_b = \lambda/\Delta n = 50$  cm was chosen at a wavelength of 1,550 nm, where  $n$  is the fiber birefringence. According to the approximate stability criterion of Cao and McKinstrie (1993), this choice allows stable propagation of picosecond vector solitons. By avoiding the sub-picosecond regime, ultrashort pulse effects such as intrapulse Raman scattering will not be present. The dispersion  $D = 2\pi c\beta_2/\lambda^2 = 16$  ps/km/nm and  $L_d = 2T_0^2/|\beta_2| \approx 70$  m, where  $T_0 = T_{FWHM}/1.763$ , are characteristic pulse width related to the full width at half maximum (FWHM) pulse width. Since  $L_d \gg L_b$ , degenerate four-wave mixing due to coherent coupling between the two polarization components can be neglected (Menyuk 1989). Furthermore, the total propagation distance is greater than 7 dispersion lengths.

Polarization instability, in which the fast axis component is unstable, occurs when  $L_{nl} = (\gamma P)^{-1}$  is of the same order of magnitude or smaller than  $L_b$ , as observed in Barad and Silberberg (1997). The nonlinearity parameter  $\gamma = 2\pi n_2/\lambda A_{eff} = 1.3(\text{km W})^{-1}$ , with Kerr nonlinearity coefficient  $n_2 = 2.6 \times 10^{-20} \text{ m}^2/\text{W}$  and measured effective mode area  $A_{eff} = 83 \mu\text{m}^2$ . In the LB-PMF, the fundamental vector soliton power is  $P \approx 14$  W; thus  $L_{nl} = 55$  m  $\gg L_b$  mitigating the effect of polarization instability.

## Vector Soliton Propagation

We first studied propagation of vector solitons using both lasers independently. The wavelength shift for each component is shown in Fig. 21a as a function of the input polarization angle  $\phi$ , controlled through the half-wave plate. Due to the anomalous dispersion of the fiber at this wavelength, the component in the slow (fast) axis will shift to shorter (longer) wavelengths to compensate the birefringence. The total amount of wavelength shift between components  $\Delta\lambda_{xy} = \Delta\beta_1/D = 0.64$  nm where  $\Delta\beta_1 = |\beta_{1x} - \beta_{1y}| = 10.3$  ps/km is the birefringence-

induced group velocity difference and dispersion  $D = 2\pi c\beta_2/\lambda^2 = 16$  ps/km/nm.

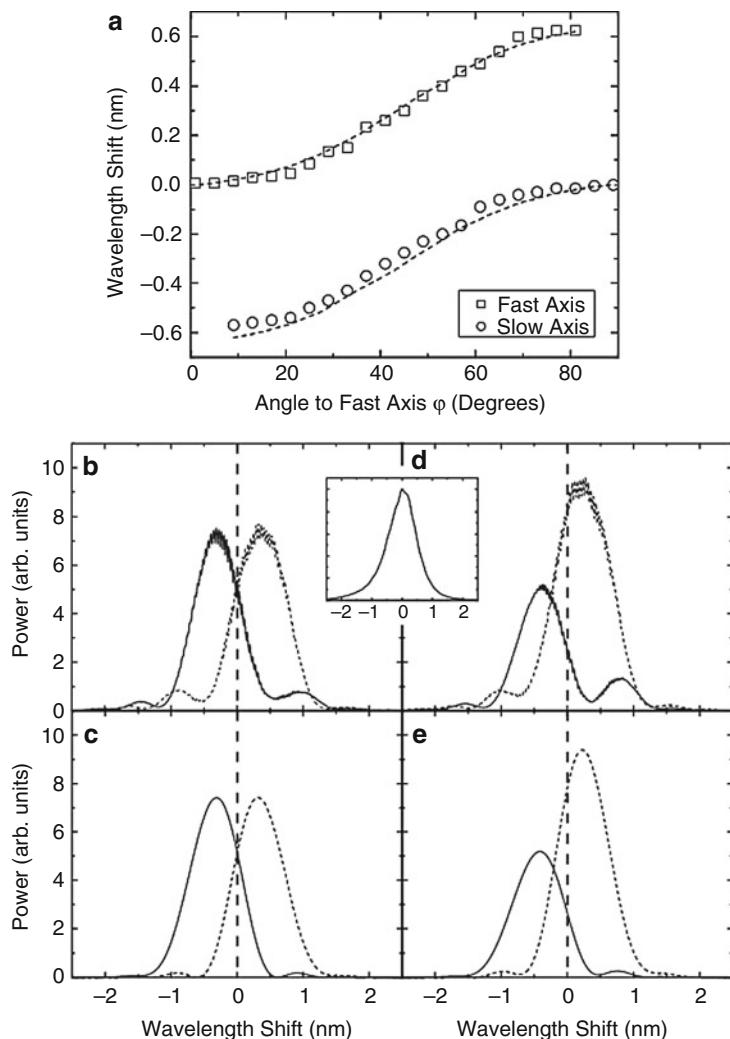
As  $\phi$  approaches 0° (90°), the vector soliton approaches the scalar soliton limit, and the fast (slow) axis does not shift in wavelength, as expected. At  $\phi = 45^\circ$ , a symmetric shift results. For unequal amplitude solitons, the smaller component shifts more in wavelength than the larger component, because the former experiences more XPM. Numerical simulations of Eq. 9, given by the dashed lines of Fig. 21a, agree very well with the experimental results. Also shown in Fig. 21 are two cases,  $\phi = 45^\circ$  and 37°, as well as the numerical prediction. The experimental spectra show some oscillatory features at 5 GHz, which are a modulation of the EDFL2 repetition rate on the optical spectrum. A sample input pulse spectrum from EDFL1 is shown in the inset of Fig. 21, which shows no modulation due to the limited resolution of the OSA. Vector solitons from both lasers produced similar results. In this and all subsequent plots in this section, the slow and fast axis components are depicted by solid and dashed lines, respectively.

As the two component amplitudes become more unequal, satellite peaks become more pronounced in the smaller component. These features are also present in the simulations, but are not as dominant (cf. Fig. 21d, e). We attribute this to the input pulse, which is calibrated for the  $\phi = 45^\circ$  case, because the power threshold for vector soliton formation in this case is largest due to the 2/3 factor between SPM and XPM nonlinear terms in the CNLSE. As the input is rotated toward unequal components, there will be extra power in the input pulse, which will radiate in the form of dispersive waves as the vector soliton forms. Due to the nature of this system, these dispersive waves can be nonlinearly trapped, giving rise to the satellite features in the optical spectra. This effect is not as prevalent in the simulations because the threshold was numerically determined at each input angle  $\phi$ .

To prepare the experiment for a collision, we operated both lasers simultaneously, detuned in wavelength to allow for dispersion-induced walk-off, and adjusted the delay line in such a way that the collision occurred halfway down the fiber. We

**Computing  
with Solitons,**

**Fig. 21** Arbitrary-amplitude vector soliton propagation. (a) Wavelength shift versus angle to fast axis  $\phi$ , numerical curves given by dashed lines; (b, d) experimental results for  $\phi = 45^\circ$  and  $37^\circ$  with EDL2, respectively. Inset: input spectrum for EDL1; (c, e) corresponding numerical simulations of  $\phi = 45^\circ$  and  $37^\circ$ , respectively. The slow and fast axis components are depicted by solid and dashed lines, respectively (Reprinted with permission from Rand et al. (2007). Copyright by the American Physical Society)



define a collision length  $L_{\text{coll}} = 2T_{\text{FWHM}}/\Delta D\lambda$ , where  $\Delta\lambda$  is the wavelength separation between two vector solutions. For our setup,  $\Delta\lambda = 3$  nm, and  $L_{\text{coll}} = 83.3$  m. An asymptotic theory of soliton collisions, in which a full collision takes place, requires at least 5 collision lengths. The total fiber length in this experiment is equal to 6 collision lengths, long enough to ensure sufficient separation of solitons before and after collision. In this way, results of our experiments can be compared to the asymptotic theory, even though full numerical simulations will be shown for comparison. To quantify our results, we introduce a quantity  $R = \tan^2 \phi$ , defined as the amplitude ratio between the slow and fast components.

Recall that in section “[Manakov Solitons](#),” we introduced the Manakov equations (Eq. 4) and described collision-induced transformations of the polarization state of the soliton, which come about due to the asymptotic analysis of the soliton collision. The polarization state is the ratio between the two components  $\rho \equiv A_x/A_y = \cot \phi \exp(i\Delta\theta)$  and is therefore a function of the polarization angle  $\phi$  and the relative phase  $\Delta\theta$  between the two components. In the context of the experiments described in this section, these state transformations (Eqs. 5 and 7) predict that the resulting energy exchange will be a function of amplitude ratios  $R_{1,2}$ , wavelength separation  $\Delta\lambda$ , and the relative phase  $\Delta\theta_{1,2}$  between the two components of each soliton,

where soliton 1 (2) is the shorter (longer) wavelength soliton.

A word of caution is in order at this point. An interesting consequence of the 2/3 ratio between SPM and XPM, which sets the birefringent fiber model apart from the Manakov model, is the relative phase between the two components. For the Manakov soliton, each component “feels” the same amount of total nonlinearity, because the strengths of both SPM and XPM are equal. Therefore, regardless of the polarization angle, the amount of total nonlinear phase shift for each component is the same (even though the contributions of SPM and XPM phase shifts are in general not equal). As a result, the relative phase between the two components stays constant during propagation, as does the polarization state. This is *not* the case for vector solitons in birefringent fiber. For the case of equal amplitudes, each component does experience the same amount of nonlinear phase shift, and therefore the polarization state is constant as a function of propagation distance. However, for arbitrary (unequal) amplitudes, the total phase shift for each component will be different. Consequently, the relative phase will change *linearly* as a function of propagation distance, and the polarization state will not be constant. As a result, the collision-induced change in polarization state, while being a function of the amplitude ratios  $R_{1,2}$  and wavelength separation  $\Delta\lambda$ , will also depend upon the collision position due to the propagation dependence of the relative phase  $\Delta\theta_{1,2}(z)$ . To bypass this complication, we ensure that all collisions occur at the same spatial point in the fiber.

Because only one half-wave plate is used in our experiment (see Fig. 20), it was not possible to prepare each vector soliton individually with an arbitrary  $R$ . In addition due to the wavelength dependence of the half-wave plate, it was not possible to adjust  $\Delta\lambda$  without affecting  $R$ .

First, we investigated the phase dependence of the collision. This was done by changing the length of the HB-PMF entering the LB-PMF while keeping  $R$  and  $\Delta\lambda$  constant. As a result, we could change  $\Delta\theta_{1,2}$  due to the birefringence of the HB-PMF. Approximately 0.5 m of HB-PMF was added to ensure that the total amount of temporal

pulse splitting did not affect the vector soliton formation. The results are shown in Fig. 22, where Fig. 22a–f correspond to the short and long HB-PMFs, respectively. Figure 22a, d show the two vector solitons, which propagate independently when no collision occurs; expected, the two results are similar because the OSA measurement does not depend on  $\Delta\theta_{1,2}$ . The result of the collision is depicted in Fig. 22b, e, along with the corresponding simulation results in Fig. 22c, f.

In both of these collisions, an energy exchange between components occurs, and two important relations are satisfied: the total energy in each solution and in each component is conserved. It can be seen that when one component in a solution increased as a result of the collision, the other component decreases, with the opposite exchange in the second soliton. The difference between these two collisions is dramatic, in that the energy redistributes in opposite directions. For the simulations, idealized such pulses for each component were used as initial conditions, and propagation was modeled without accounting for losses. The experimental amplitude ratio was used and (without loss of generality (Jakubowski et al. 1998; Manakov 1973; Radhakrishnan et al. 1997))  $\Delta\theta_2$  was varied while  $\Delta\theta_1 = 0$ . Best fits gave  $\Delta\theta_2 = 90^\circ$  (Fig. 22c) and  $50^\circ$  (Fig. 22f). Despite the normal approximations, experimental and numerical results all agree to within 15% (Fig. 23).

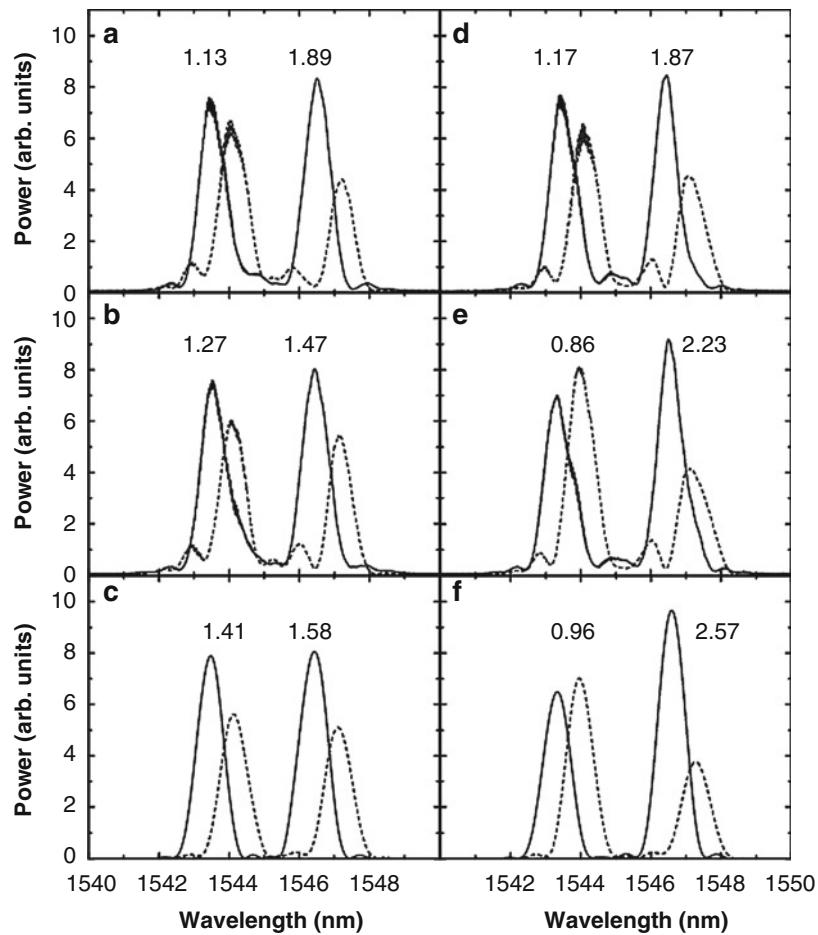
In the second set of results (Fig. 23), we changed  $R$  while keeping all other parameters constant. More specifically, we used the short HB-PMF, with initial phase difference  $\Delta\theta_2 = 90^\circ$  and changed the amplitude ratio. In agreement with theoretical predictions, the same direction of energy exchange is observed as in Fig. 22a–c.

## Spatial Soliton Collisions

We mention here analogous experiments with spatial solitons in photorefractive media by Anastassiou et al. In Anastassiou et al. (1999), it is shown that energy is transferred in a collision of vector spatial solitons in a way consistent with the predictions for the Manakov system (although

**Computing with Solitons,**

**Fig. 22** Demonstration of phase-dependent energy-exchanging collisions. (a–c) Short HB-PMF; (d–f) long HB-PMF; (a, d) experiment, without collision; (b, e) experiment with collision; (c, f) simulated collision result with c  $\Delta\theta_2 = 90^\circ$  and f  $\Delta\theta_2 = 50^\circ$ . Values of slow-fast amplitude ratio  $R$  are given above each soliton. The slow and fast axis components are depicted by solid and dashed lines, respectively (Reprinted with permission from Rand et al. (2007). Copyright by the American Physical Society)



the medium is a saturable one and only approximates the Kerr nonlinearity). The experiment in Anastassiou et al. (2001) goes one step farther, showing that one soliton can be used as an intermediary to transfer energy from a second soliton to a third. We thus are now at a point where the ability of both temporal and spatial vector solitons to process information for computation has been demonstrated.

## Future Directions

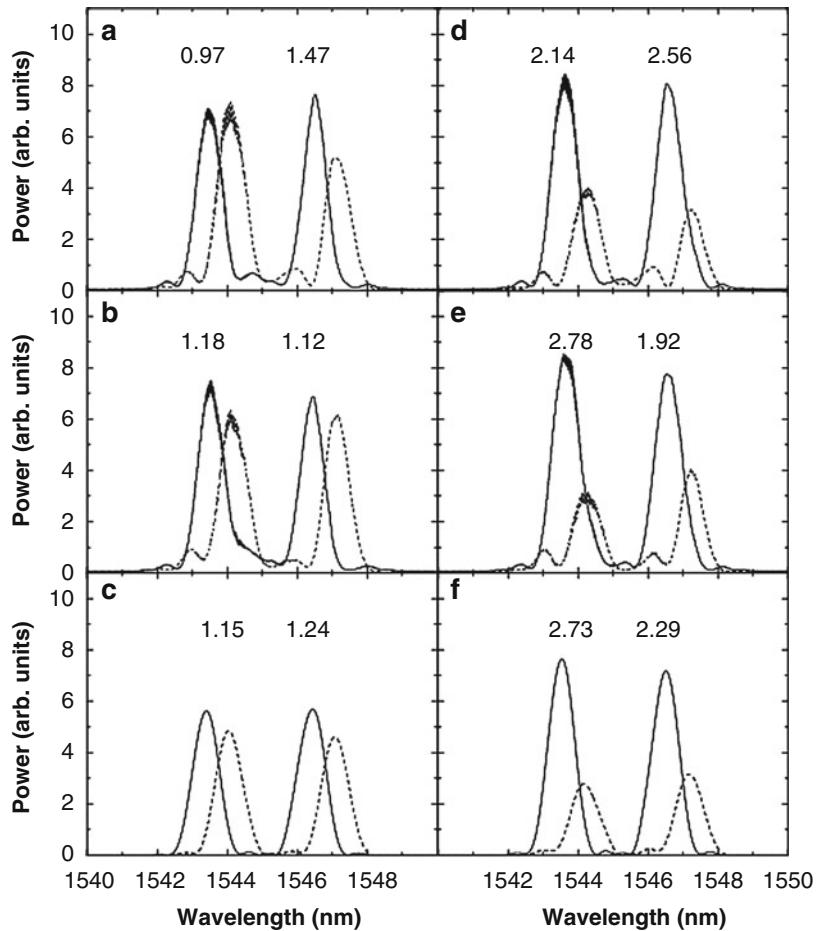
This article discussed computing with solitons and attempted to address the subject from basic physical principles to applications. Although the nonlinearity of fibers is very weak, the ultralow loss and tight modal confinement make them

technologically attractive. By no means, however, are they the only potential material for soliton-based information processing. Others include photorefractive crystals, semiconductor waveguides, quadratic media, and Bose-Einstein condensates, while future materials research may provide new candidate systems.

From a computing perspective, scalar soliton collisions are insufficient. Although measurable phase and position shifts do occur, these phenomena cannot be cascaded to affect future soliton collisions and therefore cannot transfer information from one collision to the next. Meaningful computation using soliton collisions requires a new degree of freedom, that is, a new component. Collisions of vector solitons display interesting energy-exchanging effects between components, which can be exploited for arbitrary computation and bistability.

**Computing  
with Solitons,**

**Fig. 23** Additional energy-exchanging collisions. (a, d) Experiment, without collision; (b, e) experiment, with Collision; (c, f) simulated collision results, using  $\Delta\theta_2 = 90^\circ$  inferred from the experiment of Fig. 22. Values of slow-fast amplitude ration  $R$  are given above each soliton. The slow and fast axis components are depicted by solid and dashed lines, respectively (Reprinted with permission from Rand et al. (2007))



The vector soliton experiments of section “[Experiments](#)” were proof-of-principle ones. The first follow-up experiments with temporal vector solitons in birefringent fiber can be directed toward a full characterization of the collision process. This can be done fairly simple by using the experimental setup of Fig. 20 updated in such a way as to allow independent control of two vector soliton inputs. This would involve separate polarizers and half-wave plates, followed by a polarization-preserving fiber coupler.

Cascaded collisions of temporal solitons also await experimental study. As demonstrated in photorefractive crystals with a saturable nonlinearity (Anastassiou et al. 2001), one can show that information can be passed from one collision to the next. Beyond a first demonstration of two collisions is the prospect

of setting up a multi-collision feedback cycle. Discussed in section “[Multistable Soliton Collision Cycles](#),” these collision cycles can be bistable and lead to interesting applications in computation.

Furthermore, the recent work of Ablowitz et al. (2006) shows theoretically that the useful energy redistribution properties of vector soliton collisions extend perfectly to the semi-discrete case, that is, to the case where space is discretized, but time remains continuous. This models, for example, propagation in an array of coupled nonlinear waveguides (Christodoulides and Joseph 1988). The work suggests alternative physical implementations for soliton switching or computing and also hints that the phenomenon of soliton information processing is a very general one.

## Bibliography

- Ablowitz MJ, Prinari B, Trubatch AD (2004) Soliton interactions in the vector nls equation. *Inverse Prob* 20(4):1217–1237
- Ablowitz MJ, Prinari B, Trubatch AD (2006) Discrete vector solitons: composite solitons, Yang-Baxter maps and computation. *Stud Appl Math* 116:97–133
- Agrawal GP (2001) Nonlinear fiber optics, 3rd edn. Academic, San Diego
- Anastassiou C, Segev M, Steiglitz K, Giordmaine JA, Mitchell M, Shih MF, Lan S, Martin J (1999) Energy-exchange interactions between colliding vector solitons. *Phys Rev Lett* 83(12):2332–2335
- Anastassiou C, Fleischer JW, Carmon T, Segev M, Steiglitz K (2001) Information transfer via cascaded collisions of vector solitons. *Opt Lett* 26(19):1498–1500
- Barad Y, Silberberg Y (1997) *Phys Rev Lett* 78:3290
- Berlekamp ER, Conway JH, Guy RK (1982) Winning ways for your mathematical plays, vol 2. Academic [Harcourt Brace Jovanovich Publishers], London
- Cao XD, McKinstrie CJ (1993) *J Opt Soc Am B* 10:1202
- Chen ZG, Segev M, Coskun TH, Christodoulides DN (1996) Observation of incoherently coupled photorefractive spatial soliton pairs. *Opt Lett* 21(18):1436–1438
- Christodoulides DN, Joseph RI (1988) Discrete self-focusing in nonlinear arrays of coupled waveguides. *Opt Lett* 13:794–796
- Christodoulides DN, Singh SR, Carvalho MI, Segev M (1996) Incoherently coupled soliton pairs in biased photorefractive crystals. *Appl Phys Lett* 68(13): 1763–1765
- Cundiff ST, Collings BC, Akhmediev NN, Soto-Crespo JM, Bergman K, Knox WH (1999) *Phys Rev Lett* 82:3988
- Fredkin E, Toffoli T (1982) Conservative logic. *Int J Theor Phys* 21(3/4):219–253
- Hasegawa A, Tappert F (1973) Transmission of stationary nonlinear optical pulses in dispersive dielectric fibers I: anomalous dispersion. *Appl Phys Lett* 23(3):142–144
- Islam MN, Poole CD, Gordon JP (1989) *Opt Lett* 14:1011
- Jakubowski MH, Steiglitz K, Squier R (1998) State transformations of colliding optical solitons and possible application to computation in bulk media. *Phys Rev E* 58(5):6752–6758
- Kang JU, Stegeman GI, Aitchison JS, Akhmediev N (1996) Observation of Manakov spatial solitons in AlGaAs planar waveguides. *Phys Rev Lett* 76(20): 3699–3702
- Korolev AE, Nazarov VN, Nolan DA, Truesdale CM (2005) *Opt Lett* 14:132
- Manakov SV (1973) On the theory of two-dimensional stationary self-focusing of electromagnetic waves. *Zh Eksp Teor Fiz* 65(2):505–516 [Sov. Phys. JETP 38, 248 (1974)]
- Mano MM (1972) Computer logic design. Prentice-Hall, Englewood Cliffs
- Menyuk CR (1987) *Opt Lett* 12:614
- Menyuk CR (1988) *J Opt Soc Am B* 5:392
- Menyuk CR (1989) Pulse propagation in an elliptically birefringent Kerr medium. *IEEE J Quantum Electron* 25(12):2674–2682
- Mollenauer LF, Stolen RH, Gordon JP (1980) Experimental observation of picosecond pulse narrowing and solitons in optical fibers. *Phys Rev Lett* 45(13): 1095–1098
- Nishizawa N, Goto T (2002) *Opt Express* 10:1151–1160
- Radhakrishnan R, Lakshmanan M, Hietarinta J (1997) Inelastic collision and switching of coupled bright solitons in optical fibers. *Phys Rev E* 56(2):2213–2216
- Rand D, Steiglitz K, Prucnal PR (2005) Signal standardization in collision-based soliton computing. *Int J Unconv Comput* 1:31–45
- Rand D, Glesk I, Bres CS, Nolan DA, Chen X, Koh J, Fleischer JW, Steiglitz K, Prucnal PR (2007) Observation of temporal vector soliton propagation and collision in birefringent fiber. *Phys Rev Lett* 98(5):053902
- Russell JS (1844) Report on waves. In: Report of the 14th meeting of the British Association for the Advancement of Science. Taylor and Francis, London, pp 331–390
- Shih MF, Segev M (1996) Incoherent collisions between two-dimensional bright steady-state photorefractive spatial screening solitons. *Opt Lett* 21(19):1538–1540
- Shor PW (1994) Algorithms for quantum computation: discrete logarithms and factoring. In: 35th IEEE Press, Piscataway, pp 124–134
- Squier RK, Steiglitz K (1993) 2-d FHP lattice gasses are computation universal. *Complex Syst* 7:297–307
- Stebliina VV, Buryak AV, Sammut RA, Zhou DY, Segev M, Prucnal P (2000) Stable self-guided propagation of two optical harmonics coupled by a microwave or a terahertz wave. *J Opt Soc Am B* 17(12): 2026–2031
- Steiglitz K (2000) Time-gated Manakov spatial solitons are computationally universal. *Phys Rev E* 63(1): 016608
- Steiglitz K (2001) Multistable collision cycles of Manakov spatial solitons. *Phys Rev E* 63(4):046607
- Yang J (1997) *Physica D* 108:92–112
- Yang J (1999) Multisoliton perturbation theory for the Manakov equations and its applications to nonlinear optics. *Phys Rev E* 59(2):2393–2405
- Zakharov VE, Shabat AB (1971) Exact theory of two-dimensional self-focusing and one-dimensional self-modulation of waves in nonlinear media. *Zh Eksp Teor Fiz* 61(1):118–134 [Sov. Phys. JETP 34, 62 (1972)]



## Reaction-Diffusion Computing

Andrew Adamatzky<sup>1</sup> and Benjamin De Lacy Costello<sup>2</sup>

<sup>1</sup>Unconventional Computing Centre, University of the West of England, Bristol, UK

<sup>2</sup>Centre for Analytical Chemistry and Smart Materials, University of the West of England, Bristol, UK

### Article Outline

Glossary

Introduction

Classification of Reaction-Diffusion Processors

Recipes of the Chemical Processors

Specialized Processors

Universal Processors

Arithmetic Circuits

Future Directions

Bibliography

### Glossary

**Belousov–Zhabotinsky (BZ) reaction** is a term applied to a group of chemical reactions in which an organic substrate (typically malonic acid) is oxidized by bromate ions in the presence of acid and a one electron transfer redox catalyst (e.g., ferroin, or the light-sensitive ruthenium complex). During the BZ reaction, there are three major interlinked processes – firstly the reduction of the inhibitor (bromide ions) via reaction with bromate ions, secondly autocatalysis in bromous acid and the oxidation of the redox catalyst, and finally reduction of the redox catalyst and production of the inhibitor (bromide ions) via a reaction with the organic substrate and its brominated derivative. The reaction produces oscillations in well-stirred reactors and traveling waves in thin layers, which may be

visualized if the redox behavior of the catalyst is accompanied by a change of color (e.g., the color is changed from orange to blue when ferroin is oxidized to ferriin).

**Excitable medium** is spatially distributed assembly of coupled **excitable systems**; spatial distribution and coupling allow for propagation of excitation waves. **Excitable system** is a system with a single steady quiescent state that is stable to small perturbations, but responds with an excursion from its quiescent state (excitation event) if the perturbation is above a critical threshold level. After excitation, the system enters a refractory period during which time it is insensitive to further excitation before returning to its steady state.

**Light-sensitive Belousov–Zhabotinsky reaction** is a **Belousov– Zhabotinsky reaction** where the ruthenium catalyst is excited by 460 nm light (blue) and reacts with bromomalonic acid to produce bromine, an inhibitor of autocatalysis. By varying the light levels, the excitability of the system can be controlled.

**Logical gate** is an elementary building block of a digital, or logical, circuit, which represents (mostly) binary logical operations, e.g., AND, OR, XOR, with two input terminals and one output terminal. In Boolean logic, terminals are in one of two binary conditions (e.g., low voltage and high voltage) corresponding to TRUE and FALSE values of logical variables.

**Logically universal processor** is a system which can realize a functionally complete set of logical operations in its development, e.g., conjunction and negation. **Oregonator** is a system of three (or two in a modified version) coupled differential equations aimed to simulate oscillatory phenomena in the **Belousov–Zhabotinsky reaction**.

**Palladium processor** is a **reaction-diffusion** chemical medium, where the substrate includes palladium chloride, and data are represented by drops of potassium iodide, which forms a colored precipitate of iodo-palladium species when it reacts with palladium chloride. Interaction of

chemical waves in the medium leads to the formation of uncolored domains, which – depending on particulars of the problem – represent the result of the computation (e.g., **Voronoi diagram**).

**Precipitation reaction** is a reaction where soluble components mixed together to form an insoluble compound that drops out of solution as a solid (called a precipitate). **Reaction-diffusion processor** is a thin layer of a reagent mixture which reacts to changes of one reagent's concentration – data configuration – in a predictable way to form a stationary pattern corresponding to the concentration of the reagent – result configuration. A computation in the chemical processor is implemented via the spreading and interaction of diffusive or phase waves.

**Reagent** is a substance used in a chemical reaction.

**Shortest-path problem** is the problem of finding a path between two sites (e.g., vertices of the graph, locations in the space) such that the length (sum of the weights of the graph edges or travel distances) of the path is minimized.

**Skeleton** of a planar contour is a set of centers of bitangent circles lying inside the contour.

**Subexcitable medium** is a medium whose steady state lies between the excitable and unexcitable domains. In excitable media, waves initiated by perturbations of a sufficient size propagate throughout the media. In an unexcitable medium, no perturbation is large enough to trigger a wave. In a subexcitable medium, **wave fragments** with open ends are formed.

**Substrate** is any stratum lying underneath another; in chemical processors, it is commonly the substance impregnated into a gel layer and acted upon by diffusing species.

**Voronoi diagram** (also known as Dirichlet tessellation) of a planar set  $P$  of planar points is a partition of the plane into regions, each for any element of  $P$ , such that a region corresponding to a unique point  $p$  contains all those points of the plane that are closer to  $p$  than to any other node of  $P$ .

**Wave fragment** is an excitation wave formed in a **subexcitable medium**; this is a segment with free ends, which either expand or contract, depending on their size and the medium's

excitability. In a subexcitable medium with lower excitability, waves with free ends contract and eventually disappear.

## Introduction

First ever paper on reaction-diffusion computing was published by Kuhnert and Agladze (Kuhnert 1986a, b; Kuhnert et al. 1989). They demonstrated that a spatially extended excitatable light-sensitive chemical system – Belousov-Zhabotinsky (BZ) reaction (Zaikin and Zhabotinsky 1970) is capable of implementing memory and basic image processing functions. These first BZ processors did not employ the propagation of excitation waves but relied on global switching of the medium between excited and no excited states termed phase shifts. In a light-sensitive analog of the BZ reaction, the timing of these phase shifts could be coupled to the projected light intensity due to the light sensitivity of the catalyst used in the reaction. However, the publications encouraged other researchers to experiment with information processing in excitable chemical systems. The image processing capabilities of BZ systems were further explored and the research results were enhanced in (Agladze et al. 1995; Rambidi 1997, 1998, 2003; Rambidi and Yakovenchuk 2001; Rambidi et al. 2002).

In the mid-nineties, the first results concerning the directed, one-way, propagation of excitation waves in geometrically constrained chemical media was announced (Agladze et al. 1996). This led to a series of experimental studies concerning the construction of Boolean (Tóth and Showalter 1995; Steinbock et al. 1996; Motoike and Yoshikawa 2003) and three-valued (Motoike and Adamatzky 2004) logical gates and eventually led to the construction of more advanced circuits (Gorecka and Gorecki 2003; Gorecki et al. 2003; Ichino et al. 2003; Motoike and Yoshikawa 1999) and dynamical memory (Motoike et al. 2001) in the BZ reaction. The BZ medium was also used for one optimization task – shortest-path computation (Agladze et al. 1997; Rambidi and Yakovenchuk 2001; Steinbock et al. 1995).

The studies in reaction-diffusion computing were boosted by algorithms of spatial

computation in cellular automata, when the first automaton model for Voronoi diagram construction was designed (Adamatzky 1994, 1996). The algorithm was subsequently implemented under experimental laboratory conditions using a precipitating chemical processor (Tolmachiev and Adamatzky 1996; Adamatzky and Tolmachiev 1997), and later a variety of precipitating chemical systems were discovered to be capable of Voronoi diagram approximation (De Lacy Costello 2003; De Lacy Costello and Adamatzky 2003; de Lacy Costello et al. 2004a, b). The precipitating systems also proved to be efficient in the realization of basic logical gates, including many-valued gates (Adamatzky and De Lacy Costello 2002a).

In the early 2000s, the first ever excitable chemical controller mounted on-board a wheeled robot was constructed and tested under experimental laboratory conditions (Adamatzky and De Lacy Costello 2002c; Adamatzky et al. 2003, 2004), and also a robotic hand was interfaced and controlled using a Belousov-Zhabotinsky medium (Yokoi et al. 2004). These preliminary experiments opened the door to the future construction of embedded robotic controllers and other intelligent reaction-diffusion processors.

As for computational universality, sub-excitable analogs of chemical media have been proved by experiment to be powerful collision-based based computers, capable of the implementation of universal and arithmetical logical circuits via the collision of propagating wave-fragments (Adamatzky 2004; Adamatzky and De Lacy Costello 2007; Tóth et al. 2009).

## Classification of Reaction-Diffusion Processors

In reaction-diffusion processors, both the data and the results of the computation are encoded as concentration profiles of the reagents. The computation per se is performed via the spreading and interaction of wavefronts.

This can be summarized using the following relationships between architecture or operation and implementation:

- Component base, computing substrate → Thin layer of reagents
- Data representation → Initial concentration profile
- Information transfer, communication → Diffusive and phase waves
- Computation → Interaction of waves
- Result representation → Final concentration profile.

Reaction-diffusion computers are parallel because the chemical medium's microvolumes update their states simultaneously, and molecules diffuse and react in parallel. Architecture and operations of reaction-diffusion computers are based on three principles (Adamatzky 2001; Margolus 1984):

- Computation is dynamical: physical action measures the amount of information.
- Computation is local: physical information travels only a finite distance.
- Computation is spatial: the nature is governed by waves and spreading patterns.

Potentially, a reaction-diffusion computer is a supercomputer in a goo. Liquid-phase chemical computing media are characterized by:

- Massive parallelism: millions of elementary – 2–4 bit – processors in a small chemical reactor.
- Local connectivity: every microvolume of the medium changes its state depending on the states of its closest neighbors.
- Parallel I/O: optical input – control of initial excitation dynamics by illumination masks, output is parallel because concentration profile representing results of computation is visualized by indicators.
- Fault tolerance and automatic reconfiguration: because if we remove some quantity of the liquid phase, the topology is restored almost immediately.

As far as reusability is concerned, there are two main classes of reaction-diffusion processors: reusable and disposable. Excitable chemical media are one of a family of reusable processors because given

an unlimited supply of reagents, unlimited lifetime of the catalyst and the removal of any by-products the system can be excited many times and at many simultaneous points on the computing substrate. The excitation waves leave almost no trace a certain time after the propagation. Precipitating chemical systems, on the other hand, are disposable processors. Once a locus of the substrate is converted to precipitate, it stays in the precipitate state indefinitely and therefore this processor cannot be used again within a realistic timeframe.

With regard to the communication between domains of the computing medium, the chemical media can be classified into either broadcasting or peer-to-peer architectures. Excitable chemical media and most precipitating systems are broadcasting. When a single site of a medium is perturbed the perturbation spreads in all directions, as a classical circular wave. Thus each perturbed site broadcasts. However, when diffusion or excitation in the medium are limited, e.g., the subexcitable Belousov-Zhabotinsky system, no circular waves are formed but instead self-localized propagating wave fragments are formed (chemical relatives of dissipative solitons). These wave-fragments propagate in a predetermined direction, therefore they can be seen as data packets transferred between any two points of the computing medium.

Broadcasting chemical processors are good for image processing tasks, some kinds of robot navigation and path computation. They are appropriate for all tasks when massive spatially extended data must be analyzed, modified, and processed. The circular wave based communication becomes less handy when one tries to build logical circuits: it is preferable to route the signal directly between two gates without affecting or disturbing nearby gates.

When one uses an excitable medium to implement a logical circuit, the propagation of the excitable waves must be restricted, e.g., by making conductive channels surrounded by nonconductive barriers, or via the use of localized excitations in a subexcitable chemical medium. Reaction-diffusion processors where the computing space is geometrically inhomogeneous are called geometrically constrained processors. Subexcitable chemical media, where wave-fragments propagate freely

are called collision-based (because the computation is implemented when soliton-like wave-fragments collide) or architecture-less/free-space computers (because the medium is homogeneous, regular, and uniform).

Further, we provide examples of architecture-less chemical processors acting in either broadcasting or peer-to-peer modes.

## Recipes of the Chemical Processors

In the chapter, we refer to only three kinds of chemical reaction-diffusion processors: the precipitating palladium processor, and two chemical processors: an excitable Belousov-Zhabotinsky (BZ) medium and a light-sensitive subexcitable analog of the BZ reaction. Below we show how to prepare the reaction-diffusion processors in laboratory conditions.

### Palladium Processor

A gel of agar (1.5% by weight, agar select Sigma-Aldrich Company Ltd. Poole, Dorset, BH12 4XA) containing palladium chloride (Palladium (II) chloride 99%, Sigma Aldrich Company Ltd.) in the range 0.2–0.9% by weight (0.011–0.051 M) is prepared by mixing the solids in warm deionized water. The mixture is heated with a naked flame and constant stirring until it boils to ensure full dissolution of the palladium chloride and production of a uniform gel (on cooling). The boiling liquid is then transferred to Petri dishes or alternatively spread on acetate sheets to a thickness of 1–2 mm and left to set. We favor the use of Petri dishes because the reaction process is relatively slow and drying of the gel can occur if kept open to the atmosphere. The non-reacted gel processors are then kept for 30 min although they remained stable for in excess of 24 h provided drying was controlled. A saturated solution (at 20 °C) of potassium iodide (ACS reagent grade, Aldrich Chemical Co.) is used as the outer electrolyte for the reactions. Drops of outer electrolyte are applied to the surface of the gel to initiate the reaction process.

### BZ Processor

A thin layer Belousov-Zhabotinsky (BZ) medium is usually prepared using a recipe adapted from (Field and Winfree 1979): an acidic bromate stock solution incorporating potassium bromate and sulphuric acid ( $[\text{BrO}_3^-] = 0.5 \text{ M}$  and  $[\text{H}^+] = 0.59 \text{ M}$ ) (Solution A); solution of malonic acid (solution B) ( $[\text{CH}_2(\text{CO}_2\text{H})_2] = 0.5 \text{ M}$ ), and sodium bromide (solution C) ( $[\text{Br}^-] = 0.97 \text{ M}$ ). Ferroin (1,10-phenanthroline iron-II sulphate,  $0.025 \text{ M}$ ) is used as a catalyst and a visual indicator of the excitation activity in the BZ medium. To prepare a thin layer of the BZ medium, we mix solutions A (7 mL), B (3.5 mL), and C (1.2 mL); finally, when the solution becomes colorless, ferroin (1 mL) is added and the mixture is transferred to a Petri dish (layer thickness 1 mm). Excitation waves in the BZ reaction are initiated using a silver colloid solution.

### Light-Sensitive Sub-Excitable BZ Processor

The recipe is quoted from Tóth et al. (2009). A light-sensitive BZ processor consists of a gel impregnated with catalyst and a catalyst-free solution pumped around the gel. The gel is produced using a sodium silicate solution prepared by mixing 222 mL of the sodium silicate solution with 57 mL of 2 M sulfuric acid and 187 mL of deionized water (Tóth et al. 2009). The catalyst  $\text{Ru}(\text{bpy})_3\text{SO}_4$  is recrystallized from the chloride salt with sulfuric acid. Precured solutions for making gels are prepared by mixing 2.5 mL of the acidified silicate solution with 0.6 mL of 0.025 M  $\text{Ru}(\text{bpy})_3\text{SO}_4$  and 0.65 mL of 1.0 M sulfuric acid solution. Using capillary action, portions of this solution were quickly transferred into a custom-designed 25 cm long, 0.3 mm deep Perspex mould covered with microscope slides. The solutions are left for 3 h in the mould to permit complete gelation. After gelation, the adherence to the Perspex mould is negligible leaving a thin gel layer on the glass slide. After 3 h, the slides with the gel on them are carefully removed from the mould and placed into 0.2 M sulphuric acid solution for an hour. Then they are washed in deionized water at least five times to remove by-products. The gels are  $26 \times 26 \text{ mm}$ , with a wet thickness of

approximately 300 m. The gels are stored under water and rinsed just before use.

The catalyst-free reaction mixture is freshly prepared in a 30 mL continuously fed stirred tank reactor, which involves the *in situ* synthesis of stoichiometric bromomalonic acid from malonic acid and bromine generated from the partial reduction of sodium bromate. This reactor is continuously fed with fresh catalyst-free BZ solution in order to maintain a nonequilibrium state. The final composition of the catalyst-free reaction solution in the reactor is following 0.42 M sodium bromate, 0.19 M malonic acid, 0.64 M sulphuric acid, and 0.11 M bromide.

A light projector is used to illuminate the computer-controlled image. Images are captured using a digital camera. The open reactor is surrounded by a water jacket thermostated at  $22^\circ\text{C}$ . Peristaltic pumps are used to pump the reaction solution into the reactor and remove the effluent.

In some case, we applied the regular structure of illumination onto the BZ chemical reactor. Four light levels can be used: black (zero light level), at which level the reaction BZ oscillates; dark ( $0.035 \text{ mW cm}^{-2}$ ), which represents the excitable medium in which a chemical wave is able to propagate; white (maximum light intensity:  $3.5 \text{ mW cm}^{-2}$ ), the inhibitory level; and finally the light level ( $1.35 \text{ mW cm}^{-2}$ ) corresponding to the weakly excitable medium, where excitation just manages to propagate.

Waves were initiated by setting the light intensity to zero within a small square at specific points on the gel surface. We use a black oscillating square to initiate waves periodically. The waves are then directed from source into a weakly excitable area (controlled by projecting a light intensity of  $1.35 \text{ mW cm}^{-2}$ ) such that only small fragments are able to propagate.

### Specialized Processors

As their name states specialized processors are designed to solve just one possible computational task (or family of very similar tasks). In the section, we provide examples of specialized

processors for computational geometry, image processing, and robotics.

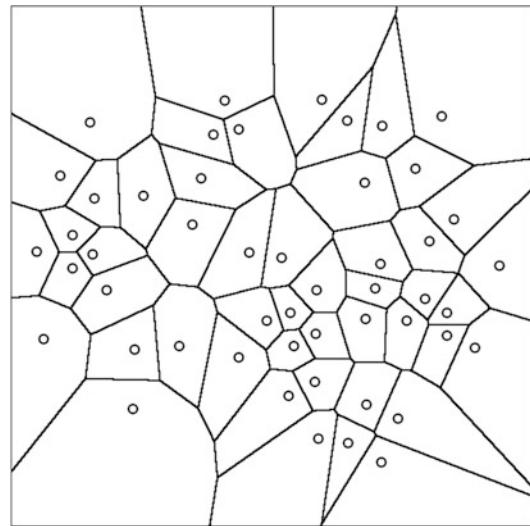
### Voronoi Diagram and Skeletonization

Let  $\mathbf{P}$  be a nonempty finite set of planar points. A planar Voronoi diagram of the set  $\mathbf{P}$  is a partition of the plane into such regions, that for any element of  $\mathbf{P}$ , a region corresponding to a unique point  $p$  contains all those points of the plane which are closer to  $p$  than to any other node of  $\mathbf{P}$ . A unique region  $\text{vor}(p) = \{z \in \mathbf{R}^2 : d(p, z) < d(p, m) \forall m \in \mathbf{R}^2, m \neq z\}$  assigned to point  $p$  is called a Voronoi cell of the point  $p$ . The boundary of the Voronoi cell of a point  $p$  is built of segments of bisectors separating pairs of geographically closest points of the given planar set  $\mathbf{P}$ . A union of all boundaries of the Voronoi cells determines the *planar Voronoi diagram*:  $VD(\mathbf{P}) = \bigcup_{p \in \mathbf{P}} \partial_{\text{vor}}(p)$ . A variety of Voronoi diagrams and algorithms of their construction can be found in (Klein 1990). An example of a Voronoi diagram is shown in Fig. 1.

Voronoi cells of a planar set represent the natural or geographical neighborhood of the set's elements. Therefore, the computation of a Voronoi diagram based on the spreading of some “substance” from the data points is usually the first approach of those trying to design massively parallel algorithms in chemical nonlinear systems.

The “spreading substance” can be represented by a potential or an oscillatory field, or diffusing or phase waves, or simply by some traveling inhomogeneities or disturbances in the physical computing medium. This idea was first explored by Blum, Calabi, and Hartnett in their “grass-fire” transformation algorithm (Blum 1967, 1973; Calabi and Hartnett 1968). A fire is started at planar points of given data set, the fire spreads from the data points on a substrate (that constitutes the computing medium) and the sites where the fire fronts meet represent the edges of the Voronoi diagram of the planar data set.

Quite similarly, to construct the diagram with a potential field one puts the field generators at the data points and then detects sites where two or more fronts of the field waves meet. This technique is employed in the computation of a Voronoi diagram via repulsive potential and oscillatory fields in homogeneous neural networks (Hwang and



**Reaction-Diffusion Computing, Fig. 1** Voronoi diagram of 50 planar points randomly distributed in the unit disc

Ahuja 1992; Lemmon 1991). These approaches are “natural” and intuitive and their simplicity of implementation made them an invaluable tool for massively parallel image processing. The weakness of the approach is that a stationary structure is formed as a result of the computation and the meeting points of the colliding wavefronts must be detected by some “artificial” or extrinsic methods. This disadvantage is eliminated in the reaction-diffusion technique as we demonstrate.

To compute a Voronoi diagram of planar points, we represent each point by a unique drop of outer electrolyte. Configuration of the outer electrolyte drops corresponds to the configuration of data to be subdivided by the Voronoi diagram. We represent data objects (to be separated) using a clear solution of potassium iodide. The potassium iodide diffuses from the sites of the drops or from the edges of planar shapes into the palladium chloride loaded gel. The potassium iodide reacts with the palladium chloride to form iodo-palladium species. Palladium chloride gel is light yellow colored while iodo-palladium species are dark brown. Thus, during the process of computation we observe the growth of dark-brown patterns emanating from the initial data sites. At sites where two or more diffusive fronts meet each other, almost no precipitate is

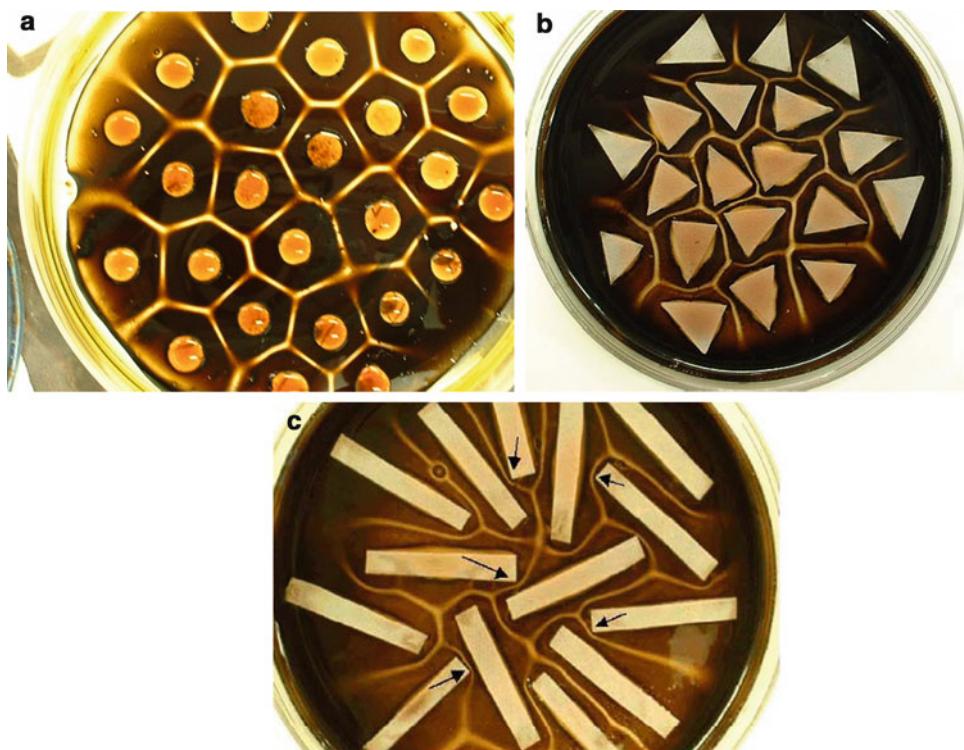
formed; these sites, therefore, remain uncolored, and they represent the bisectors of a Voronoi diagram, generated by the initial geometrical configuration of data objects (Fig. 2a).

Voronoi diagrams of other geometric shapes can be constructed by substituting the drops of outer electrolyte for pieces of absorbent materials soaked in the electrolyte solution and applying these to the gel surface (Fig. 2b, c).

The processor starts its computation as soon as drops/shapes with potassium iodide are applied; the computation is finished when no more precipitate is formed. Configurations of drops/filter-paper templates of potassium iodide correspond to the input states of the processor; the resulting two-dimensional concentration profile of the precipitate (iodo-palladium species) is the output state of the processor. The processor can be seen as a preprogrammed or hard-wired device because the way in which the chemical species react cannot be changed.

It should be noted that the “palladium processor” is just one of a huge number of chemical-based processors capable of Voronoi diagram calculation. Some others particularly those based on gels containing potassium ferrocyanide and mixed with various metal salts are discussed in (Adamatzky et al. 2005). Reagents with differing diffusion on the same substrate and differing chemical reactivities to the gel substrate can be used to form a number of generalized weighted Voronoi diagrams.

It is also interesting to note that in these chemical systems – the more data that is physically inputted in a given area – the faster the computational outcome. The limit of the processors is currently determined by the gel thickness and integrity and the ability to accurately input data points – but these are just design issues which can realistically be overcome and do not limit the theoretical capabilities of the systems. Also these systems can be practically used to implement three-dimensional



**Reaction-Diffusion Computing, Fig. 2** Voronoi diagram of planar points/discs (a), triangles (b), and rectangles (c) computed in the palladium processor (From Adamatzky et al. 2005)

Voronoi tessellations. For example, algorithms of classical computation of Voronoi diagram of planar points and planar objects are very different because when arbitrary shaped objects are involved one need to employ algebraic curves to produce bisectors, which does increase complexity. In chemical processors, it does not matter what shape data objects have, time of computation is determined only by maximum distance between two geographically neighboring objects.

Skeletonization (Fig. 3) of the planar shapes in a reaction-diffusion chemical processor is yet further proof of the correctness of our proposed mechanism, namely the wave-velocity-dependent bisector formation. During skeletonization, there are no interactions of wavefronts emanating from different sources; there is only one front, generated at the edges of the closed planar shape. Concave parts of the shape produce bisectors; however, the further the bisecting sites are from the given shape the larger the width of the bisectors and the more precipitate they contain (this is

because the velocity of the fronts corresponding to the concave parts reduces with time) (Adamatzky and De Lacy Costello 2002b).

### Collision-Free Path Calculation

A BZ parallel processor (Adamatzky and De Lacy Costello 2002c) which computes a collision-free path solves the following problem. Given a space with a set of obstacles and two selected sites, find the shortest path between the source and the destination such that every site of the path is as far from the obstacles as possible. This is not only a classical problem in mathematical optimization but the one heavily relied on in robotics, logistics, electronic design, etc.

The problem has already been tackled in a framework of wave-based computing. Three experimental prototypes exist, where BZ processors have been designed to compute shortest paths, these are as follows: (a) extraction of an optimal path in a labyrinth from excitation wavefront dynamics in the labyrinth (Steinbock

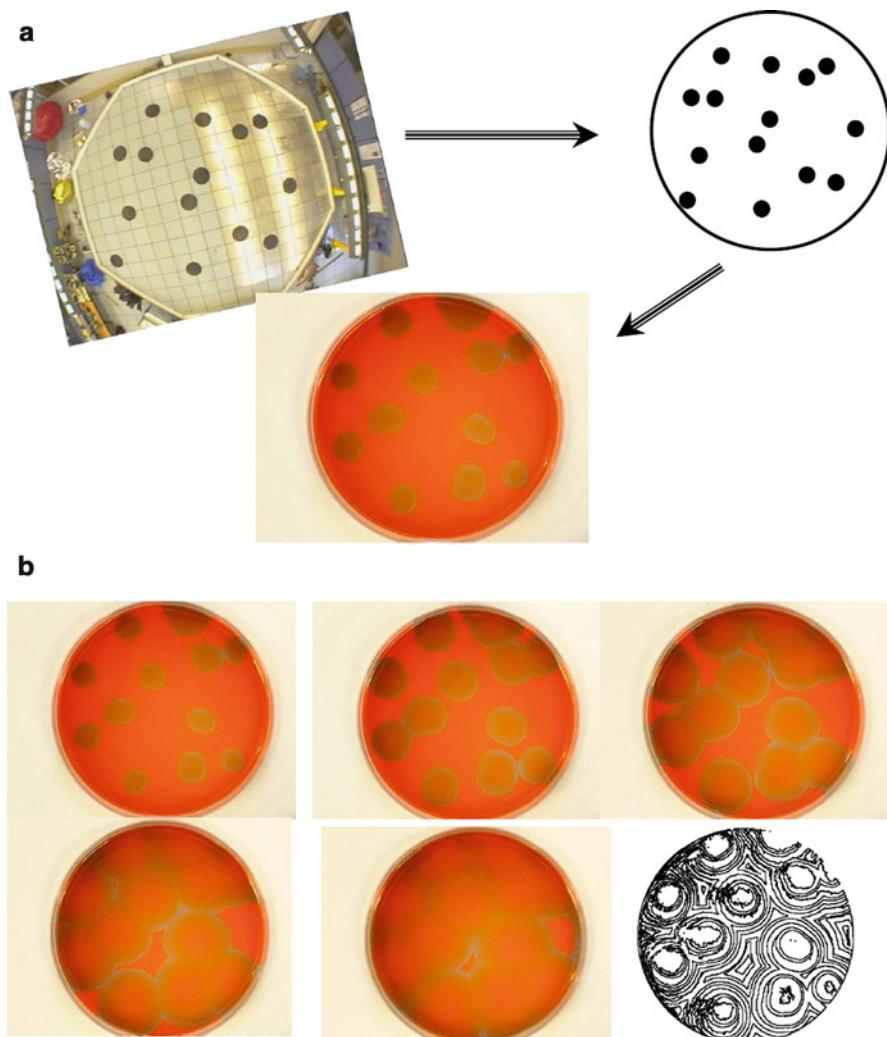


**Reaction-Diffusion Computing, Fig. 3** Computation of the skeleton of a planar shape in the palladium processor (From Adamatzky and De Lacy Costello (2002b))

et al. 1995); the path is extracted from time-lapsed snapshots of the wavefront motion; (b) collision-free path using the BZ medium, where obstacles are represented by drops of KCl or strong illumination (Agladze et al. 1997); a path is extracted from the motion of excitation waves; (c) approximation of a path on a tree in a light-sensitive BZ-medium (Rambidi and Yakovenchuk 2001); this technique is rather complicated because a data-tree is represented by gradients of the medium and image processing routines were implemented at every step of the medium's development.

We have designed a BZ processor coupled with a two-dimensional cellular automaton, the hybrid system computes the shortest path by first approximating a distance field generated by the obstacles (BZ medium) and then calculating the shortest path in the field (cellular automaton) (Adamatzky et al. 2003).

A configuration of obstacles (Fig. 4a) is represented by an identical configuration of silver wires parallel to each other and perpendicular to the surface of the BZ medium. Each circular obstacle is represented by a unique wire



**Reaction-Diffusion Computing, Fig. 4** Converting obstacles to excitations: (a) mapping experimental arena (30 m in diameter) to Belousov-Zhabotinsky medium

(9 cm in diameter), (b) approximation of a distance field (From Adamatzky et al. (2003))

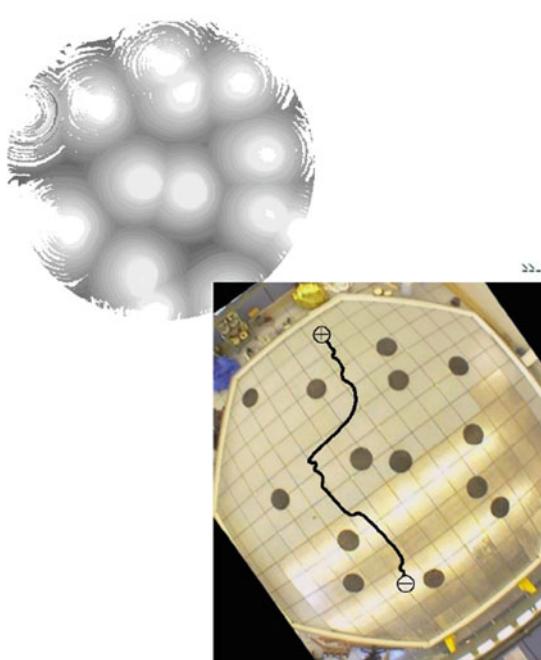
positioned exactly at the center of the obstacle. To start the computation in the BZ processor, we briefly immerse the tips of all the wires into the BZ mixture (in the reduced steady state). Immersing silver wires at specific points of the BZ mixture reversibly removes bromide ions from these sites. Bromide ions act as primary inhibitors of the autocatalytic reaction in the BZ medium, therefore local removal of bromide ions stimulates wave generation. The digital images of BZ medium are taken at certain intervals until all primary wavefronts had been annihilated (Fig. 4b). Then we transform a series of color snapshots taken to a grey-level matrix which represents a distance scalar field derived from a configuration of obstacles (Fig. 5, on the left).

The distance matrix is mapped to a two-dimensional excitable cellular automaton, which calculates a shortest path between any two points of the experimental arena, labeling the path with the local pointers (Fig. 5, on the right). The configuration of local pointers, representing the shortest collision-free path, can be communicated to a mobile robot which navigates the experimental arena (Fig. 5, center).

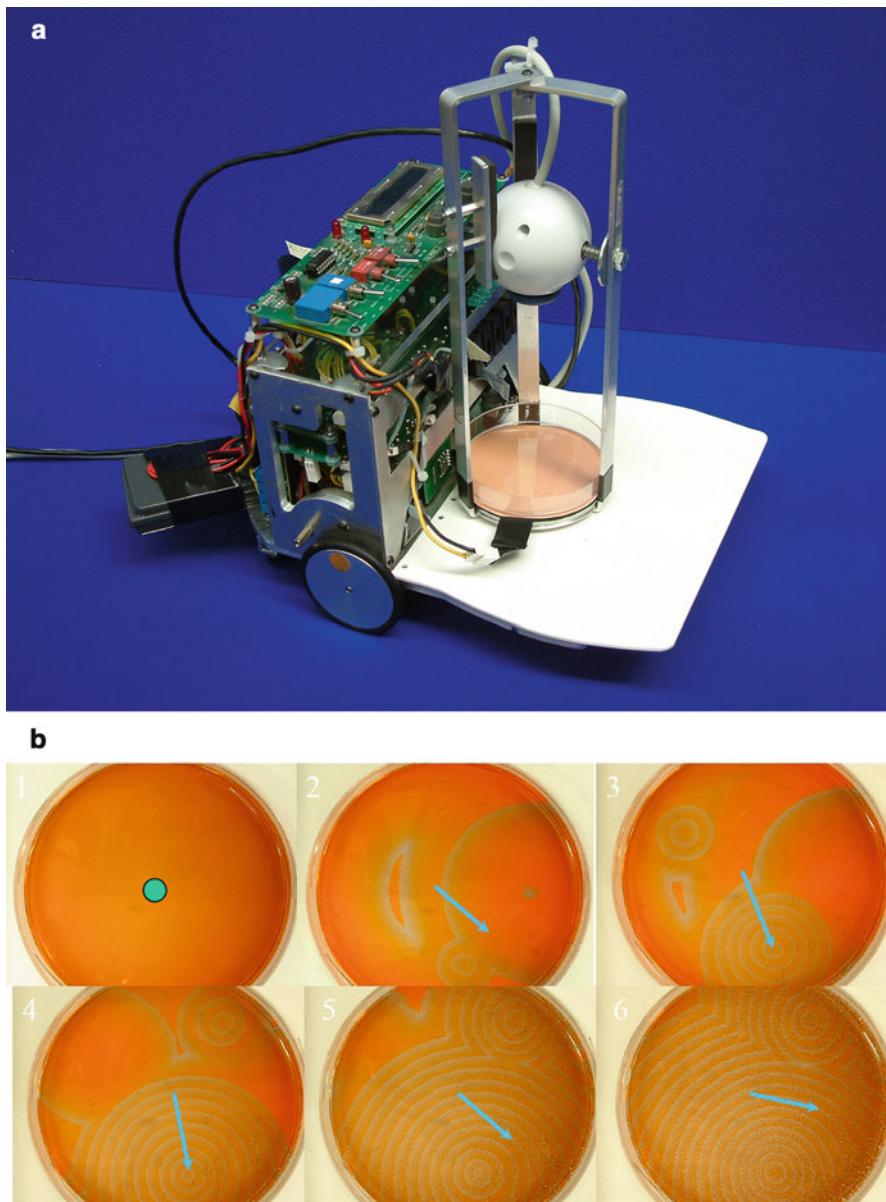
## Taxis

The Belousov-Zhabotinsky reaction is capable of assisting in the navigation of a mobile robot in a real-life environment (Adamatzky et al. 2004). Given an on-board thin-layer chemical reactor with the liquid-phase Belousov-Zhabotinsky medium, one can apply by intermittent stimulation of the medium to sensibly guide the robot.

To make a BZ robotic taxis controller we prepare a thin layer Belousov-Zhabotinsky (BZ) reaction using a recipe adapted from (Field and Winfree 1979). Excitation waves in the BZ reaction are initiated using a silver colloid solution, which is added to the medium by a human operator. The chemical controller is placed on-board a wheeled mobile robot (Fig. 6a). The robot is about 23 cm in diameter and able to turn on the spot; wheel motors are controlled by Motorola 68,332 on-board processor. The robot features a horizontal platform, where the Petri dish (9 cm in diameter) is fixed, and a stand with a digital camera Logitech QuickCam (in  $120 \times 160$  pixels resolution mode), to record excitation dynamics. Robot controller and camera are connected to a PC via serial port RS-232 and USB 1.0, respectively.



**Reaction-Diffusion Computing, Fig. 5** Extraction of the shortest collision-free path (From Adamatzky et al. (2005))



**Reaction-Diffusion Computing, Fig. 6** Wheeled robot controlled by BZ medium (a) and snapshots of excitation dynamics with robot's velocity vector extracted (b) (From Adamatzky et al. (2004))

Because vibrations affect processes in BZ reaction systems we make the movements of the robot as smooth as possible; thus in our experiments, the robot moves with a speed of circa 1 cm/s. We found that rotational movements were particularly disruptive and caused spreading of the excitation waves from the site of the existing wavefronts faster than would be seen if the process was just

diffusion. Therefore, to minimize this effect we made the robot rotate with a very low speed of around 1°/s.

To enhance images of the excitation dynamics, the Petri dish is illuminated from underneath by a flexible electroluminescent sheet (0.7 mm thick) cut to the shape of the dish. The sheet, powered through an inverter from the robot's batteries,

produces a cool uniform blue light not affecting (at least in conditions of our experimental set-up) physicochemical processes in the reactor. We found it is preferable to implement experiments in dark areas to avoid interference of daylight and lamps with the light from the luminescent sheet.

When we excite the medium with a silver wire a local generator of target waves is formed (Fig. 6b). The robot extracts the position of the stimulation point, relative to the center of the chemical reactor, from the topology of the excitation target waves. The robot adjusts its velocity vector to match the vector toward the source of stimulation. To guide the robot more precisely one can excite the chemical medium in several points (Fig. 6b). It should be noted that eventually it was intended for environmental interaction to become the source of stimulation for excitation waves. The forced stimulation was simply to test algorithms on board with real chemical controllers and to learn about problems with coupling chemical reactors and robots. Something which was identified as a problem – i.e., physical disruption of chemical waves could actually turn to be advantageous as it may feed back both direction, velocity, and even rotational movement of robot.

### Control of Robotic Hand

In previous sections, we described the designs of chemical controllers for robots which can calculate a shortest collision-free path in the robotic arena and guide the robot toward the source of stimulation (taxis). However, the controllers described lacked a feedback from the robot. In a set of remarkable experiments undertaken by Hiroshi Yokoi and Ben De Lacy Costello (Yokoi et al. 2004), it was demonstrated that when a closed-loop interface between the chemical controller and the robot is established the behavior of the hybrid systems becomes even more intriguing.

In the chemical controller of the robotic hand (Yokoi et al. 2004), excitation waves propagating in the BZ reactor are sensed by photodiodes, which in turn trigger finger motion. When the bending fingers touch the chemical medium, glass “nails” fitted with capillary tubes release small quantities of colloidal silver into the solution and this triggers additional circular waves in

the medium (Adamatzky et al. 2005) (Fig. 7). Starting from any initial configuration, the chemical-robotic system does always reach a coherent activity mode, where fingers move in regular, somewhat melodic patterns, and just a few generators of target waves govern the dynamics of the excitation in the reactor (Yokoi et al. 2004).

### Parallel Actuators

How can a reaction-diffusion medium manipulate objects? To find out we can couple a simulated abstract parallel manipulator (Adamatzky et al. 2006) with an experimental Belousov-Zhabotinsky (BZ) chemical medium. The simulated manipulator is a two-dimensional array of actuating units, each unit can apply a force vector of unit length to the manipulated objects. The velocity vector of the object is derived via the integration of results of all local force vectors acting on the manipulated object.

Coupling of an excitable chemical medium with a manipulator allows us to convert experimental snapshots of the BZ medium to a force vector field and then simulate the motion of manipulated objects in the force field, thus achieving reaction-diffusion medium controlled actuation. To built an interface between the recordings of space-time snapshots of the excitation dynamics in BZ medium and simulated physical objects, we calculate force fields generated by mobile



**Reaction-Diffusion Computing, Fig. 7** Robotic hand interacts with Belousov-Zhabotinsky medium (From Yokoi et al. (2004))

excitation patterns and then simulate the behavior of an object in this force field (Skachek et al. 2005).

Chemical medium to perform actuation is prepared following the typical recipe (see Field and Winfree 1979), based on a ferroin catalyzed BZ reaction. A silica gel plate is cut and soaked in a ferroin solution. The gel sheet is placed in a Petri dish and BZ solution added. Dynamics of the chemical system is recorded at 30 s intervals using a digital camera.

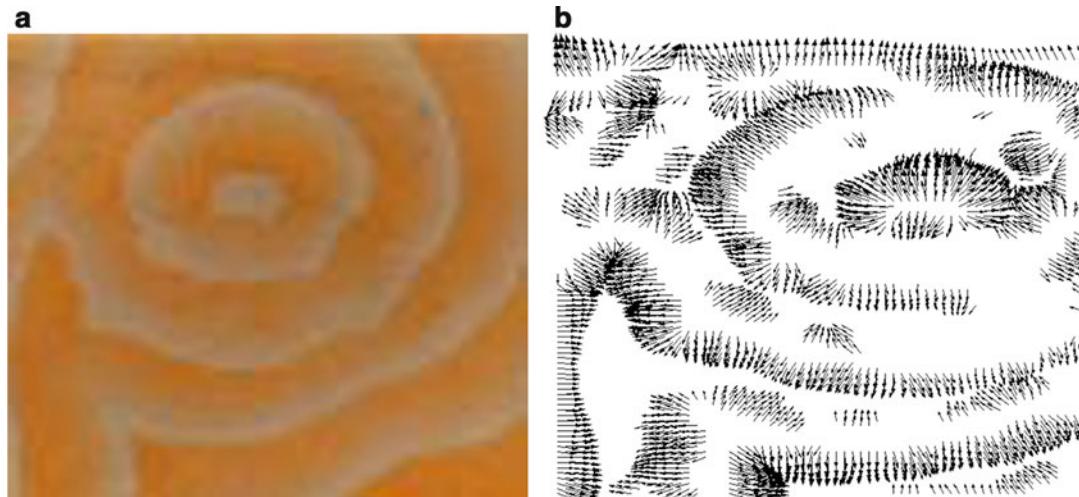
The cross-section profile of the BZ wavefront recorded on a digital snapshot shows a steep rise of red color values in the pixels at the wave-front's head and a gradual descent in the pixels along the wavefront's tail. Assuming that excitation waves push the object, local force vectors generated at each site – pixel of the digitized image – of the medium should be oriented along local gradients of the red color values. From the digitized snapshot of the BZ medium, we extract an array of red components from the snapshot's pixels and then calculate the projection of a virtual vector force at the pixel.

Force fields generated by the excitation patterns in a BZ system Fig. 8 result in tangential forces being applied to a manipulated object, thus causing translational and rotational motions of the object (Adamatzky et al. 2006).

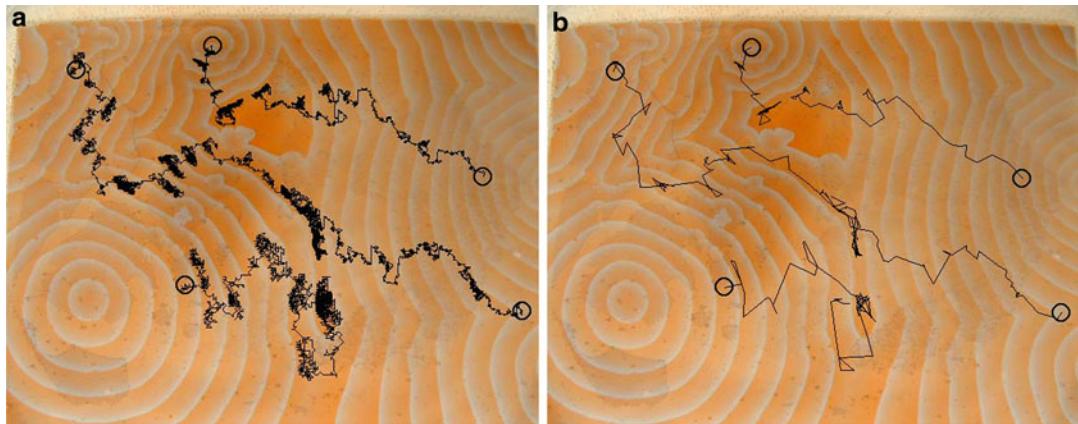
We demonstrated that the BZ medium controlled actuators can be used for sorting and manipulating both small objects, comparable in size to the elementary actuating unit, and larger objects, with lengths of tens or hundreds of actuating units.

Pixel-objects, due to their small size, are subjected to random forces, caused by impurities of the physical medium and imprecision of the actuating units. In this case, no averaging of forces is allowed and the pixel-objects themselves sensitively react to a single force vector. Therefore, we adopt the following model of manipulating a pixel-object: if all force vectors at the eight-pixel neighborhood of the current site of the pixel-object are nil then the pixel-object jumps to a randomly chosen neighboring pixel of its neighborhood; otherwise, the pixel-object is translated by the maximum force vector in its neighborhood.

When placed on the simulated manipulating surface, pixel-objects move at random in the domains of the resting medium, however by randomly drifting each pixel-object does eventually encounter a domain of coaligned vectors (representing excitation wavefront in BZ medium) and is translated along the vectors. An example of several pixel-objects transported on a “frozen” snapshot of the chemical medium is shown in Fig. 9. Trajectories of pixel-objects



**Reaction-Diffusion Computing, Fig. 8** Force vector field (b) calculated from BZ medium's image (a) (Adamatzky et al. 2006)



**Reaction-Diffusion Computing, Fig. 9** Examples of manipulating five pixel-objects using the BZ medium: (a) trajectories of pixel-objects, (b) jump-trajectories of

pixel-objects recorded every 100th time step. Initial positions of the pixel-objects are shown by circles (Adamatzky et al. 2006)

(Fig. 9a) show distinctive intermittent modes of random motion separated by modes of directed “jumps” guided by traveling wavefronts. Smoothed trajectories of pixel-objects (Fig. 9b) demonstrate that despite a very strong chaotic component in manipulation, pixel objects are transported to the sites of the medium where two or more excitation wave-fronts meet.

The overall speed of pixel-object transportation depends on the frequency of wave generations by sources of target waves. As a rule, the higher the frequency the faster the objects are transported. This is because in parts of the medium spanned by low-frequency target waves there are lengthy domains of resting states, where no force vectors are formed. Therefore, pixel-sized object can wander randomly for a long time until climbing the next wavefront (Adamatzky et al. 2006).

Spatially extended objects follow the general pattern of motion observed for the pixel-sized objects. However, due to integration of many force vectors, the motion of planar objects is smoother and less sensitive to the orientation of any particular force-vector.

Outcome of manipulation depends on the size of the object, with increasing size of the object – due to larger numbers of local vector-forces acting on the object – the objects become

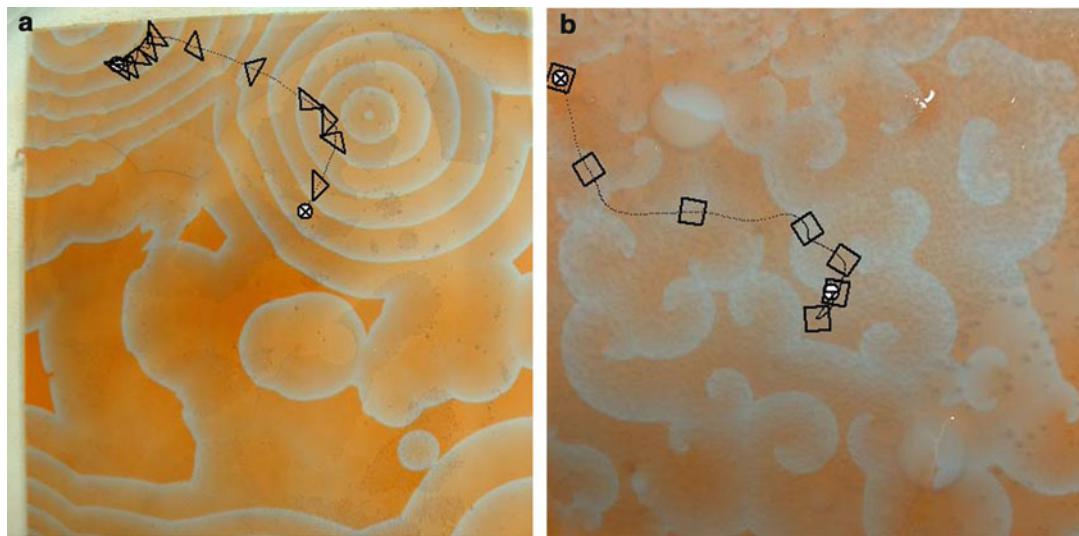
more controllable by the excitation wavefronts (Fig. 10).

## Universal Processors

Certain families of thin-layer reaction-diffusion chemical media can implement sensible transformation of initial (data) spatial distribution of chemical species concentrations to final (result) concentration profile (Adamatzky 2001; Sienko et al. 2003). In these reaction-diffusion computers, a computation is realized via spreading and interaction of diffusive or phase waves. Specialized, intended to solve a particular problem, experimental chemical processors implement basic operations of image processing (Adamatzky et al. 2005; Kuhnert 1986a; Rambidi 1998; Rambidi et al. 2002), computation of optimal paths (Adamatzky et al. 2005; Agladze et al. 1997; Steinbock et al. 1995), and control of mobile robots (Adamatzky et al. 2005).

A device is called computationally universal if it implements a functionally complete system of logical gates, e.g., a tuple of negation and conjunction, in its spacetime dynamics.

A number of computationally universal reaction-diffusion devices were implemented, the findings include logical gates (Tóth and



**Reaction-Diffusion Computing, Fig. 10** Manipulating planar object in BZ medium. (a) Right-angled triangle moved by fronts of target waves. (b) Square object moved by fronts of fragmented waves in subexcitable BZ medium. Trajectories of center of mass of the square are

shown by the dotted line. Exact orientation of the objects is displayed every 20 steps. Initial position of the object is shown by  $\ominus$  and the final position by  $\otimes$  (Adamatzky et al. 2006)

Showalter 1995; Sielewiesiuk and Gorecki 2001) and diodes (Dupont et al. 1998; Kusumi et al. 1997; Motoike and Yoshikawa 1999) in Belousov-Zhabotinsky (BZ) medium, and XOR gate in palladium processor (Adamatzky and De Lacy Costello 2002a).

Most known so far experimental prototypes of reaction-diffusion processors exploit interaction of wavefronts in a geometrically constrained chemical medium, i.e., the computation is based on a stationary architecture of medium's inhomogeneities. Constrained by stationary wires and gates reaction-diffusion chemical universal processors provide little computational novelty and no dynamical reconfiguration ability because they simply imitate architectures of conventional silicon computing devices. To appreciate in full the inherent massive-parallelism of thin-layer chemical media and to free the chemical processors from the imposed limitations of fixed computing architectures, we have adopted an unconventional paradigm of architecture-less, or collision-based, computing. An architecture-based, or stationary, computation implies that a logical circuit is

embedded into the system in such a manner that all elements of the circuit are represented by the system's stationary states. The architecture is static. If there is any kind of "artificial" or "natural" compartmentalization the medium is classified as an architecture-based computing device. Personal computers, living neural networks, cells, and networks of chemical reactors are typical examples of architecture-based computers.

A collision-based, or dynamical, computation employs mobile compact finite patterns, mobile self-localized excitations, or simply localizations, in active nonlinear medium. Essentials of collision-based computing are as follows.

Information values (e.g., truth values of logical variables) are given by either absence or presence of the localizations or other parameters of the localizations. The localizations travel in space and perform computation when they collide with each other. There are no predetermined stationary wires, a trajectory of the traveling pattern is a momentary wire. Almost any part of the medium space can be used as a wire. Localizations can collide anywhere within the medium's overall

space, there are no fixed positions at which specific operations occur, nor location specified gates with fixed operations. The localizations undergo transformations, form bound states, and annihilate or fuse when they interact with other mobile patterns. Information values of localizations are transformed as a result of the collision and thus a computation is implemented (Adamatzky et al. 2005).

The paradigm of collision-based computing originates from the technique of proving computational universality of the Game of Life (Berlekamp et al. 1982), conservative logic and billiard-ball model (Fredkin and Toffoli 1982), and their cellular-automaton implementations (Margolus 1984).

Solitons, defects in tubulin microtubules, excitons in Scheibe aggregates and breather in polymer chains are most frequently considered candidates for a role of information carrier in nature-inspired collision-based computers, see overview in Adamatzky (2001). It is experimentally difficult to reproduce all these artifacts in natural systems, therefore the existence of mobile localizations in an experiment-friendly chemical media opens new horizons for fabrication of collision-based computers.

The basis for material implementation of collision-based universality of reaction-diffusion chemical media is discovered by Sendiña-Nadal et al. (2001). They experimentally proved existence of localized excitations – traveling wave fragments which behave like quasi-particles – in photosensitive sub-excitable Belousov-Zhabotinsky medium.

### Basic Gates in Excitable Chemical Medium

Most collisions between wavefragments in sub-excitable BZ medium follow the basic rules of Fredkin-Toffoli billiard ball model but not always conservative. In most cases, particularly when chemical laboratory experiments are concerned, the wave-fragment do not simply scatter as a result of collision but rather fuse, split, or generate an additional new mobile localization. Two examples of logical gates realized in the BZ medium are shown in Fig. 11.

In the first example (Fig. 11a, b), the wave-fragment traveling North-West collides with the

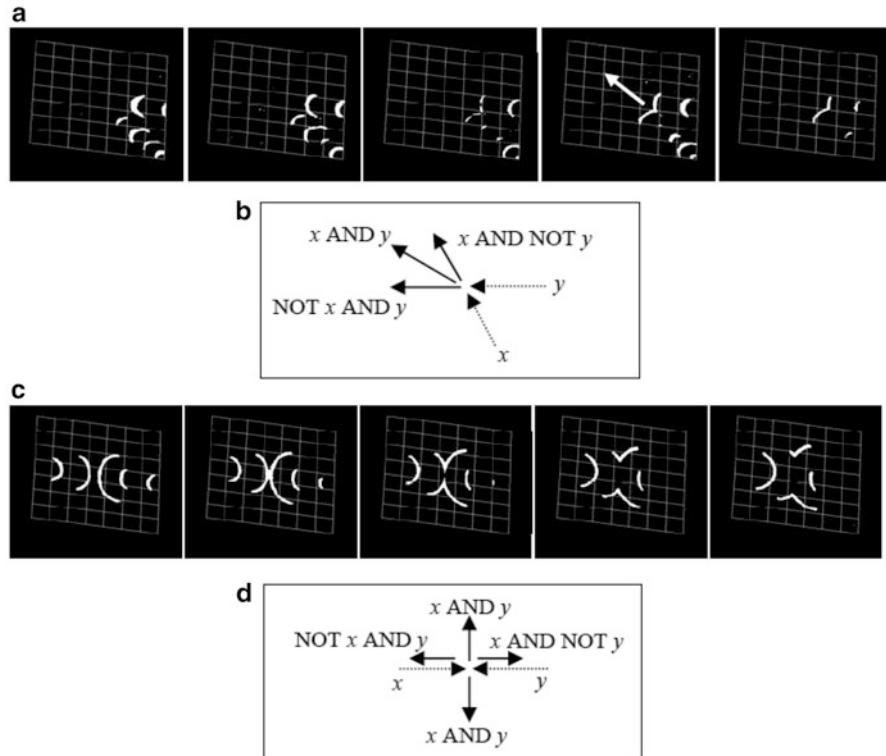
wave-fragment traveling West, a new wave-fragment traveling NorthWest-West is produced in the result of the collision (Fig. 11a). This new wave-fragment represents conjunction of the Boolean variables represented by the colliding wave-fragments (Fig. 11b).

In the second example of experimental implementation (Fig. 11c, d), we see two wave-fragments, traveling East and West, collides “head-on.” Two new wave-fragments are produced in the result of the collision. One wave-fragment travels North, another wave-fragment travels South. The corresponding logical gate is shown in (Fig. 11d).

### Constant Truth Generators

In previous section, we demonstrated the design of logical gates in theoretical and experimental implementations of a subexcitable BZ medium. The negation operator was the only thing missing in our construction, without negation and just one AND or OR gate a functional completeness could not be achieved. In cellular automata models of collision-based computers (see overview in Adamatzky 2002), negation is implemented with the help of glider guns – an autonomous generator of gliders. Streams of gliders, emitted by the glider gun are collided with streams of data gliders to perform the negation operation. Until recently we had no experimental proof that this missing component of current collision-based circuits could be obtained via experiments with the BZ medium. The gap was filled in de Lacy Costello et al. (2009) analogs of glider guns were realized in chemical laboratory conditions.

To produce the guns we used BZ light-sensitive subexcitable processor with a slight modification (see de Lacy Costello et al. 2009 for details). A controllable regular configuration of excitable and non-excitabile cells is created by projecting a  $10 \times 10$  checkerboard pattern comprising low ( $0.394 \text{ mW cm}^{-2}$ ) and high ( $9.97 \text{ mW cm}^{-2}$ ) light intensity cells onto the gel surface using a data projector. In the checkboard pattern, wave propagation depends on junctions between excitable and non-excitabile cells. Junction permeability is affected



**Reaction-Diffusion Computing, Fig. 11** To logical gates implemented in experimental laboratory conditions (Tóth et al. 2009). Time lapsed snapshots of propagating

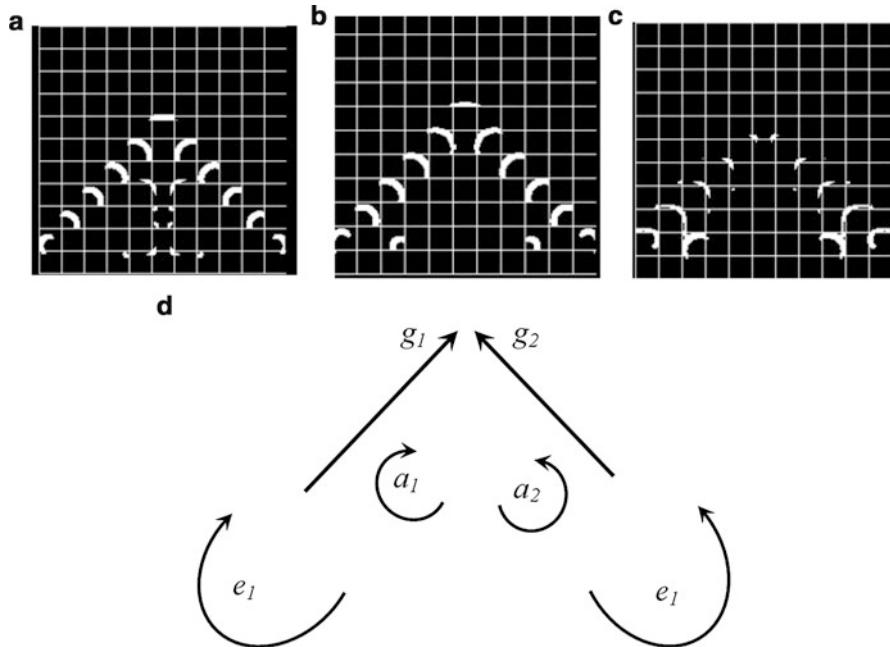
wave-fragments are shown in (a) and (c), schemes of the logical gates are shown in (b) and (d)

by the excitability of the cell, the excitability of the neighboring cells, the size and symmetry of the network, the curvature of the wave fragments in the excitable cells, and the previous states of the junctions. When a junction fails, it acts as an impurity in the system, which leads to formation of a spiral wave. In certain conditions, a tail of the spiral wave becomes periodically severed. Thus independently moving localized excitations are produced. The spiral generates trains of wave fragments. An example of two gliders guns, generating trains of mobile self-localized excitations, is shown in Fig. 12.

## Arithmetic Circuits

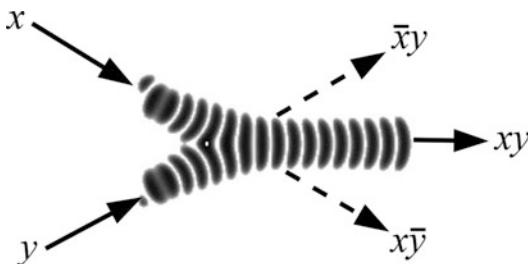
While designing reaction-diffusion computers, we found that when the excitation wave-fragments in a thin-layer Belousov-Zhabotinsky

(BZ) medium collide at an acute angle they fuse into a single wave-fragment which propagates along bisector of the collision angle (Fig. 13). Thus a fusion gate came into play (Adamatzky 2004). If one of the wave-fragments was not present another wave-fragment would move along its original trajectory. We interpret the presence and absence of wave-fragments at given site at given time as TRUE (“1”) and FALSE (“0”) values of Boolean variables. Let wave-fragment traveling South-East represent value  $x$  and wave-fragment traveling North-East represent value  $y$ . If  $y = 0$ , the corresponding wave-fragment is not present. Then the wave-fragment  $x$  continues its travel undisturbed. Thus its output trajectory represents  $\bar{x}y$  (Fig. 13). The output trajectory of undisturbed wave-fragment  $y$  represents  $\bar{x}y$ . When both input variables are TRUE, the wave-fragments  $x$  and  $y$  collide and merge into a single



**Reaction-Diffusion Computing, Fig. 12** Two glider guns where streams of fragments are in collision. (a–c) Snapshots of excitation dynamics in the BZ medium. (d) Scheme of the glider guns:  $e_1$  and  $e_2$  spiral wave fragments

generating streams,  $g_1$  and  $g_2$  of localized excitations,  $a_1$  and  $a_2$  auxiliary wave fragments interacting with the wave streams (From de Lacy Costello et al. 2009)



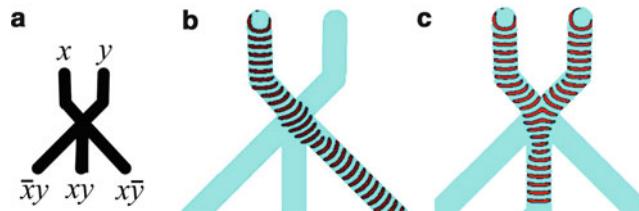
**Reaction-Diffusion Computing, Fig. 13** Time lapse overlays of the fusion of two excitation wave-fragments. One fragment is traveling from north-west to south-east, another fragment from south-west to north-east. The wave-fragments collide and fuse into a new localized excitation traveling east. Note that these are not trains of waves but two single-wave-fragments recorded at regular time intervals and superimposed on the same image

wave-fragment. This newly born wave-fragment represents conjunction  $xy$ .

Interaction of wave-fragments shown in Fig. 13 happens in a free space. Advantage of this is that literally any loci of space can be a

signal conductor, cause “wires” are momentary. Disadvantage is that, due to instability of wave-fragments which either expand or collapse, we must continuously monitor a size of the wave-fragment and adjust excitability of the medium to conserve the fragment’s shape. A compromise can be achieved by geometrically constraining excitation waves to excitable channels and allowing wave-fragments to interact at subexcitable junctions. In Adamatzky (2015), we employed this type of collision to implement a very simple one-bit half-adder. Propagation of excitation waves in the fusion gates and the adder is illustrated by numerical integration of two-variable Oregonator model (see details in Adamatzky 2015).

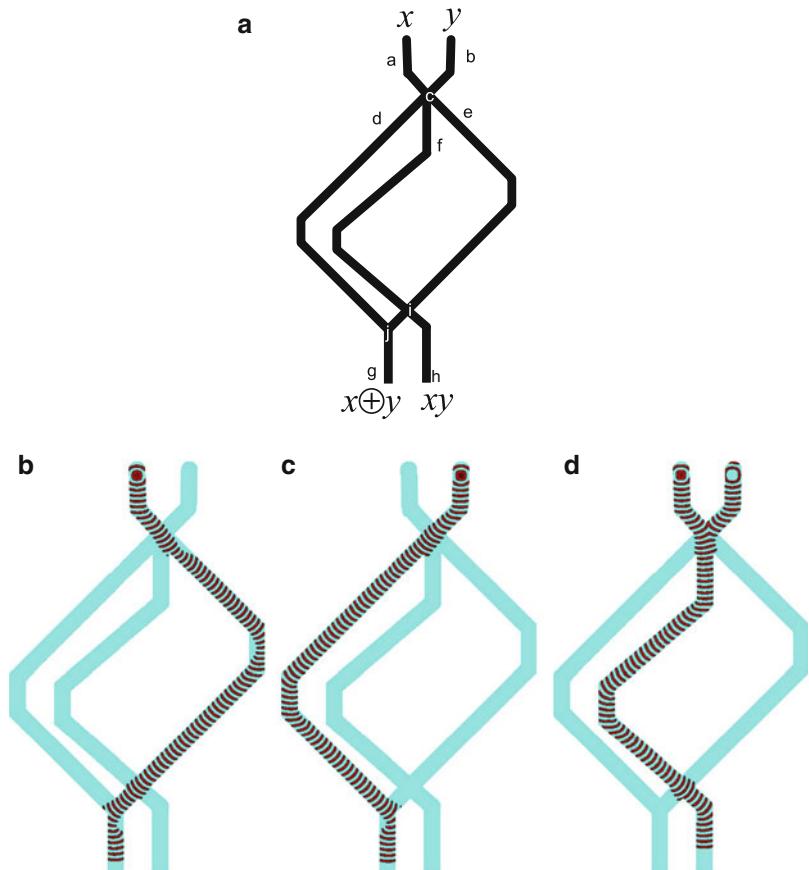
Geometry of the fusion gate  $F$  is shown in Fig. 14a. Channels are excitable. That is if the channels’ width and length were infinite the excitation wave would be a growing circle. Thus we do not waste resources by controlling excitability when signals are in transit. If we kept a junction excitable then excitation would propagate to all



**Reaction-Diffusion Computing, Fig. 14** Fusion gate  $F$ . Channels are excitable. Junctions are subexcitable. (a) Scheme: inputs are  $x$  and  $y$ , outputs are  $xy$ ,  $\bar{x}y$ ,  $x\bar{y}$ . (b, c) Time lapsed overlays of excitation waves. Note that these

are not trains of waves but two single-wave-fragments recorded at regular time interval and superimposed on the same image. (b)  $x = 1$ ,  $y = 0$ . (c)  $x = 1$ ,  $y = 1$  (From Adamatzky (2015)))

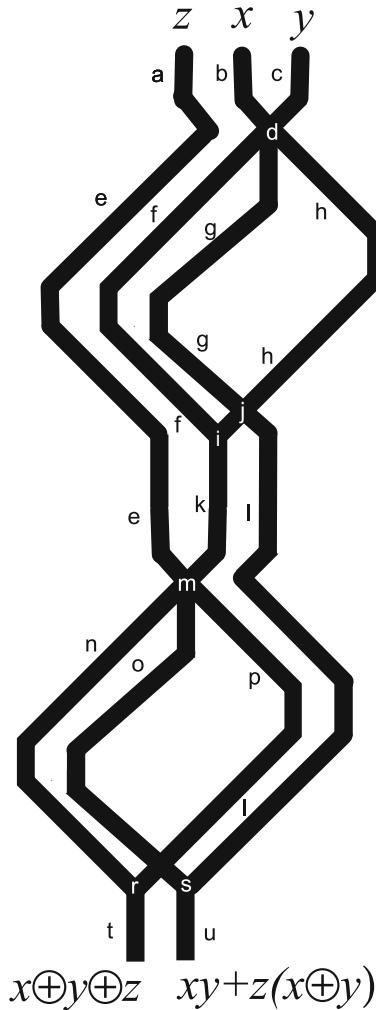
**Reaction-Diffusion Computing, Fig. 15** A one-bit half-adder filled with excitable chemical medium. (a) Scheme of the half-adder, gate A:  $a$  and  $b$  are input channels;  $g$  and  $h$  are output channels;  $d$ ,  $e$ ,  $f$  are internal channels; and  $c$ ,  $i$ , and  $j$  are junctions. Input variables  $x$  and  $y$  are fed into channels  $a$  and  $b$ ; results  $x \oplus y$  and  $xy$  are read from channels  $g$  and  $h$ . (b-d) Time lapsed overlays of excitation waves propagation for inputs (b)  $x = 1$ ,  $y = 0$ , (c)  $x = 1$ ,  $y = 0$ , (d)  $x = 1$ ,  $y = 1$ . Sites of initial perturbation are visible as discs. These are time lapsed snapshots of a single wave (b, c) or two waves merging into a single wave (From Adamatzky (2015))



output channels. We keep the junction subexcitable and therefore wave-fragments either propagate across the junction, along their original trajectories, without spreading into branching channels (just one input is “1” Fig. 14c) or

collide and merge into a single wave-fragment entering the central channel (two inputs are “1”, Fig. 14e).

By merging two lateral output channels  $\bar{x}y$  and  $\bar{x}\bar{y}$  into a single output channel we get a



**Reaction-Diffusion Computing, Fig. 16** A scheme of a one-bit full-adder: input channels are *a*, *b*, and *c*; output channels are *t* and *u*; internal channels *e*, *f*, *g*, *h*, *k*, *l*, *n*, *o*, *p*; and junctions are *d*, *i*, *j*, *m*, *r*, *s*. Input variables *x* and *y* are fed in channels *b* and *c*, Carry In is fed in channel *a*, and results  $x \oplus y \oplus z$  and  $xy + z(x \oplus y)$  are read from channels *t* and *u* (From Adamatzky (2015))

one-bit half-adder, gate *A*. A one-bit half-adder is a device with two inputs *x* and *y* and two outputs  $xy$  (Carry out) and  $x \oplus y$  (Sum) (Fig. 15a). It consists of two input channels *a* and *b* and two output channels *g* and *h*. When gate *A* is filled with BZ medium, we call this device  $A_{BZ}$ . Presence/absence of a

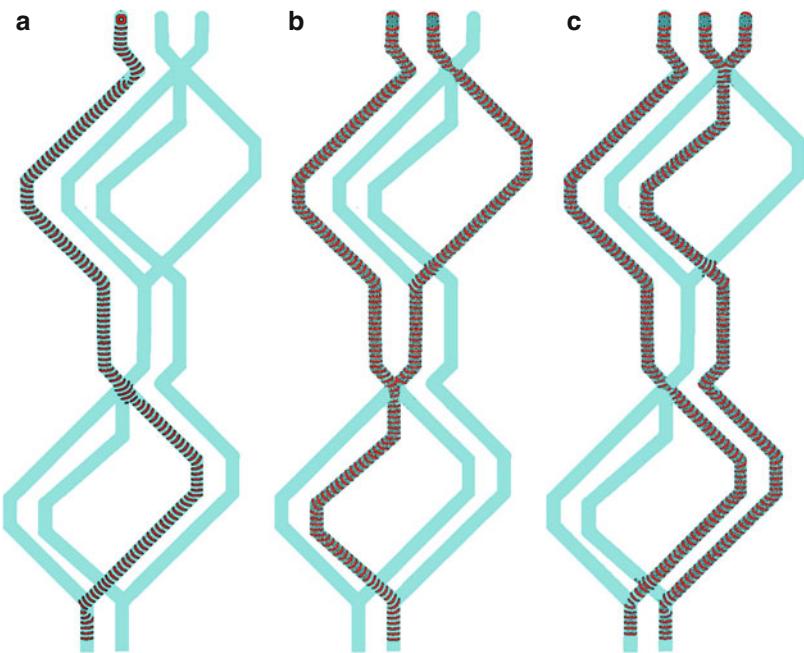
wave-fragment in an input/output channel of  $A_{BZ}$  symbolizes logical TRUE/FALSE state of the input variable assigned to the channel. Synchronization of signal wave-fragments is achieved geometrically:  $|a| + |e| + |g| = |b| + |d| + |g| = |a| + |f| + |h|$ , where  $|\cdot|$  is a length of a channel (Fig. 15a). Functioning of the half-adder  $A_{BZ}$  is shown in (Fig. 15b–d).

A one-bit full adder is a device with three inputs *x*, *y*, and *z* (Carry in) and two outputs  $xy + z(x \oplus y)$  (carry out) and  $x \oplus y \oplus z$  (Sum). The full adder is made of two half-adders (Fig. 16) by cascading Carry in into input channel of second half-adder, Sum of first half-adder into second input of second half-adder, and Carry out of first half-adder into OR junction with Carry out output channel of second half-adder. The one-bit full adder has three inputs channels *a*, *b*, and *c* (they represent carry in *z*, and added bits *x* and *y*) and two output channels *t* and *u* (they represent sum  $x \oplus y \oplus z$  and carry out  $xy + z(x \oplus y)$ ). Synchronization of signals is achieved geometrically:  $|a| + |e| + |p| + |t| = |b| + |h| + |k| + |n| + |t| = |c| + |f| + |k| + |n| + |t| = |b| + |g| + |l| + |u|$ .

The one-bit full adder in action is shown in Fig. 17. When Carry in has bit up, *z* = 1, and two inputs bits are down, *x* = 0 and *y* = 0, the wave-fragment is originated only in input channel *a* (Fig. 17a). The wave travels along channel *e*, through junction *m*, enters channel *p*, propagates through junction *r* and finishes its journey in output channel *t*. For input tuple *z* = 0, *x* = 1 and *y* = 0, an excitation wavefront is initiated in input channel *b*, propagates through junction *d* into channel *h*, through junctions *j* and *i* into channel *k*, through junction *n* into channel *n*, and finally enters the output channel *t*. When inputs are *z* = 0, *x* = 0, and *y* = 1, a wavefront initiated in input channel *c* propagates through *d*, along *f*, through *i*, along *k*, through *m*, along *n*, and ends up in output channel *t*. For combination of inputs *z* = 0, *x* = 1, and *y* = 1, wave-fragments are initiated in input channels *b* and *c*, they merge into a single wave-fragment at junction *d*. This newly born wave-fragment propagates along *g*, through *j*,

**Reaction-Diffusion Computing,**

**Fig. 17** Time lapsed overlays of excitation waves propagation in the one-bit full adder (Fig. 16) for inputs (a)  $x = 0, y = 0, z = 1$ , (b)  $x = 1, y = 0, z = 1$ , (c)  $x = 1, y = 1, z = 1$  (From Adamatzky (2015))



along  $l$ , through  $s$  and into output channel  $u$ . Dynamics of excitation waves for inputs  $z = 1, z = 1$  and  $y = 0$  is shown in Fig. 17b. The wave-fragments merge into a single wave-fragment at junction  $m$  and the newly born fragment propagates into output channel  $u$ . When all three input has state bit up,  $z = 1, x = 1, y = 1$ , the wave-fragment representing  $z = 1$  does not interact with the wave-fragment produced by merged wave-fragments representing  $x = 1$  and  $y = 1$  (Fig. 17c). Thus we have wave-fragments appearing on both output channels  $t$  and  $u$ .

Trajectories of wave-fragments for all possible combinations of inputs are shown in Table 1. The output channel  $t$  symbolizes Sum  $x \oplus y \oplus z$  and channel  $u$  Carry out  $xy + z(x \oplus y)$ .

## Future Directions

We listed major achievements in the field of reaction-diffusion computing and showed using selected examples of chemical laboratory

prototypes that chemical reaction-diffusion computers can solve a variety of tasks. They can perform image processing, calculate Voronoi diagram of arbitrary shaped planar objects and skeleton of planar shapes, approximate collision-free shortest path, guide robots toward sources of stimulation, manipulate several small objects in parallel, and implement logical functions. We provided recipes for the preparation of precipitating and excitable chemical media capable of carrying out such computations. The chapter we hope is sufficient in order for the reader to obtain a basic understanding of reaction-diffusion computing; however, readers inspired to discover and design their own chemical reaction-diffusion computers are advised to consult a more specific and thus more detailed text (Adamatzky et al. 2005). Future directions of research and development in reaction-diffusion computing will focus on design and experimental implementation of many-bit arithmetic logical units, embedded robotic processors, and realization of principles of reaction-diffusion computing at a nanoscale.

**Reaction-Diffusion Computing, Table 1** Schematic dynamic of one-bit full adder for all possible combinations of inputs. The adder, as shown in Fig. 16, is rotated to the left (From Adamatzky (2015))

$x$	$y$	$C_{in}$	$Sum$	$C_{out}$	Configuration
0	0	0	0	0	
1	0	0	1	0	
0	1	0	1	0	
1	1	0	0	1	
0	0	1	1	0	
1	0	1	0	1	
0	1	1	0	1	
1	1	1	1	1	

## Bibliography

- Adamatzky A (1994) Reaction-diffusion algorithm for constructing discrete generalized Voronoi diagram. *Neural Netw World* 6:635–643
- Adamatzky A (1996) Voronoi-like partition of lattice in cellular automata. *Math Comput Model* 23:51–66
- Adamatzky A (2001) Computing in nonlinear media and automata collectives. IoP Publishing, Bristol
- Adamatzky A (ed) (2002) Collision-based computing. Springer, London
- Adamatzky A (2004) Collision-based computing in Belousov–Zhabotinsky medium. *Chaos Solitons Fractals* 21:1259–1264
- Adamatzky A (2015) Binary full adder, made of fusion gates, in a subexcitable Belousov-Zhabotinsky system. *Phys Rev E* 92(3):032811
- Adamatzky A, De Lacy Costello BPJ (2002a) Experimental logical gates in a reaction-diffusion medium: the XOR gate and beyond. *Phys Rev E* 66:046112
- Adamatzky A, De Lacy Costello B (2002b) Experimental reaction-diffusion pre-processor for shape recognition. *Phys Lett A* 297:344–352
- Adamatzky A, De Lacy Costello BPJ (2002c) Collision-free path planning in the Belousov–Zhabotinsky medium assisted by a cellular automaton. *Naturwissenschaften* 89:474–478
- Adamatzky A, De Lacy Costello B (2007) Binary collisions between wave-fragments in a subexcitable Belousov-Zhabotinsky medium. *Chaos Solitons Fractals* 34:307–315
- Adamatzky A, Teuscher C (eds) (2006) From Utopian to genuine unconventional computers. Luniver Press, Beckington
- Adamatzky A, Tolmachiev D (1997) Chemical processor for computation of skeleton of planar shape. *Adv Mater Opt Electron* 7:135–139
- Adamatzky A, De Lacy Costello B, Melhuish C, Ratcliffe N (2003) Experimental reaction-diffusion chemical processors for robot path planning. *J Intell Robot Syst* 37:233–249
- Adamatzky A, De Lacy Costello B, Melhuish C, Ratcliffe N (2004) Experimental implementation of mobile robot taxis with onboard Belousov–Zhabotinsky chemical medium. *Mater Sci Eng C* 24:541–548
- Adamatzky A, De Lacy Costello B, Asai T (2005) Reaction diffusion computers. Elsevier, Amsterdam
- Adamatzky A, de Lacy Costello B, Skachek S, Melhuish C (2006) Manipulating objects with chemical waves: open loop case of experimental Belousov–Zhabotinsky medium coupled with simulated actuator array. *Phys Lett A* 350(3):201–209
- Adamatzky A, Bull L, De Lacy Costello B, Stepney S, Teuscher C (eds) (2007) Unconventional computing 2007. Luniver Press, Beckington
- Agladze K, Obata S, Yoshikawa K (1995) Phase-shift as a basis of image processing in oscillating chemical medium. *Physica D* 84:238–245
- Agladze K, Aliev RR, Yamaguchi T, Yoshikawa K (1996) Chemical diode. *J Phys Chem* 100:13895–13897
- Agladze K, Magome N, Aliev R, Yamaguchi T, Yoshikawa K (1997) Finding the optimal path with the aid of chemical wave. *Physica D* 106:247–254
- Akl SG, Calude CS, Dinneen MJ, Rozenberg G (eds) (2007) Unconventional computation: 6th international conference, UC 2007, Kingston, 13–17 Aug 2007, Proceedings. Springer
- Berlekamp ER, Conway JH, Guy RL (1982) Winning ways for your mathematical plays, vol 2. Academic Press, New York
- Blum HA (1967) Transformation for extracting new descriptors of shape. In: Wathen-Dunn W (ed) Models for the perception of speech and visual form. MIT Press, Cambridge, MA, pp 362–380
- Blum H (1973) Biological shape and visual science. *J Theor Biol* 38:205–287
- Calabi L, Hartnett WE (1968) Shape recognition, prairie fires, convex deficiencies and skeletons. *Am Math Mon* 75:335–342
- Courant R, Robbins H (1941) What is mathematics? Oxford University Press, Oxford
- De Lacy Costello BPJ (2003) Constructive chemical processors – experimental evidence that shows that this class of programmable pattern forming reactions exist at the edge of a highly non-linear region. *Int J Bifurcat Chaos* 13:1561–1564
- De Lacy Costello B, Adamatzky A (2003) On multitasking in parallel chemical processors: experimental findings. *Int J Bifurcat Chaos* 13:521–533
- de Lacy Costello BPJ, Hantz P, Ratcliffe NM (2004a) Voronoi diagrams generated by regressing edges of precipitation fronts. *J Chem Phys* 120(5): 2413–2416
- De Lacy Costello BPJ, Adamatzky A, Ratcliffe NM, Zanin A, Purwins HG, Liehr A (2004b) The formation of Voronoi diagrams in chemical and physical systems: Experimental findings and theoretical models. *Int J Bifurcat Chaos* 14(7):2187–2210
- de Lacy Costello B, Toth R, Stone C, Adamatzky A, Bull L (2009) Implementation of glider guns in the light-sensitive Belousov-Zhabotinsky medium. *Phys Rev E* 79 (2):026114
- Dupont C, Agladze K, Krinsky V (1998) Excitable medium with left-right symmetry breaking. *Physica A* 249:47–52
- Field RJ, Winfree AT (1979) Travelling waves of chemical activity in the Zaikin–Zhabotinsky–Winfree reagent. *J Chem Educ* 56:754
- Fredkin F, Toffoli T (1982) Conservative logic. *Int J Theor Phys* 21:219–253
- Fuerstman MJ, Deschatelets P, Kane R, Schwartz A, Kenis PJA, Deutch JM, Whitesides GM (2003) *Langmuir* 19:4714
- Gorecka J, Gorecki J (2003) T-shaped coincidence detector as a band filter of chemical signal frequency. *Phys Rev E* 67:067203

- Gorecki J, Yoshikawa K, Igarashi Y (2003) On chemical reactors that can count. *J Phys Chem A* 107:1664–1669
- Gorecki J, Gorecka JN, Yoshikawa K, Igarashi Y, Nagahara H (2005) *Phys Rev E* 72:046201
- Hwang YK, Ahuja N (1992) A potential field approach to path planning. *IEEE Trans Robot Autom* 8:23–32
- Ichino T, Igarashi Y, Motoike IN, Yoshikawa K (2003) Different operations on a single circuit: field computation on an excitable chemical system. *J Chem Phys* 118:8185–8190
- Klein R (1990) Concrete and abstract Voronoi diagrams. Springer, Berlin
- Kuhnert L (1986a) Photochemische Manipulation von chemischen Wellen. *Naturwissenschaften* 76:96–97
- Kuhnert L (1986b) A new photochemical memory device in a light sensitive active medium. *Nature* 319:393
- Kuhnert L, Agladze KL, Krinsky VI (1989) Image processing using light-sensitive chemical waves. *Nature* 337:244–247
- Kusumi T, Yamaguchi T, Aliev R, Amemiya T, Ohmori T, Hashimoto H, Yoshikawa K (1997) Numerical study on time delay for chemical wave transmission via an inactive gap. *Chem Phys Lett* 271:355–360
- Lemmon MD (1991) 2-degree-of-freedom robot path planning using cooperative neural fields. *Neural Comput* 3:350–362
- Margolus N (1984) Physics-like models of computation. *Physica D* 10:81–95
- Mills JW (2008) The nature of the extended analog computer. *Physica D* 237(9):1235–1256
- Motoike IN, Adamatzky A (2004) Three-valued logic gates in reaction-diffusion excitable media. *Chaos Solitons Fractals* 24:107–114
- Motoike I, Yoshikawa K (1999) Information operations with an excitable field. *Phys Rev E* 59:5354–5360
- Motoike IN, Yoshikawa K (2003) Information operations with multiple pulses on an excitable field. *Chaos Solitons Fractals* 17:455–461
- Motoike IN, Yoshikawa K, Iguchi Y, Nakata S (2001) Real-time memory on an excitable field. *Phys Rev E* 63:036220
- Nakagaki T, Yamada H, Tóth A (2001) *Biophys Chem* 92:47
- Rambidi NG (1997) Biomolecular computer: roots and promises. *Biosystems* 44:1–15
- Rambidi NG (1998) Neural network devices based on reaction-diffusion media: an approach to artificial retina. *Supramol Sci* 5:765–767
- Rambidi N (2003) Chemical-based computing and problems of high computational complexity: the reaction-diffusion paradigm. In: Seinko T, Adamatzky A, Rambidi N, Conrad M (eds) *Molecular computing*. MIT Press, Cambridge, MA
- Rambidi NG, Yakovenchuk D (2001) Chemical reaction-diffusion implementation of finding the shortest paths in a labyrinth. *Phys Rev E* 63:026607
- Rambidi NG, Shamayaev KR, Peshkov GY (2002) Image processing using light-sensitive chemical waves. *Phys Lett A* 298:375–382
- Saltenis V (1999) Simulation of wet film evolution and the Euclidean Steiner problem. *Informatica* 10:457–466
- Sendiña-Nadal I, Mihaliuk E, Wang J, Pérez-Muñoz V, Showalter K (2001) Wave propagation in subexcitable media with periodically modulated excitability. *Phys Rev Lett* 86:1646–1649
- Shirakawa T, Adamatzky A, Gunji YP, Miyake Y (2009) On simultaneous construction of Voronoi diagram and Delaunay triangulation by *Physarum polycephalum*. *Int J Bifurcat Chaos* 19(9):3109–3117
- Sielewiesiuk J, Gorecki J (2001) Logical functions of a cross junction of excitable chemical media. *J Phys Chem A* 105:8189–8195
- Sienko T, Adamatzky A, Rambidi N, Conrad M (eds) (2003) *Molecular computing*. MIT Press, Cambridge, MA
- Skachek S, Adamatzky A, Melhuish C (2005) Manipulating objects by discrete excitable media coupled with contact-less actuator array: open-loop case. *Chaos Solitons Fractals* 26:1377–1389
- Steinbock O, Tóth A, Showalter K (1995) Navigating complex labyrinths: optimal paths from chemical waves. *Science* 267:868–871
- Steinbock O, Kettunen P, Showalter K (1996) Chemical wave logic gates. *J Phys Chem* 100(49):18970
- Tolmachiev D, Adamatzky A (1996) Chemical processor for computation of Voronoi diagram. *Adv Mater Opt Electron* 6:191–196
- Tóth A, Showalter K (1995) Logic gates in excitable media. *J Chem Phys* 103:2058–2066
- Toth R, Stone C, Adamatzky A, de Lacy Costello B, Bull L (2009) Experimental validation of binary collisions between wave fragments in the photosensitive Belousov-Zhabotinsky reaction. *Chaos, Solitons Fractals* 41(4):1605–1615
- Yokoi H, Adamatzky A, De Lacy Costello B, Melhuish C (2004) Excitable chemical medium controlled by a robotic hand: closed loop experiments. *Int J Bifurcat Chaos* 14:3347–3354
- Zaikin AN, Zhabotinsky AM (1970) Concentration wave propagation in two-dimensional liquid-phase self-oscillating system. *Nature* 225:535



---

## Computing in Geometrical Constrained Excitable Chemical Systems

Jerzy Gorecki<sup>1</sup> and Joanna Natalia Gorecka<sup>2</sup>

<sup>1</sup>Institute of Physical Chemistry, Polish Academy of Science, Warsaw, Poland

<sup>2</sup>Institute of Physics, Polish Academy of Science, Warsaw, Poland

### Article Outline

Glossary

Definition of the Subject

Introduction

Logic Gates, Coincidence Detectors and Signal Filters

Chemical Sensors Built with Structured Excitable Media

The Ring Memory and Its Applications

Artificial Chemical Neurons with Excitable Medium

Perspectives and Conclusions

Conclusions

Bibliography

### Glossary

Some of the terms used in our article are the same as in the article of Adamatzky in this volume, and they are explained in the glossary of Reaction-Diffusion Computing.

**Activator** A substance that increases the rate of reaction.

**Excitability** Here we call a dynamical system excitable if it has a single stable state (the rest state) with the following properties: if the rest state is slightly perturbed then the perturbation uniformly decreases as the system evolves towards it. However, if the perturbation is

sufficiently strong it may grow by orders of magnitude before the system approaches the rest state. The increase in variables characterizing the system (usually rapid if compared with the time necessary to reach the rest state) is called an excitation. A forest is a classical example of excitable medium and a wildfire that burns it is an excitation. A dynamical system is non-excitatory if applied perturbations do not grow up and finally decay.

**Excitability level** The measure of how strong a perturbation has to be applied to excite the system. For example, for the Ru-catalyzed Belousov-Zhabotinsky reaction, increasing illumination makes the medium less excitable. The decrease in the excitability level can be observed in reduced amplitudes of spikes and in decreased velocities of autowaves.

**Firing number** The ratio between the number of generated excitations to the number of applied external stimulations. In most of the cases we define the firing number as the ratio between the number of output spikes to the number of arriving pulses.

**Inhibitor** A substance that decreases the rate of reaction or even prevents it.

**Medium** In the article we consider a chemical medium, fully characterized by local concentrations of reagents and external conditions like temperature or illumination level. The time evolution of concentrations is governed by a set of reaction-diffusion equations, where the reaction term is an algebraic function of variables characterizing the system and the non-local coupling is described by the diffusion operator. We are mainly concerned with a two dimensional medium (i.e., a membrane with a solution of reagents used in experiments), but the presented ideas can be also applied to one-dimensional or three-dimensional media.

**Refractory period** A period of time during which an excitable system is incapable of repeating its response to an applied, strong perturbation. After the refractory period the

excitable medium is ready to produce an excitation as an answer to the stimulus.

**Spike, autowave** In a spatially distributed excitable medium, a local excitation can spread around as a pulse of excitation. Usually a propagating pulse of excitation converges to a stationary shape characteristic for the medium, which is not dependent on initialization and propagates with a constant velocity - thus it is called an autowave.

**Subexcitability** A system is called subexcitable if the amplitude and size of the initiated pulse of excitation decreases in time. However, if the decay time is comparable with the characteristic time for the system, defined as the ratio between system size and pulse velocity, then pulses in a subexcitable system travel for a sufficiently long distance to carry information. Subexcitable media can be used to control the amplitude of excitations. Subexcitability is usually related to system dynamics, but it may also appear as the result of geometrical constraints. For example, a narrow excitable channel surrounded by a non-excitabile medium may behave as a subexcitable system because a propagating pulse dies out due to the diffusion of the activator in the neighborhood.

## Definition of the Subject

It has been shown in the article of Adamatzky Reaction-Diffusion Computing that an excitable system can be used as an information processing medium. In such a medium, information is coded in pulses of excitation; the presence of a single excitation or of a group of excitations forms a message. Information processing discussed by Adamatzky is based on a homogeneous excitable medium and the interaction between pulses in such medium.

Here we focus our attention on a quite specific type of excitable medium that has an intentionally introduced structure of regions characterized by different excitability levels. As the simplest case we consider a medium composed of excitable regions where autowaves can propagate and

non-excitabile ones where excitations rapidly die. Using such two types of medium one can, for example, construct signal channels: stripes of excitable medium where pulses can propagate surrounded by non-excitabile areas thick enough to cancel potential interactions with pulses propagating in neighboring channels. Therefore, the propagation of a pulse along a selected line in an excitable system can be realized in two ways. In a homogeneous excitable medium, it can be done by a continuous control of pulse propagation and the local feedback with activating and inhibiting factors (Reaction-Diffusion Computing). In a structured excitable medium, the same result can be achieved by creating a proper pattern of excitable and non-excitabile regions. The first method gives more flexibility, the second is just simpler and does not require a permanent watch. As we show in this article, a number of devices that perform simple information processing operations, including the basic logic functions, can be easily constructed with structured excitable medium. Combining these devices as building blocks we can perform complex signal processing operations. Such an approach seems similar to the development of electronic computing where early computers were built of simple integrated circuits.

The research on information processing with structured excitable media has been motivated by a few important problems. First, we would like to investigate how the properties of a medium can be efficiently used to construct devices performing given functions, and what tasks are the most suitable for chemical computing. There is also a question of generic designs valid for any excitable medium and specific ones that use unique features of a particular system (for example, a one-dimensional generator of excitation pulses that can be built with an excitable surface reaction (Gorecka and Gorecki 2005)). In information processing with a structured excitable medium, the geometry of the medium is as important as its dynamics, and it seems interesting to know what type of structures are related to specific functions. In the article we present a number of such structures characteristic for particular information processing operations.

Another important motivation for research comes from biology. Even the simplest biological organisms can process information and make decisions important for their life without CPU-s, clocks or sequences of commands as it is in the standard von Neumann computer architecture (Feynman et al. 2000). In biological organisms, even at a very basic cellular level, excitable chemical reactions are responsible for information processing. The cell body considered as an information processing medium is highly structured. We believe that analogs of geometrical structures used for certain types of information processing operations in structured excitable media will be recognized in biological systems, so we will better understand their role in living organisms. At a higher level, the analogies between information processing with chemical media and signal processing in the brain seems to be even closer because the excitable dynamics of calcium in neural tissue is responsible for signal propagation in nerve system (Haken 2002), Excitable chemical channels that transmit signals between processing elements look similar to dendrites and axons. As we show in the article, the biological neuron has its chemical analog, and this allows for the construction of artificial neural networks using chemical processes. Having in mind that neural networks are less vulnerable to random errors than classical algorithms one can go back from biology to man-made computing and adopt the concepts in a fast excitable medium, for example especially prepared semiconductors (Unconventional Computing, Novel Hardware for).

The article is organized in the following way. In the next section we discuss the basic properties of a structured chemical medium that seem useful for information processing. Next we consider the binary information coded in propagating pulses of concentration and demonstrate how logic gates can be built. In the following chapter we show that a structured excitable medium can acquire information about distances and directions of incoming stimuli Next we present a simple realization of read-write memory cell, discuss its applications in chemical counting devices, and show its importance for programming with pulses

of excitation In the following section we present a chemical realization of artificial neurons that perform multiargument operation on sets of input pulses. Finally we discuss the perspectives of the field, in particular more efficient methods of information coding and some ideas of self-organization that can produce structured media capable of information processing.

## Introduction

Excitability is the wide spread behavior of far-from-equilibrium systems (Kapral and Showalter 1995; Kuramoto 1984) observed, for example, in chemical reactions (Bielousov-Zhabotinsky BZ-reaction (Mikhailov and Showalter 2006), CO oxidation on Pt (Krischer et al. 1992), combustion of gases (Gorecki and Kawczynski 1996)) as well as in many other physical (laser action) and biochemical (signaling in neural systems, contraction of cardiovascular tissues) processes (Murray 1989). All types of excitable systems share a common property that they have a stable stationary state (the rest state) they reside in when they are not perturbed. A small perturbation of the rest state results only in a small-amplitude linear response of the system that uniformly decays in time. However, if a perturbation is sufficiently large then the system can evolve far away from the rest state before finally returning to it. This response is strongly nonlinear and it is accompanied by a large excursion of the variables through phase space, which corresponds to an excitation peak (a spike). The system is refractory after a spike, which means that it takes a certain recovery time before another excitation can take place. The excitability is closely related with relaxation oscillations and the phenomena differ by one bifurcation only (Plaza et al. 1997).

Properties of excitable systems have an important impact on their ability to process information. If an excitable medium is spatially distributed then an excitation at one point of the medium (usually seen as an area with a high concentration of a certain reagent), may introduce a sufficiently large perturbation to excite the neighboring points

as the result of diffusion or energy transport. Therefore, an excitation can propagate in space in the form of a pulse. Unlike mechanical waves that dissipate the initial energy and finally decay, traveling spikes use the energy of the medium to propagate and dissipate it. In a typical excitable medium, after a sufficiently long time, an excitation pulse converges to the stationary shape, independent of the initial condition what justifies to call it an autowave. Undamped propagation of signals is especially important if the distances between the emitting and receiving devices are large. The medium's energy comes from the non-equilibrium conditions at which the system is kept. In the case of a batch reactor, the energy of initial composition of reagents allows for the propagation of pulses even for days (Lázár et al. 1995), but a typical time of an experiment in such conditions is less than an hour. In a continuously fed reactor pulses can run as long as the reactants are delivered (Maselko et al. 1989).

If the refractory period of the medium is sufficiently long then the region behind a pulse cannot be excited again for a long time. As a consequence, colliding pulses annihilate. This type of behavior is quite common in excitable systems. Another important feature is the dispersion relation for a train of excitations. Typically, the first pulse is the fastest, and the subsequent ones are slower, which is related to the fact that the medium behind the first pulse has not relaxed completely (Sielewiesiuk and Gorecki 2002a). For example, this phenomenon is responsible for stabilization of positions in a train of pulses rotating on a ring-shaped excitable area. However, in some systems (Manz et al. 2000) the anomalous dispersion relation is observed and there are selected stable distances between subsequent spikes. The excitable systems characterized by the anomalous dispersion can play an important role in information processing, because packages of pulses are stable and thus the information coded in such packages can propagate without dispersion.

The mathematical description of excitable chemical media is based on differential equations of reaction-diffusion type, sometimes supplemented by additional equations that describe the

evolution of the other important properties of the medium, for example the orientation of the surface in the case of CO oxidation on a Pt surface (Suzuki et al. 2000; Krischer et al. 1992). Numerical simulations of pulse propagation in an excitable medium which are presented in many papers (Motoike and Yoshikawa 1999; Motoike et al. 2001) use the FitzHugh-Nagumo model describing the time evolution of electrical potentials in nerve channels (FitzHugh 1960; FitzHugh 1961; Nagumo et al. 1962). The models for systems with the Belousov-Zhabotinsky (BZ) reaction, for example the Rovinsky and Zhabotinsky model (Rovinsky and Zhabotinsky 1984; Rovinsky 1986) for the ferroin catalyzed BZ reaction with the immobilized catalyst, can be derived from "realistic" reaction schemes (Field et al. 1972) via different techniques of variable reduction. Experiments with the Ru-catalyzed, photosensitive BZ reaction have become standard in experiments with structured excitable media because the level of excitation can be easily controlled by illumination; see for example (Agladze et al. 2000; Gorecki et al. 2003; Ichino et al. 2003). Light catalyzes the production of bromine that inhibits the reaction, so non illuminated regions are excitable and those strongly illuminated are not. The pattern of excitable (dark) and non-excitible (transparent) fields is just projected on a membrane filled with the reagents. For example, the labyrinth shown in Fig. 1 has been obtained by illuminating a membrane through a proper mask. The presence of a membrane is important because it stops convection in the solution and reduces the speed and size of spikes, so studied systems can be smaller. The other methods of forming excitable channels based on immobilizing a catalyst by imprinting it on a membrane (Steinbock et al. 1995b) or attaching it by lithography (Gorecki et al. 2003; Suzuki et al. 1999; Suzuki et al. 2000) have been also used, but they seem to be more difficult.

Numerical simulations of the Ru-catalyzed BZ reaction can be performed with different variants of the Oregonator model (Amemiya et al. 2000; Field and Noyes 1974; Gaspar et al. 1983; Krug et al. 1990). For example, the three-variable model uses the following equations:

$$\varepsilon_1 \frac{\partial u}{\partial t} = u(1-u) - w(u-q) + D_u \nabla^2 u \quad (1)$$

$$\frac{\partial u}{\partial t} = u - v \quad (2)$$

$$\varepsilon_2 \frac{\partial w}{\partial t} = \phi + fv - w(u+q) + D_w \nabla^2 w \quad (3)$$

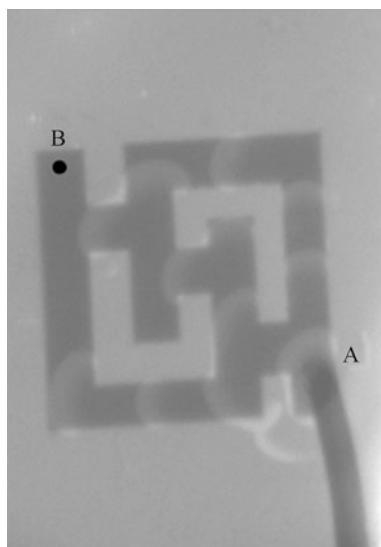
where  $u$ ,  $v$  and  $w$  denote dimensionless concentrations of the following reagents: HBrO<sub>2</sub>, Ru(4,4' - dm - bpy)<sub>3</sub><sup>3+</sup>, and Br<sup>-</sup>, respectively. In the considered system of equations,  $u$  is an activator and  $v$  is an inhibitor. The set of Oregonator equations given above reduces to the two-variable model cited in the article of Adamatzky Reaction-Diffusion Computing if the processes responsible for bromide production are very fast if compared to the other reactions ( $\varepsilon_2 \ll \varepsilon_1 \ll 1$ ). In such cases, the local value of  $w$  can be calculated assuming that it corresponds to the stationary solution of the third equation. If such a  $w$  is substituted into the first equation, one obtains the two-variable Oregonator model. In the equations given above, the units of space and time are dimensionless and they have been chosen to scale the reaction rates. Here we also neglected the diffusion of ruthenium catalytic complex because usually it is much smaller than those of the other reagents. The reaction-diffusion equations describing the time evolution of the system can be solved with the standard numerical techniques (Press et al. 2007).

The parameter  $\phi$  represents the rate of bromide production caused by illumination and it is proportional to the applied light intensity. Therefore, by adjusting the local illumination (or choosing the proper  $\phi$  as a function of space variables in simulations) we create regions with the required level of excitability, like for example excitable stripes insulated by a non-excitatory neighborhood. Of course, the reagents can freely diffuse between the regions characterized by different illuminations.

The structure of the equations describing the system's evolution in time and space gives the name “reaction diffusion computing”

(Adamatzky et al. 2005) to information processing with an excitable medium. Simulations play an important role in tests of potential information processing devices, because, unlike in experiment, the conditions and parameters of studied systems can be easily adjusted with the required precision and kept forever. Most of the information processing devices discussed below were first tested in simulations and next verified experimentally. One of the problems is related to the short time of a typical experiment in a batch condition (a membrane filled with reagents) that does not exceed 1 h. In such systems the period in which the conditions can be regarded as stable is usually shorter than 30 min and within this time an experimentalist should prepare the medium, introduce the required illumination and perform observations. The problem can be solved when one uses a continuously fed reactor (Maselko et al. 1989), but experimental setups are more complex. On the other hand, simulations often indicate that the range of parameters in which a given phenomenon appears is very narrow. Fortunately experiments seem to be more robust than simulations and the expected effects can be observed despite of inevitable randomness in reagent preparation. Having in mind the relatively short time of experiments in batch conditions and the low stability of the medium, we believe that applications of a liquid chemical excitable medium like the Ru-catalyzed BZ reaction as information processors (wetware) are rather academic and they are mainly oriented on the verification of ideas. Practical applications of reaction-diffusion computers will probably be based on an other type of medium, like structured semiconductors (Unconventional Computing, Novel Hardware for).

In an excitable reaction-diffusion medium, spikes propagate along the minimum time path. Historically, one of the first applications of a structured chemical medium used in information processing was the solution of the problem of finding the shortest path in a labyrinth (Steinbock et al. 1995a). The idea is illustrated in Fig. 1. The labyrinth is built of excitable channels (dark) separated by non-excitatory medium



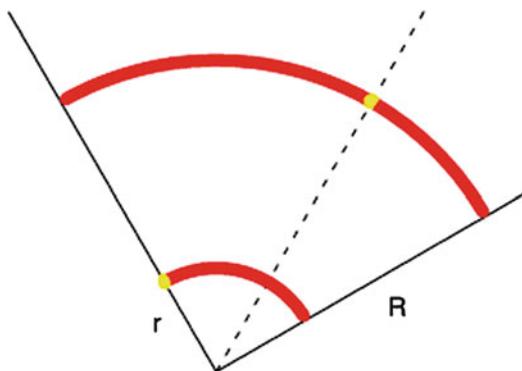
**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 1** Pulses of excitation propagating in a labyrinth observed in an experiment with a Ru-catalyzed BZ- reaction. The excitable areas are *dark*, the non-excitatory *light*. The source of a train of pulses (a tip of a silver wire) is placed at the point A

(light) that does not allow for interactions between pulses propagating in different channels. Let us assume that we are interested in the distance between the right bottom corner (point A) and the left upper corner (point B) of the labyrinth shown. The algorithm that scans all the possible paths between these points and selects the shortest one is automatically executed if the paths in labyrinth are build of an excitable chemical medium. To see how it works, let us excite the medium at the point A. The excitation spreads out through the labyrinth, separates at the junctions and spikes enter all possible paths. During the time evolution, pulses of excitation can collide and annihilate, but the one that propagates along the shortest path has always unexcited medium in front. Knowing the time difference between the moment when the pulse is initiated at the point A and the moment when it arrives at the point B and assuming that the speed of a pulse is constant, we can estimate the length of shortest path linking both points. The algorithm described above is called the “prairie fire” algorithm and it is automatically executed by an excitable medium. It finds the shortest path in a

highly parallel manner scanning all possible routes at the same time. It is quite remarkable that the time required for finding the shortest path does not depend on the complexity of labyrinth structure, but only on the distance between the considered points.

Although the estimation of the minimum distance separating two points in a labyrinth is relatively easy (within the assumption that corners do not significantly change pulse speed) it is more difficult to tell what is the shortest path. To do it one can trace the pulse that arrived first and plot a line tangential to its velocity. This idea was discussed in (Agladze et al. 1997) in the context of finding the length of the shortest path in a nonhomogeneous chemical medium with obstacles. The method, although relatively easy, requires the presence of an external observer who follows the propagation of pulses. An alternative technique of extracting the shortest path based on the image processing was demonstrated in (Rambidi and Yakovenchuk 1999). One can also locate the shortest path connecting two points in a labyrinth in a purely chemical way using the coincidence of excitations generated at the endpoints of the path. Such a method, described in (Gorecki and Gorecka 2006), allows one to find the midpoint of a trajectory, and thus locate the shortest path point by point.

The approximately constant speed of excitation pulses in a reaction-diffusion medium allows one to solve some geometrically oriented problems. For example one can “measure” the number  $\pi$  by comparing the time of pulse propagation around a non-excitatory circle of the radius  $d$  with the time of propagation around a square with the same side (Steinbock and Kettunen 1996). Similarly, a constant speed of propagating pulses can be used to obtain a given fraction of an angle. For example, a trisection of an angle can be done if the angle arms are linked with two arc-shaped excitable channels as shown in Fig. 2. The ratio of channel radii should be equal to 3. Both channels are excited at the same time at points on the same arm. The position of excitation at the larger arch at the moment when the excitation propagating on the shorter arch reaches the other arm belongs to a line that trisects the angle.



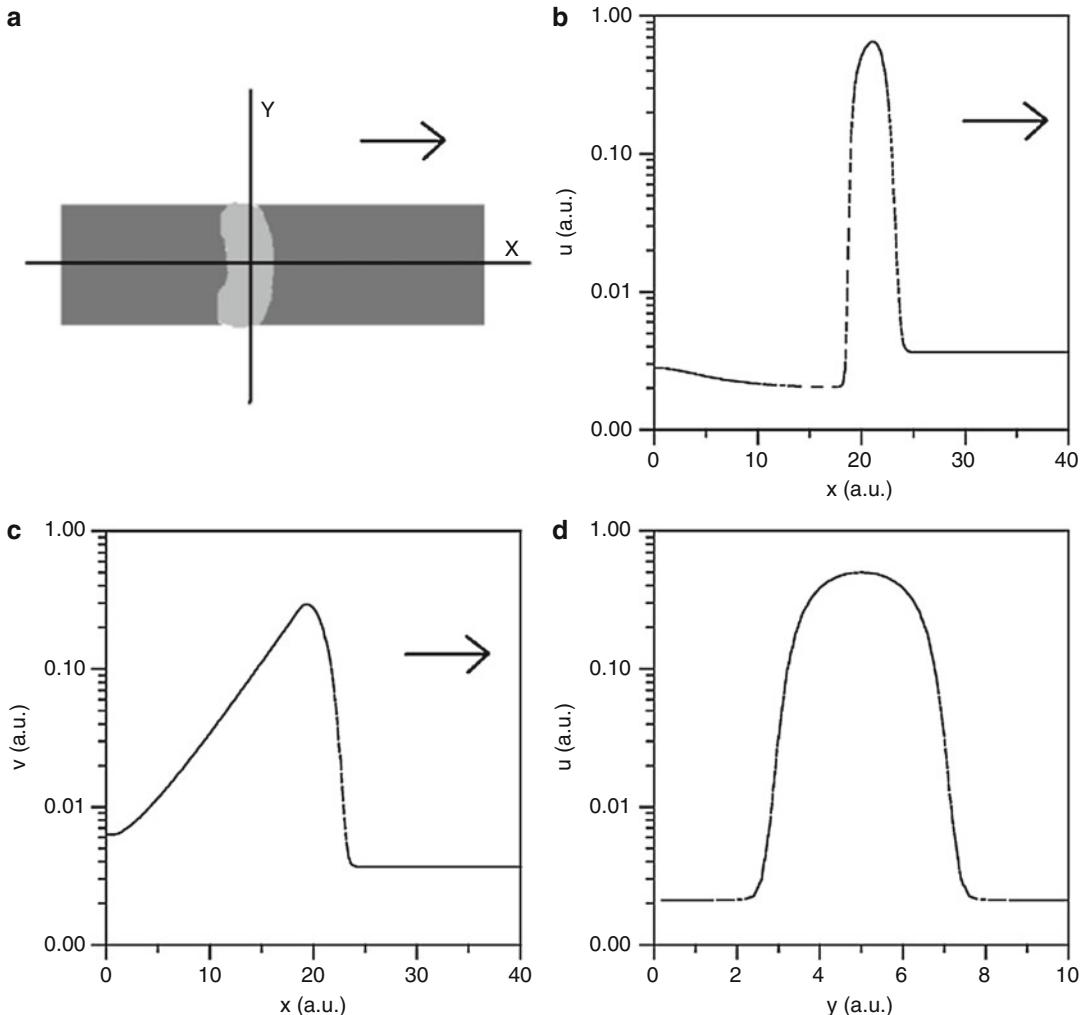
**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 2** The idea of angle trisection. Angle arms are linked with red arc shaped excitable channels with the radius ratio  $r : R = 1 : 3$ . The channels have been excited at the same time at the right arm of the angle. At the moment when the excitation pulse (a yellow dot) on a smaller arch reaches the second arm the excitation on the other arch shows a point on a line trisecting the angle (the dashed line)

The examples given above are based on two properties of structured excitable medium: the fact that the non-excitability of the neighborhood can restrict the motion of an excitation pulse to a channel, and that the speed of propagation depends on the excitability level of the medium but not on channel shape. This is true when channels are wide and the curvature is small. There are also other properties of an excitable medium useful for information processing. A typical shape of a pulse propagating in a stripe of excitable medium is illustrated in Fig. 3. The pulse moves from the left to the right as the arrow indicates. The profiles on Fig. 3b, c show cross sections of the concentrations of the activator  $u(x, y)$  and inhibitor  $v(x, y)$  along the horizontal axis of the stripe at a selected moment of time. The peak of the inhibitor follows activator maximum and is responsible for the refractory character of the region behind the pulse. Figure 3d illustrates the profile of an activator along the line perpendicular to the stripe axis. The concentration of the activator reaches its maximum in the stripe center and rapidly decreases at the boundary between the excitable and non-excitible areas. Therefore, the width of the excitable channel can be used as a parameter that controls the maximum concentration of an activator in a propagating pulse.

Figure 4 shows profiles of a concentration in an excitable channel with a triangular shape. The two curves on Fig. 4b illustrate the profile of  $u$  along the lines 1 and 2, respectively measured at the time when the concentration of  $u$  on a given line reaches its maximum. It can be seen that the maximum concentration of the activator decreases when a pulse propagates towards the tip of the triangle. This effect can be used to build a chemical signal diode. Let us consider two pieces of excitable medium, one of triangular shape and another rectangular, separated by a non-excitible gap as shown on Fig. 4a. It is expected that a perturbation of the rectangular area by a pulse propagating towards the tip of the triangular one is much smaller than the perturbation of the triangular area by a pulse propagating towards the end of rectangular channel. Using this effect, a chemical signal diode that transmits pulses only in one direction can be constructed just by selecting the right width of non-excitible medium separating two pieces of excitable medium: a triangular one and a rectangular one (Agladze et al. 1996; Kusumi et al. 1997).

The idea of a chemical signal diode presented in Fig. 4a was, in some sense, generalized by Davydov et al. (Morozov et al. 1999), who considered pulses of excitation propagating on a 2-dimensional surface in 3-dimensional space. It has been shown that the propagation of spikes on surfaces with rapidly changing curvature can be unidirectional. For example, such an effect occurs when an excitation propagates on the surface of a tube with a variable diameter. In such a case, a spike moving from a segment characterized by a small diameter towards a larger one is stopped.

For both of the chemical signal diodes mentioned above, the excitability of the medium is a non trivial function of some space variables. However, a signal diode can be also constructed when the excitability of the medium changes in one direction only, so in a properly selected coordinate system it is a function of a single space variable. For example, the diode behavior in a system where the excitability level is a triangular function of a single space variable has been confirmed in numerical simulations based on the Oregonator model of a photosensitive, Ru-catalyzed BZ

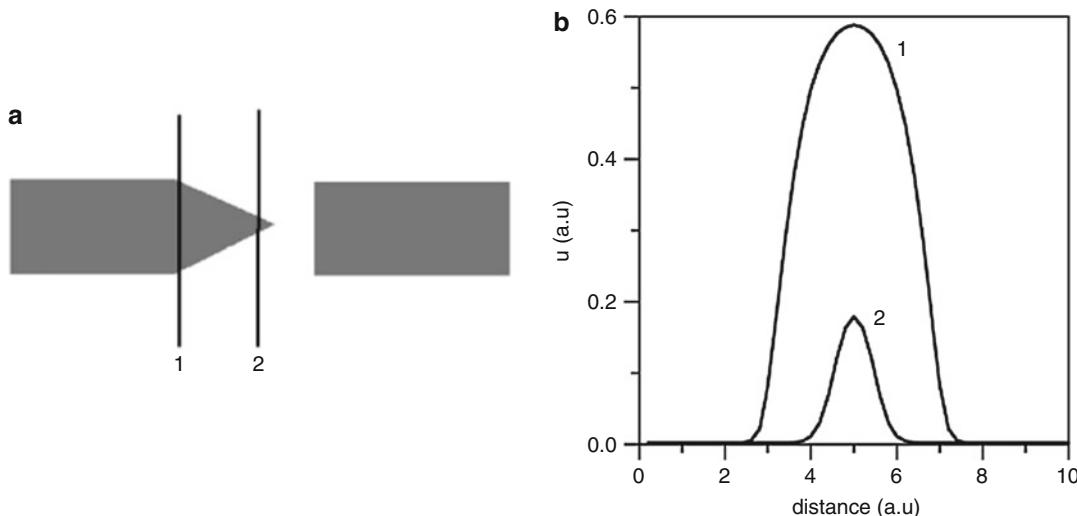


**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 3** The shape of the excitation pulse in a stripe of excitable medium. The arrow indicates the direction of propagation. Calculations were done for the Oregonator model (Eqs. 1–3) and the following values

of parameters were applied:  $f = 1.12$ ,  $q = 0.002$ ,  $\varepsilon_1 = 0.08$ ,  $\varepsilon_2 = 0.00097$ ,  $\phi_{\text{excitable}} = 0.007$ ,  $\phi_{\text{non-excitabile}} = 0.075$ . (a) The position of a spike on the stripe, (b, c) concentration of activator and inhibitor along the  $x$ -axis, (d) concentration of activator along the  $y$ -axis

reaction (Eqs. (1)–(3)) (Toth et al. 2001). If the catalyst is immobilized, then pulses that enter the region of inhomogeneous illumination from the strongly illuminated site are not transmitted, whereas the pulses propagating in the other direction can pass through (see Fig. 5a). A similar diode-like behavior resulting from a triangular profile of medium excitability can be expected for oxidation of CO on a Pt surface. Calculations demonstrate that a triangular profile of temperature (in this case, reduced with respect to that

which characterizes the excitable medium) allows for the unidirectional transmission of spikes characterized by a high surface oxygen concentration (Gorecka and Gorecki 2005). However, a realization of the chemical signal diode can be simplified. If the properties of excitable channels on both sites of a diode are the same, then the diode can be constructed with just two stripes of non-excitable medium characterized by different excitabilities as illustrated in Fig. 5b. If the symmetry is broken at the level of input channels then the



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 4** The shape of activator concentration in an excitable medium of a triangular shape. (a) The structure of excitable (dark) and non-excitatory (light)

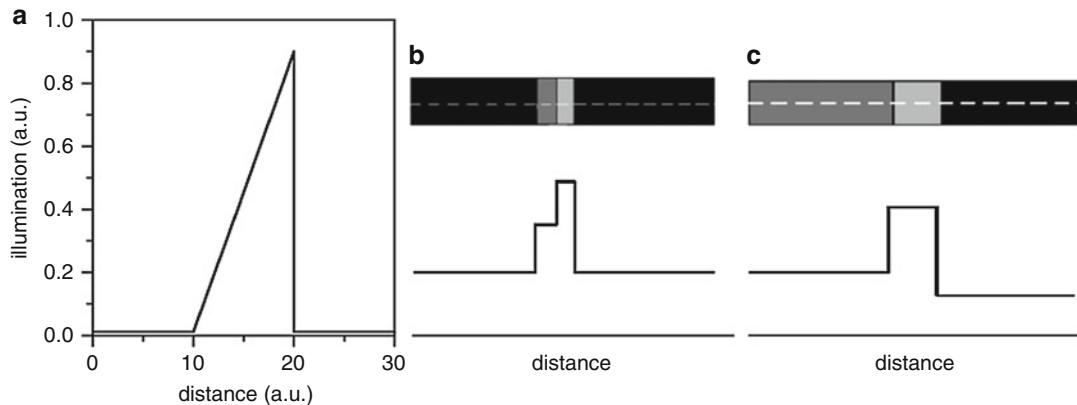
construction of a signal diode can be yet simpler, and it reduces to a single narrow non-excitatory gap with a much lower excitability than that of the neighboring channels (cf Fig. 5c). In both cases the numerical simulations based on the Oregonator model have shown that a diode works. The predictions of simulations have been qualitatively confirmed by experimental results (Gorecka et al. 2007). Different realizations of a signal diode show that even for very simple signal processing devices the corresponding geometrical structure of excitable and non-excitatory regions is not unique. Alternative constructions of chemical information processing devices seem important because they tell us on the minimum conditions necessary to build a device that performs a given function. In this respect a diode built with a single non-excitatory region looks interesting, because such situation may occur at a cellular level, where the conditions inside the cell are different from those around. The diode behavior in the geometry shown on Fig. 5c indicates that the unidirectional propagation of spikes can be forced by a channel in a membrane, transparent to molecules or ions responsible for signal transmission.

Wave-number-dependent transmission through a non-excitatory barrier is another feature of a

regions, (b) concentration of activator along the lines marked in (a). The profiles correspond to times when the activator reaches its maximum at a given line

chemical excitable medium important for information processing (Motoike and Yoshikawa 1999). Let us consider two excitable areas separated by a stripe of non-excitatory medium and a pulse of excitation propagating in one of those areas. The perturbation of the area behind the stripe introduced by an arriving pulse depends on the direction of its propagation. If the pulse wavevector is parallel to the stripe then the perturbation is smaller than in the case when it arrives perpendicularly (Motoike and Yoshikawa 1999). Therefore, the width of the stripe can be selected such that pulses propagating perpendicularly to the stripe can cross it, whereas a pulse propagating along the stripe do not excite the area on the other side. This feature is frequently used to arrange the geometry of excitable channels such that pulses arriving from one channel do not excite the other.

Non-excitatory barriers in a structured medium can play a more complex role than that described above. The problem of barrier crossing by a periodic train of pulses can be seen as an excitation via a periodic perturbation of the medium. It has been studied in detail in (Dolnik et al. 1989). The answer of the medium is quite characteristic in the form of a devil-staircase-like firing number as a function of perturbation strength. In the case



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 5** The excitability as a function of a space variable in different one-dimensional realizations of a signal diode with BZ- reaction inhibited by light: (a) a triangular profile of illumination, (b) the illumination

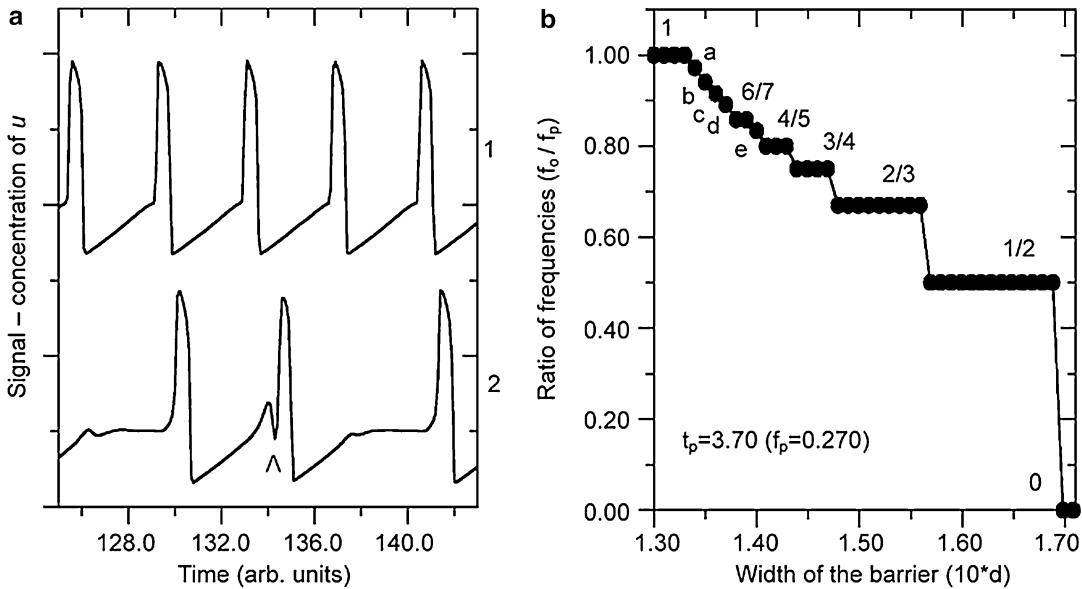
profile in a diode composed of two non-excitatory barriers, (c) the illumination profile in a single barrier diode with nonsymmetrical excitable inputs. The upper graphs in (b) and (c) illustrate illumination on a membrane used in experiments (Gorecka et al. 2007)

of barrier crossing, the strength of the excitation behind a barrier generated by an arriving pulse depends on the character of the non-excitatory medium, the barrier width, and the frequency of the incoming signal (usually, due to an uncompleted relaxation of the medium, the amplitude of spikes decreases with frequency). A typical complex frequency transformation after barrier crossing is illustrated in Fig. 6. Experimental and numerical studies on firing number of a transmitted signal have been published (Armstrong et al. 2004; Sielewiesiuk and Gorecki 2002b; Suzuki et al. 2000; Taylor et al. 2003). It is interesting that the shape of regions characterized by the same firing number in the space of two parameters, barrier width and signal frequency, is not generic and depends on the type of the excitable medium. Figure 7 compares the firing numbers obtained for FitzHugh-Nagumo and Rovinsky-Zhabotinsky models. In the first case, trains of pulses with small periods can cross wider barriers than trains characterized by low frequency; for the second model the dependence is reversed.

### Logic Gates, Coincidence Detectors and Signal Filters

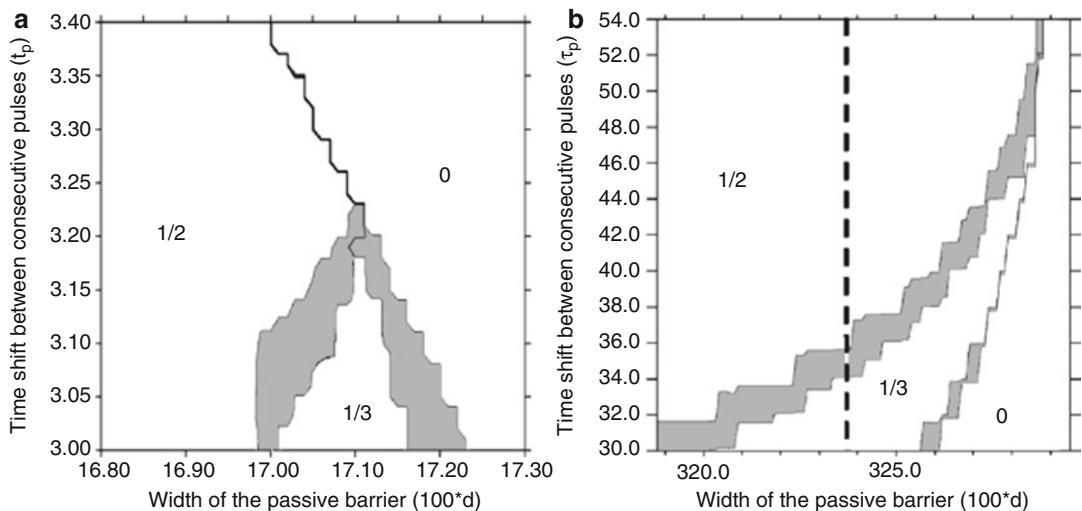
The simplest application of excitable media in information processing is based on the

assumption that the logical FALSE and TRUE variables are represented by the rest state and by the presence of an excitation pulse at a given point of the system respectively. Within such interpretation a pulse represents a bit of information propagating in space. When the system remains in its rest state, no information is recorded or processed, which looks plausible for biological interpretation. Information coded in excitation pulses is processed in regions of space where pulses interact (via collision and subsequent annihilation or via transient local change in the properties of the medium). In this section we demonstrate that the geometry of excitable channels and non-excitatory gaps can be tortured (the authors are grateful to prof S. Stepney for this expression) to the level at which the system starts to perform the simplest logic operations on pulses. The binary chemical logic gates can be used as building blocks for devices performing more complex signal processing operations. Information processing with structured excitable media is “unconventional” because it is performed without a clock that sequences the operations, as it is in the standard von Neumann type computer architecture. On the other hand, in the signal processing devices described below, the proper timing of signals is important and this is achieved by selecting the right length and geometry of channels. In some



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 6** Frequency transformation on a barrier – the results for the FitzHugh-Nagumo model (Sielewiesiuk and Gorecki 2002b). **(a)** The comparison of an arriving train of pulses (1) and the transmitted signal (2). **(b)** A typical dependence of the firing number as a function

of barrier width. The plateaus are labeled with corresponding values of firing number (1,  $6/7$ ,  $4/5$ ,  $3/4$ ,  $2/3$ ,  $1/2$  and 0). At points labeled a–e the following values have been observed: a –  $35/36$ ; b –  $14/15$ ; c –  $10/11$ ; d –  $8/9$  and e –  $5/6$



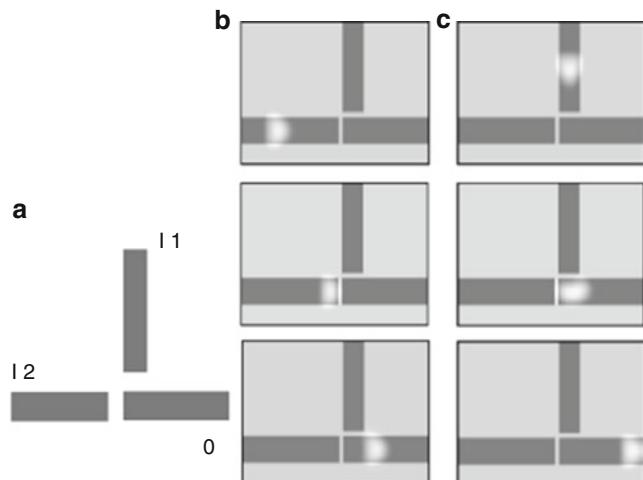
**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 7** The firing number as a function of the barrier width  $d$  and the interval of time between consecutive pulses ( $t_p$ ). Labels in the white areas give the firing number, and the gray color marks values of

parameters where more complicated transformations of frequency occur. **(a)** Results calculated for the FitzHugh-Nagumo model, **(b)** the Rovinsky-Zhabotinsky model. Both space and time are in dimensionless units

### Computing in Geometrical Constrained

#### Excitable Chemical

**Systems, Fig. 8** The distribution of excitable channels (*dark*) that form the OR gate. (b, c) illustrate the time evolution of a single pulse arriving from inputs I1 and I2, respectively



cases, for example for the operations on trains of pulses, the presence of a reference signal, which plays a role similar to a clock, would significantly help to process information (Steinbock et al. 1996). Historically, the logical gates were the first devices that have been realized with a structured chemical medium (Adamatzky and De Lacy Costello 2002; Adamatzky 2004; Epstein and Showalter 1996; Sielewiesiuk and Gorecki 2001a; Suzuki 1967; Toth and Showalter 1995).

The setup of channels that execute the logic sum (OR) operation is illustrated in Fig. 8 (Motoike and Yoshikawa 1999). The gate is composed of three excitable stripes (marked gray) surrounded by a non-excitatory medium. The gaps separating the stripes have been selected such that a pulse that arrives perpendicularly to the gap can excite the area on the other side and a pulse that propagates parallel to the gap does not generate sufficient perturbation to excite the medium behind the gap. If there is no input pulse, the OR gate remains in the rest state and does not produce any output. A pulse in any of the input channels I1 and I2 can cross the gap separating these channels from the output O and an output spike appears. The excitation of the output channel generated by a pulse from one of the input channels propagates parallel to the gap separating the other input channel so it does not interfere with the other input as seen in Fig. 8b, c. The frequency at which the described OR gate operates is limited by the refractory period of the output medium. If

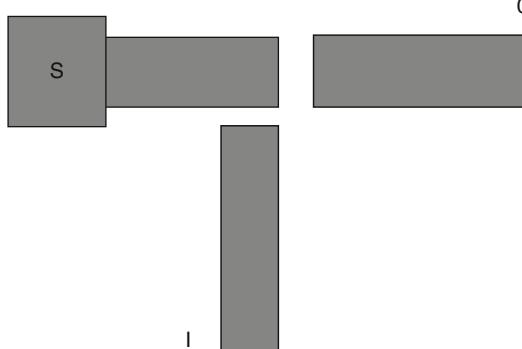
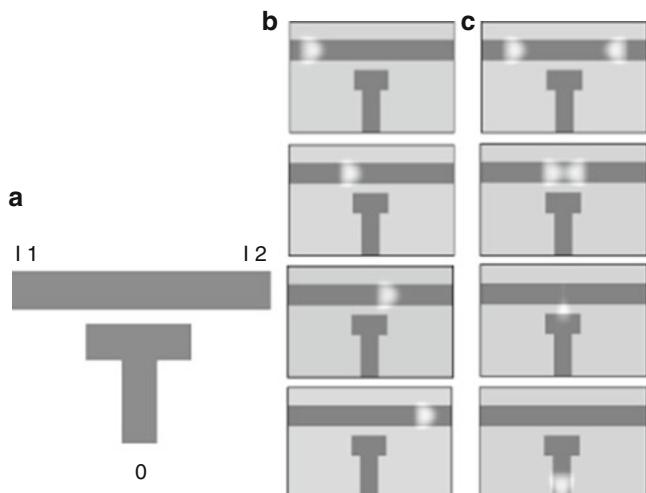
the signals from both input channels arrive, but the time difference between pulses is smaller than the refractory time, then only the first pulse will produce the output spike.

The gate that returns the logic product (AND) of input signals is illustrated in Fig. 9. The width of the gap separating the output channel O from the input one (I1,I2) is selected such that a single excitation propagating in the input channel does not excite the output (Fig. 9b). However, if two counterpropagating pulses meet, then the resulting perturbation is sufficiently strong to generate an excitation in the output channel (Fig. 9c). Therefore, the output signal appears only when both sites of the input channel have been excited and pulses of excitation collided in front of the output channel. The width of the output channel defines the time difference between input pulses treated as simultaneous. Therefore, the structure shown in Fig. 9 can also be used as a detector of time coincidence between pulses.

The design of the negation gate is shown in Fig. 10. The gaps separating the input channels, the source channel and the output channel can be crossed by a pulse that propagates perpendicularly, but they are non-penetrable for pulses propagating parallel to the gaps. The NOT gate should deliver an output signal if the input is in the rest state. Therefore, it should contain a source of excitation pulses (marked S). If the input is in the rest state then pulses from the source propagate unperturbed and enter the output channel. If

**Computing in Geometrical Constrained****Excitable Chemical****Systems, Fig. 9**

The distribution of excitable channels (dark) that form the AND gate. (b, c) illustrate the response of the gate to a single pulse arriving from input I1 and to a pair of pulses, respectively



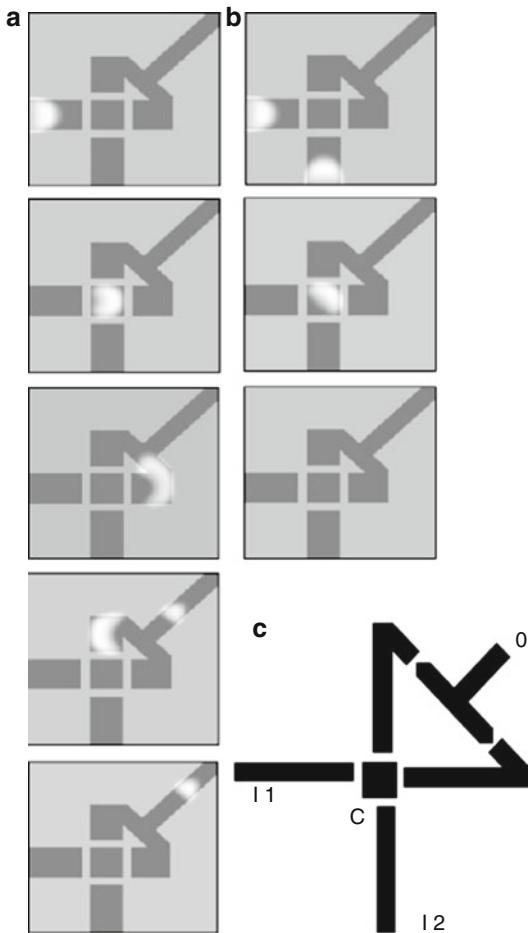
**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 10** The distribution of excitable channels (dark) that form the NOT gate

there is an excitation pulse in the input channel then it enters the channel linking the source with the output and annihilates with one of pulses generated by the source. As a result no output pulse appears. At the first look, the described NOT gate works fine, but if we assume that a single pulse is used in information coding than the input pulse should arrive at the right time to block the source. Therefore, additional synchronization of the source is required. If information is coded in trains of pulses then the frequency of the source should match with the one used for coding.

The structure of excitable channels for the exclusive OR (XOR) gate is illustrated in Fig. 11c (Igarashi et al. 2006). Two input channels

bring signals to the central area C. The output channels are linked together via diodes (Fig. 4) that stop possible backward propagation. As in the previous cases only pulses perpendicular to the gaps can pass through non-excitatory gaps between the central area and both input and output channels. The shape of the central area has been designed such that an excitation generated by a single input pulse propagates parallel to one of the outputs and is perpendicular to another. As the result one of the output channels is excited (see Fig. 11a). However, if pulses from both input channels arrive at the same time then the wavevector of the excitation in the central part is always parallel to the boundaries (Fig. 11b). Therefore, no output signal appears. Of course, there is no output signal if none of the inputs are excited. It is worth noticing that for some geometries of the XOR gate the diodes in output channels are not necessary because the backward propagation does not produce a pulse with a wavevector perpendicular to a gap as seen in the fourth frame of Fig. 11a.

Another interesting example of behavior resulting from the interaction of pulses has been observed in a cross-shaped structure built of excitable regions, separated by gaps penetrable for perpendicular pulses (Sielewiesiuk and Gorecki 2001a; Sielewiesiuk and Gorecki 2001b) shown in Fig. 12. The answer of cross-shaped junction to a pair pulses arriving from two perpendicular



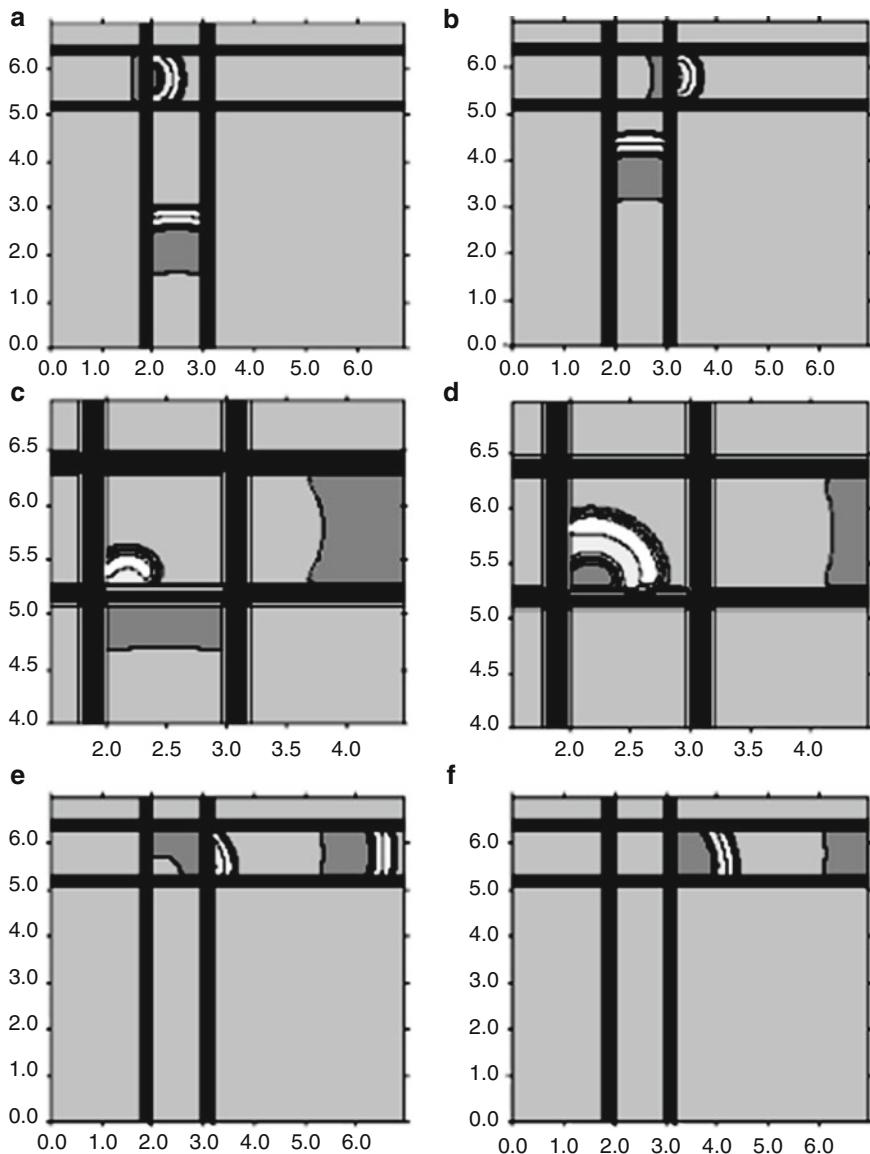
**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 11** (c) the distribution of excitable channels (dark) that form the XOR gate. (a, b) illustrate the response of the gate to a single pulse arriving from input I1 and to a pair of pulses, respectively

directions has been studied as a function of the time difference between pulses. Of course, if the time difference is large, pulses propagate independently along their channels. If the time difference is small the cross-junction acts like the AND gate and the output excitation appears in one of the corner areas. However, for a certain time difference the first arriving pulse is able to redirect the other and force it to follow. The effect is related to uncompleted relaxation of the central area of the junction at the moment when the second pulse arrives. Pulse redirection seems to be an interesting

effect from the point of programming with excitation pulses, but in practice it requires a high precision in selecting the right time difference.

The logic gates described above can also be applied to transform signals composed of many spikes. For example, two trains of pulses can be added together if they pass through an OR gate. The AND gate creates a signal composed of coinciding pulses from both trains. It is also easy to use a structured excitable medium and generate a signal that does not contain coinciding pulses (Motoike and Yoshikawa 2003). The structure of the corresponding device is illustrated in Fig. 13. All non-excitatory gaps are penetrable for perpendicular pulses. If a pulse of excitation arrives from one of the input channels then it excites the segment A and propagates towards the other end of it. If there is no spike in the other input channel then this excitation propagates unperturbed and activates the output channel. However, if a pulse from the other channel arrives it annihilates with the original pulse and no output signal is generated. The structure illustrated on Fig. 13a can be easily transformed into a device that compares two trains of pulses such that the resulting signal is composed of spikes of the first signal that do not coincide with the pulses of the second train. Such a device can be constructed just by neglecting one of the output channels. Figure 13b illustrates the geometry of the device that produces the output signal composed of pulses in arriving at the input I1 that have no corresponding spikes in the signal arriving from I2.

The coincidence detector can be used as a frequency filter that transmits periodic signals within a certain frequency interval (Gorecka and Gorecki 2003; Motoike and Yoshikawa 2003). The idea of such a filter is shown in Fig. 14. The device tests if the time between subsequent spikes of the train remains in the assumed range. As illustrated the signal arriving from the input I separates and enters the segment E through diodes D1 and D2. The segments E and F form a coincidence detector (or an AND gate, c. f Fig. 9). The excitation of the output appears when an excitation coming to E via the segment C and D2 is in the coincidence with the subsequent spike of the train that arrived directly via D1.

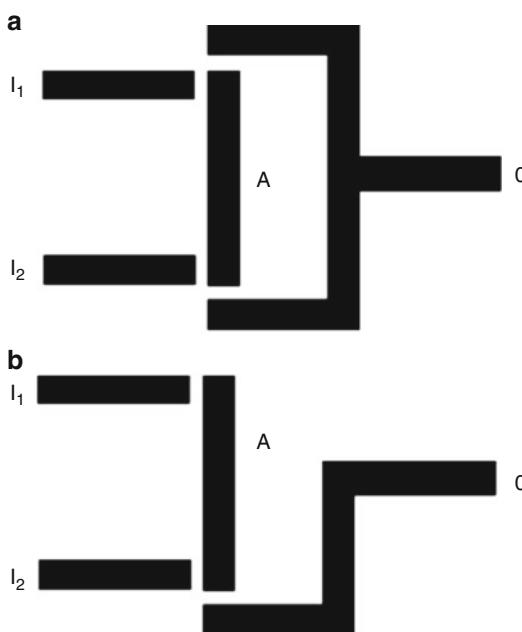


**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 12** The distribution of excitable and non-excitatory regions in a cross-shaped junction. Here the excitable regions are gray and the non-excitatory black.

The time shift for which coincidences are tested is decided by the difference in lengths of both paths. The time resolution depends on the width of the F channel (here  $l_2 - l_1$ ). For periodic signals the presented structure works as a frequency filter and transmits signals within the frequency range  $f \pm = v/(\Delta r \pm (\Delta w)/2)$ , where  $\Delta r$  is the

Consecutive figures illustrate an interesting type of time evolution caused by interaction of pulses. Two central figures are enlarged in order to show how uncompleted relaxation influences the shape of the next pulse

difference of distances traveled by spikes calculated to the point in E placed above the center of F channels ( $\sim 2^* l_e$ ),  $\Delta w$  is the width of the output channel, and  $v$  is the velocity of a spike in the medium. Typical characteristics of the filter are illustrated on Fig. 14b. The points represent results of numerical simulations and the line



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 13** The distribution of excitable channels in devices that compare trains of pulses. The device shown in (a) applies the XOR operation to a pair of signals and makes a signal composed of spikes that do not coincide with the other signal. (b) generates a signal composed of spikes that arrive through  $I_1$  and do not coincide with excitation pulses coming from  $I_2$

shows the filter characteristics calculated from the equation given above. It can be noticed that at the ends of the transmitted frequency band a change in output signal frequency is observed. This unwelcome effect can be easily avoided if a sequence of two identical filters is used. A filter tuned to a certain frequency will also pass any of its harmonics because the output signal can be generated by the coincidence of every second, third, etc. pulses in the train.

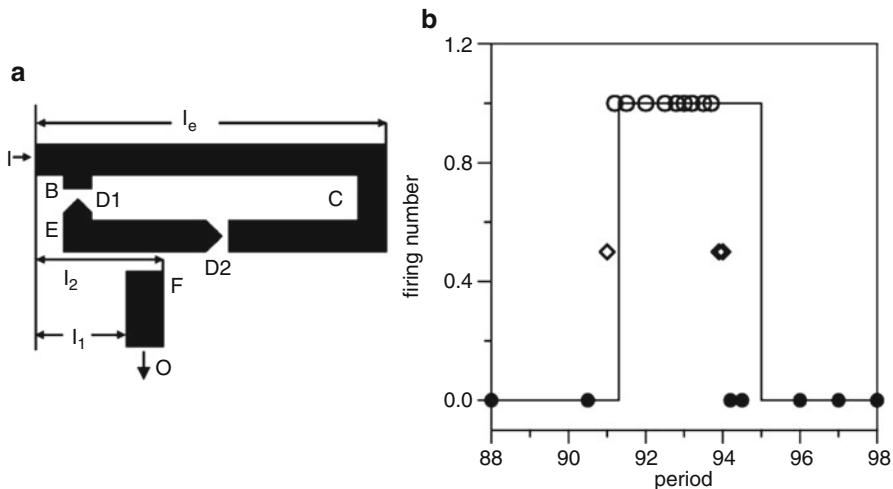
### Chemical Sensors Built with Structured Excitable Media

Even the simplest organisms without specialized nerve systems or brains are able to search for the optimum living conditions and resources of food. We should not be astonished by this fact because even chemical systems can receive and process

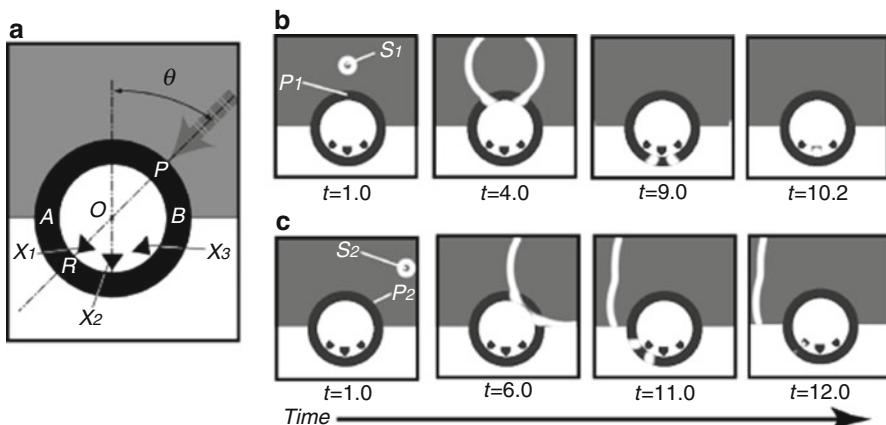
information arriving from their neighborhood. In this section we describe how simple structures built of excitable media can be used for direction and distance sensing. In order to simplify the conversion between an external stimulus and information processed by a sensor, we assume that the environment is represented by the same excitable medium as the sensor itself, so a stimulus can directly enter the sensor and be processed.

One possible strategy of sensing is based on a one-to-one relationship between the measured variable and the activated sensor channel (Nagahara et al. 2004). An example of a sensor of that type is illustrated on Fig. 15. It is constructed with a highly excitable, black ring surrounded by a number of coincidence detectors, denoted as  $X_1$ ,  $X_2$  and  $X_3$ . The excitation of a detector appears if a pair of pulses collide on the ring in front of it. Let us consider a homogeneous excitable environment and a spherical pulse of excitation with the source at the point  $S_1$ . High excitability of the ring means that excitations on the ring propagate faster than those in the surrounding environment. At a certain moment the pulse arrives at the point  $P_1$ , which lies on a line connecting the  $S_1$  and the center of the ring (see Fig. 15b). The arriving pulse creates an excitation on the ring originating from the point  $P_1$ . This excitation splits into a pair of pulses rotating in opposite directions and after propagating around the ring, they collide at the point, which is symmetric to  $P$  with respect to the center of the ring  $O$ . The point of collision can be located by an array of coincidence detectors and thus we have information on the wave vector of the arriving pulse. In this method the resolution depends on the number of detectors used because each of them corresponds to a certain range of the measured wave vectors. The fact that a pulse has appeared in a given output channel implies that no other channel of the sensor gets excited. It is interesting that the output information is reversed in space if compared with the input one; the left sensor channels are excited when an excitation arrives from the right and vice versa.

The geometry of the direction sensor that sends information as an excitation pulse in one of its detector channels can be yet simplified. Two such



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 14** (a) The distribution of excitable and non-excitable regions in a band filter. (b) Typical characteristics of the band filter (Gorecka and Gorecki 2003), The black dots mark periods for which the signal is not transmitted, the empty ones indicate full transmission, the empty diamonds mark the periods of the arriving signal for which every second spike is transmitted

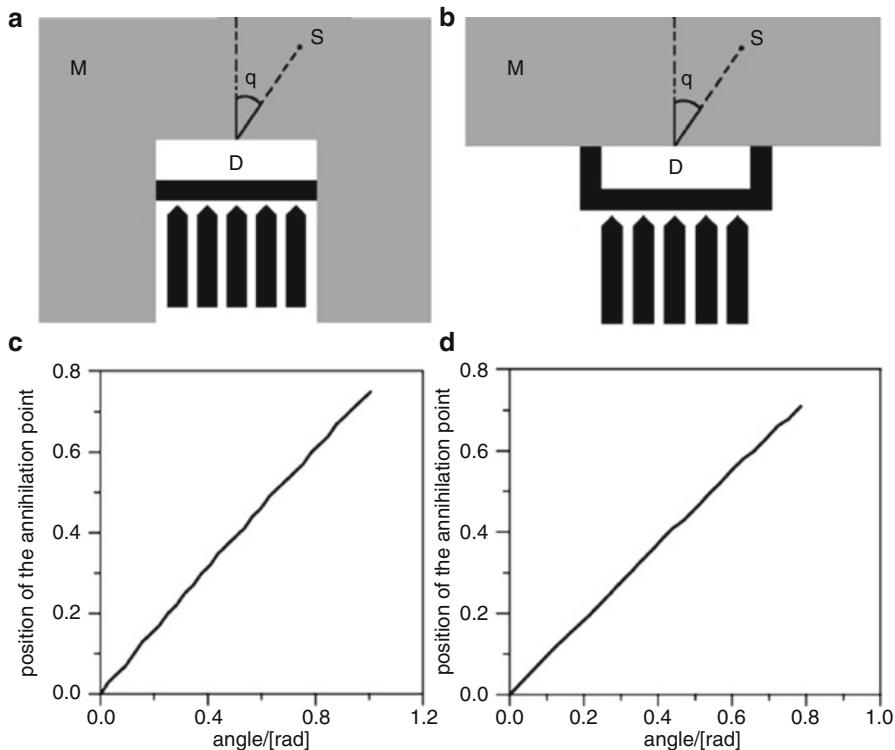


**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 15** The distribution of excitable areas (gray and black) and non-excitible regions (white) in the ring-shaped direction detector. Triangles marked  $X_1$ ,  $X_2$  and  $X_3$  show the output channels. (b, c) illustrate

time evolution of pulses generated from sources at different locations. The excited output channel depends on the direction of the source (Copied from Nagahara et al. (2004) with the permission of the authors)

realizations are illustrated in Fig. 16. The excitable areas that form the sensor are marked black, the excitable surrounding medium where stimuli propagate is gray, and the non-excitible areas are white. Let us consider an excitation generated in the medium M by a source S. It is obvious that if the source is located just above the sensor then pulses of excitation are generated at the ends of

the sensor channel D at the same time and they finally annihilate above the central coincidence channel and generate an output pulse in it. If the source is located at a certain angle with respect to the vertical line then the annihilation point is shifted off the center of D. We have performed a series of numerical simulations to find the relation between the position of the source and the



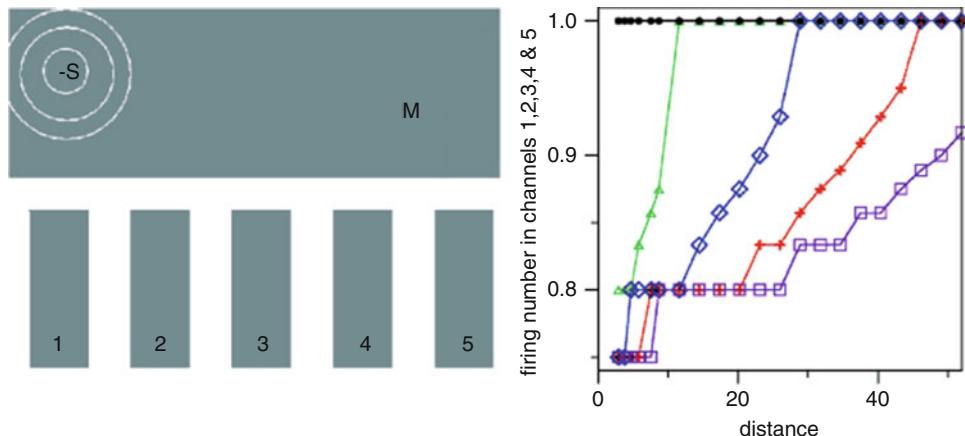
**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 16** The geometry of excitable (black) and diffusive (white) areas in two realizations of simplified distance sensors. The gray color marks the excitable medium around the sensor,  $S$  marks the position

position of annihilation point in  $D$ . In Fig. 16c, d the position of the annihilation point is plotted as a function of the angle between the source position and the vertical line. Although the constructions of sensors seem to be very similar, the working range of angles is larger for the sensor shown on Fig. 16b whereas the sensor shown in Fig. 16a offers slightly better resolution. Using information from two detectors of direction, the position of the source of excitation can be easily located because it is placed on the intersection of the lines representing the detected directions (Yoshikawa et al. 2009).

Another strategy of sensing is based on the observation that frequencies of pulses excited in a set of sensor channels by an external periodic perturbation contain information on the location of a stimulus. Within this strategy there is no direct relationship between the number of sensor channels and sensor resolution, and, as we show

of excitation source. (c, d) show the position of the annihilation point in the channel  $D$  as a function of the angle  $q$  between the source position and the vertical line. The half-length of  $D$  is used as the scale unit

below, a sensor with a relatively small number of channels can quite precisely estimate the distance separating it from the excitation source. As we have mentioned in the Introduction, the frequency of a chemical signal can change after propagating through a barrier made of non-excitatory medium. For a given frequency of arriving spikes the frequency of excitations behind the barrier depends on the barrier width and on the angle between the normal angle to the barrier and the wave vector of arriving pulses. This effect can be used for sensing. The geometrical arrangement of excitable and non-excitatory areas in a distance sensor is shown in Fig. 17. The excitable signal channels (in Fig. 17a they are numbered 1–5) are wide enough to ensure stable propagation of spikes. They are separated from one another by parallel non-excitatory gaps that do not allow for interference between pulses propagating in the neighboring



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 17** The distance sensor based on frequency transformation. (a) The distribution of excitable

channels, (b) the firing numbers in different channels as a function of distance; black, green, blue red and violet curves correspond to signals in channels 1–5, respectively

channels. The sensor channels are separated from the excitable medium  $M$  by the non-excitatory sensor gap  $G$ . The width of this gap is very important. If the gap is too wide then no excitation of the medium  $M$  can generate a pulse in the sensor channels. If the gap is narrow then any excitation in front of the sensor can pass  $G$  and create a spike in each sensor channel so the signals sent out by the sensor channels are identical. However, there is a range of gap widths such that the firing number depends on the wave vector characterizing a pulse at the gap in front of the channel. If the source  $S$  is close to the array of sensor channels, then the wave vectors characterizing excitations in front of various channels are significantly different. Thus the frequencies of excitations in various channels should differ too. On the other hand, if the source of excitations is far away from the gap  $G$  then the wave vectors in front of different channels should be almost identical and the frequencies of excitations should be the same. Therefore, the system illustrated in Fig. 17a can sense the distance separating it from the source of excitations. If this distance is small then the firing numbers in neighboring sensor channels are different and these differences decrease when the source of excitations moves away. A typical distance dependence of firing numbers observed in different channels is illustrated in Fig. 17b. This

result has been obtained in numerical simulations based on the Oregonator model.

The range of sensed distances depends on the number of sensor channels. A similar sensor, but with 4 sensor channels, was studied in (Gorecki et al. 2005). Comparing the results, we observe that the presence of 5th channel significantly improves the range of distances for which the sensor operates. On the other hand, the additional channel has almost no effect on sensor resolution at short distances. The sensor accuracy seems to be a complex function related to the width of the sensor gap and the properties of channels. The firing number of a single channel as a function of the distance between the sensor and the source has a devil-staircase-like form with long intervals where the function is constant corresponding to simple fractions as illustrated in Fig. 5b. In some range of distances the steps in firing numbers of different channels can coincide, so the resolution in this range of distances is poor. For the other regions, the firing numbers change rapidly and even small changes in distance can be easily detected.

The signal transformation on a barrier depends on the frequency of incoming pulses (c. f Fig. 6) so the distance sensor that works well for one frequency may not work for another. In order to function properly for different stimuli the sensor

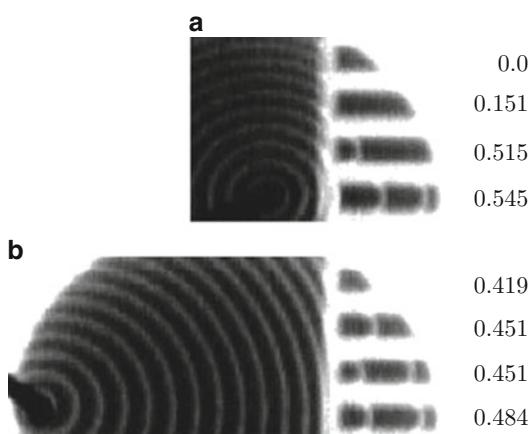
has to be adapted to the conditions it operates. In practice such adaptation can be realized by the comparison of frequencies of signals in detector channels with the frequency in a control channel. For example the control channel can be separated from the medium M by a barrier so narrow that every excitation of the medium generates a spike. The comparison between the frequency of excitations in the control channel and in the sensor channels can be used to adjust the sensor gap G. If the frequency of excitations in the sensor channels is the same as in the control channels then the width of the gap should be increased or its excitability level decreased. On the other hand if the frequency in the sensor channel is much smaller than in the control channel (or null) then the gap should be more narrow or more excitable. Such an adaptation mechanism allows one to adjust the distance detector to any frequency of arriving excitations.

The fact that the distance detector described above actually works has been confirmed in experiments with a photosensitive Ru-catalyzed BZ reaction. Typical snapshots from two experiments performed for the source placed 2 and 12 mm away from the sensors are shown in Fig. 18. The firing numbers observed in different

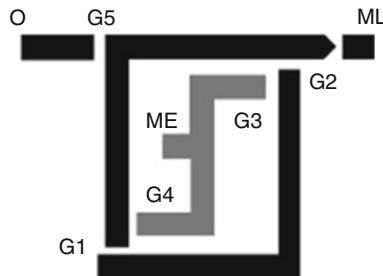
sensor channels confirm qualitatively the predictions of numerical simulations. If the source of excitations is close to the sensor gap, then the differences between firing numbers observed in neighboring channels are large. On the other hand, when the source of excitations is far away from the sensor, the frequencies in different channels become similar. The range of distances at which the sensor works is measured in centimeters so it is of the same order as the sensor size.

## The Ring Memory and Its Applications

The devices discussed in the previous section can be classified as instant machines (Rambidi and Maximychev 1997) capable of performing just the task they have been designed for. A memory where information coded in excitation pulses can be written-in, kept, read-out and, if necessary, erased, significantly increases the information processing potential of structured excitable media. Moreover, due to the fact that the state of memory can be changed by a spike, the memory allows for programming with excitation pulses. One possible realization of a chemical memory is based on the observation that a pulse of excitation can rotate on a ring-shaped excitable area as long as the reactants are supplied and the products removed (Lázár et al. 1995; Nagai et al. 2000; Noszticzius et al. 1987). Therefore, a ring with a number of spikes rotating on it can be regarded as a loaded memory cell. Such memory can be erased by counterpropagating pulses. The idea of memory with loading pulses rotating in one direction and erasing pulses in another has been discussed in (Motoike et al. 2001). If the ring is big then it can be used to memorize a large amount of information because it has many states corresponding to different numbers of rotating pulses. However, in such cases, loading the memory with subsequent pulses may not be reliable because the input can be blocked by the refractory tail left by one of already rotating pulses. The same effect can block the erasing pulses. Therefore, the memory capable of storing just a single bit seems to be more reliable and we consider it in this section. Such memory has two states: if there



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 18** Two snapshots from the experimental realization of the distance sensor with four channels. In the *upper* figure the source (1 mm thick silver wire) is placed 2 mm away from the sensor; in the *bottom* one the source is 12 mm away. The firing numbers are given next to the corresponding channels



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 19** The distribution of excitable channels (dark) that form a memory cell. The cell is composed of the loading channel ML, the memory ring, the erasing channel ME (marked gray) and the output channel O

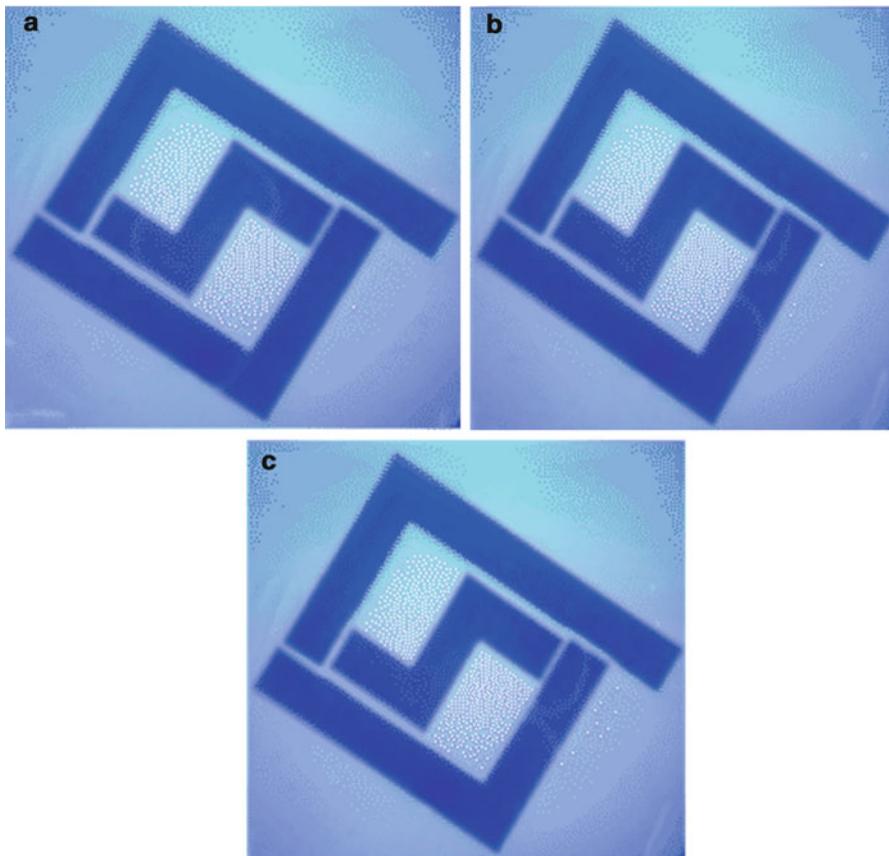
is a rotating pulse the ring is in the logical TRUE state (we call it loaded); if there is no pulse the state of memory corresponds to the logical FALSE and we call such memory erased.

Let us consider the memory illustrated in Fig. 19. The black areas define the memory ring, the output channel O, and the loading channel ML. The memory ring is formed by two L-shaped excitable areas. The areas are separated by gaps and, as we show below, with a special choice of the gaps the symmetry of the ring is broken and unidirectional rotation ensured. The Z-shaped excitable area composed of gray segments inside the ring forms the erasing channel. The widths of all non-exitable gaps separating excitable areas are selected such that a pulse of excitation propagating perpendicularly to the gap excites the active area on the other site of the gap, but the gap is impenetrable for pulses propagating parallel to the gap. The memory cell can be loaded by a spike arriving from the ML channel. Such a spike crosses the gap and generates an excitation on the ring rotating counterclockwise. The information about loaded memory is periodically sent out as a series of spikes through the output channel O. The rotating pulse does not affect the erasing channel because it always propagates parallel to it. The erasing excitation is generated in the center of the Z-shaped area and it splits into two erasing pulses. These pulses can cross the gaps separating the erasing channel from the memory ring and create a pair of pulses rotating clockwise.

A spike that propagates clockwise on the memory ring is not stable because it is not able to cross any of the gaps and dies. It also does not produce any output signal. Therefore, if the memory has not been loaded then an erasing excitation does not load it. On the other hand, if the memory is loaded then clockwise rotating pulses resulting from the excitation of the erasing channel annihilate with the loading pulse and the memory is erased. The idea of using two places where erasing pulses can enter the memory ring is used to ensure that at least one of those places is fully relaxed and so one of erasing pulses can always enter the ring.

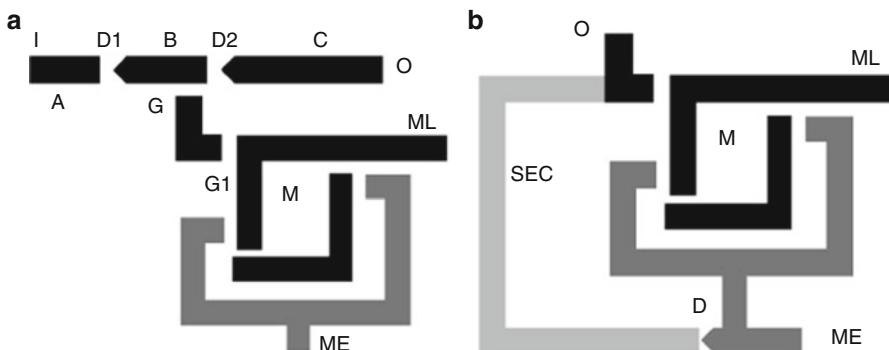
In order to verify that such memory works, we have performed a number of simulations using the Rovinsky-Zhabotinsky model of a BZ reaction and have done the experiments. We considered a loaded memory and at a random time the excitation was generated in the middle of the erasing channel. In all cases, such an excitation erased the memory ring. A typical experimental result is illustrated in Fig. 20. Here the channels are 1.5 mm thick and the gaps are 0.1 mm wide. Figure 20a shows the loaded memory and initiated pair of pulses in the erasing channel. In Fig. 20b one of the pulses from the erasing channel enters the memory ring. The memory ring in front of the left part of erasing channel is still in the refractory state so the left erasing pulse has not produced an excitation on the ring. Finally in Fig. 20c we observe the annihilation of the loading pulse with one of erasing pulses. Therefore, the state of memory changed from loaded to unloaded. The experiment was repeated a few times and the results were in a qualitative agreement with the simulations: the loaded memory cell kept its information for a few minutes and it was erased after every excitation of the erasing channel.

Figure 21 illustrates two simple, yet interesting, applications of a memory cell. Figure 21a shows a switchable unidirectional channel that can be opened or closed depending on the state of memory (Gorecki and Gorecka 2006). The channel is constructed with three excitable segments A, B and C separated by signals diodes D1 and D2 (c.f Fig. 4). The mid segment B is also linked with the output of the memory ring M. Here the erasing channels of M are placed outside the



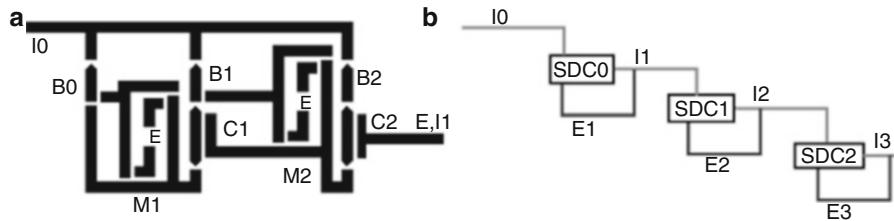
**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 20** Three snapshots from an experiment with memory erasing. The memory ring is

formed by two L-shaped excitable channels, the Z-shaped erasing channel is inside the ring



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 21** Two simple applications of a memory cell. (a) A switchable unidirectional channel that stops or transmits signals depending on the state of

memory. (b) A self-erasing memory cell that changes its state after a certain time. SEC marks the connection between the memory output and the erasing channel



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 22** The counter of excitation pulses that arrive at the input  $I_0$ . (a) Shows the geometry of excitable channels (black) in a single digit counter for the positional representation with the base 3. (b) Is a schematic

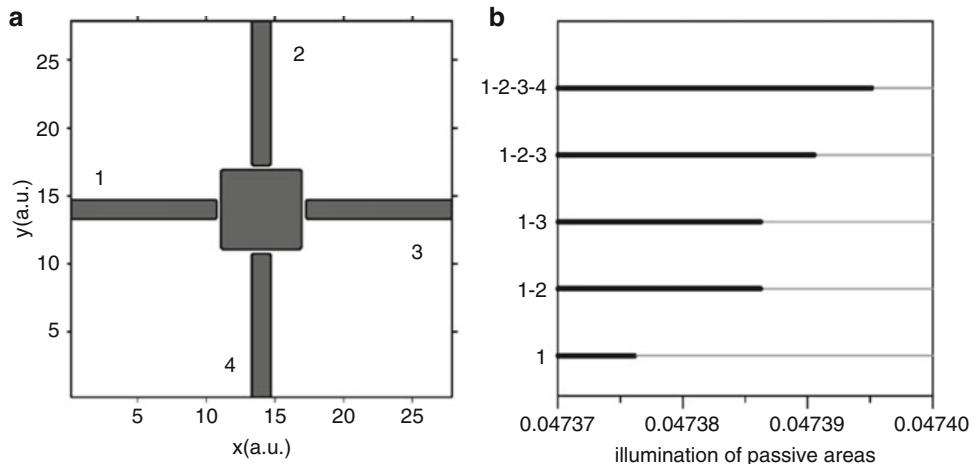
memory ring, but their function is exactly the same as in the memory illustrated in Fig. 19. The idea of switchable channel is similar to the construction of the NOT gate (Fig. 10). If the memory is not loaded then the spike propagation from input  $I$  to output  $O$  is unperturbed. However, if the memory is loaded then pulses of excitation periodically enter the segment  $B$  and annihilate with the transmitted signal. As a result, the channel is either open or blocked depending on the state of the memory. The excitations generated by the memory ring do not spread outside the  $B$  segment. On one end their propagation is stopped by the diode  $D_1$ , on the other by the geometry of the junction between the  $B$  channel and the memory output. The state of memory is controlled by the excitation pulses coming from loading and erasing channels, so switchable channels can be used in devices that are programmable with excitation pulses. Figure 21b illustrates a self-erasing memory cell that changes its state from loaded to erased after a certain time. In such a memory cell, the output channel is connected with the erasing one. When the memory is loaded the output signal appears. After some time decided by the length of connecting channels an output pulse returns as an erasing pulse and switches the memory to unloaded state. This behavior is an example of a simple feedback process common in self regulating information processing systems.

Using memory cells, signal diodes, and coincidence detectors, one can construct devices which perform more complex signal processing operations. As an example, we present a simple chemical

illustration of the cascade of single digit counters that provides a positional representation. The feedback signals from  $E_1$ ,  $E_2$  and  $E_3$  channels erase the memory of the single digit counters

realization of a device that counts arriving spikes and returns their number in any chosen positional representation (Gorecki et al. 2003). Such a counter can be assembled from single digit counters. The construction of a single digit counter depends on the representation used. Here, as an example, we consider the positional representation with the base 3. The geometry of a single digit counter is schematically shown in Fig. 22. Its main elements are two memory cells  $M_1$  and  $M_2$  and two coincidence detectors  $C_1$  and  $C_2$ . At the beginning let us assume that none of the memory cells are loaded.

When the first pulse arrives through the input channel  $I_0$ , it splits at all junctions and excitations enter segments  $B_0$ ,  $B_1$  and  $B_2$ . The pulse that has propagated through  $B_0$  loads the memory cell  $M_1$ . The pulses that have propagated through  $B_1$  and  $B_2$  die at the bottom diodes of segments  $C_1$  and  $C_2$  respectively. Thus, the first input pulse loads the memory  $M_1$  and does not change the state of  $M_2$ . When  $M_1$  is loaded, pulses of excitation are periodically sent to segments  $B_0$  and  $C_1$  via the bottom channel. Now let us consider what happens when the second pulse arrives. It does not pass through  $B_0$  because it annihilates with the pulses arriving from the memory  $M_1$ . The excitations generated by the second pulse can enter  $B_1$  and  $B_2$ . The excitation that propagated through  $B_2$  dies at the bottom diode of the segment  $C_2$ . The pulse that has propagated through  $B_1$  enters  $C_1$ , annihilates with a pulse from memory  $M_1$  and activates the coincidence detector. The output pulse from the coincidence detector loads the memory  $M_2$ . Therefore, after the second input pulse both memories  $M_1$  and  $M_2$  are loaded. If



**Computing in Geometrical Constrained Excitable Chemical Systems, Fig. 23** Artificial chemical neuron. (a) The geometry of excitable and non-excitatory areas; (b) The response of the neuron to different types of

the third pulse arrives the segments  $B_0$  and  $B_1$  are blocked by spikes sent from the memory rings. The generated excitation can enter channel  $B_2$  and its collision with a pulse coming from the memory cell  $M_2$  activates the output channel of  $C_2$ . The output signal is directed to the counter of responsible for the digit at next position ( $I_1$ ) and it is also used to erase all memory cells. Thus after the third pulse both memory cells  $M_1$  and  $M_2$  are erased. The counter shown in Fig. 22 returns a digit in a representation with the base 3: here 0 is represented by the  $(M_1, M_2) = (0, 0)$ , 1 by  $(1, 0)$ , 2 by  $(1, 1)$  and the next pulse changes the state of memory cell into  $(M_1, M_2) = (0, 0)$ . Of course, using  $n - 1$  memory cells in a single digit counter we can represent digits of the system with base  $n$ . A cascade of single digit counters (see Fig. 22b) gives a positional representation of the number of arriving pulses.

### Artificial Chemical Neurons with Excitable Medium

In this section we discuss a simple realization of an artificial neuron with structured excitable medium and show how neuron networks can be used in programmable devices. We consider a

excitations as a function of the illumination of non-excitatory regions. The numbers given on the left list the excited channels. The values of  $\phi_p$  for which the neuron body gets excited are marked by a thick line

chemical analogy of the McCulloch-Pitts neuron, i. e., a device that produces the output signal if the combined activation exceeds a critical value (Hertz et al. 1991). The geometry of a chemical neuron is inspired by the structure of biological neuron (Haken 2002). One of its realizations with an excitable medium is illustrated on Fig. 23. Another geometry of a neuron has been discussed in (Gorecka and Gorecki 2006). In an artificial chemical neuron, like in real neurons, dendrites (input channels 1–4) transmit weak signals which are added together through the processes of spatial and temporal integration inside the cell body (part C). If the aggregate excitation is larger than the threshold value the cell body gets excited. This excitation is transmitted as an output signal down the axon (the output channel) and the amplitude of the output signal does not depend on the value of integrated inputs but only on the properties of the medium that makes the output channel. In Fig. 23a the axon is not shown and we assume that it is formed by an excitable channel located perpendicularly above the cell body. In the construction discussed in (Gorecka and Gorecki 2006) both dendrites and the axon were on a single plane. We have studied the neuron using numerical simulations based on the Oregonator model and the reaction-diffusion equations have been solved on

a square grid. The square shape of the neuron body and the input channels shown on Fig. 23a allows for precise definition of the boundary between the excitable and non-excitabile parts. The idea behind the chemical neuron is similar to that of the AND gate: perturbations introduced by multiple inputs combine and generate a stronger excitation of the cell body than this resulting from a single input pulse. Therefore, it is intuitively clear that if we are able to adjust the amplitudes of excitations coming from individual input channels, then the output channel becomes excited only when the required number of excitations arrive from the inputs.

In the studied neuron, the amplitudes of spikes in input channels have been adjusted by subexcitability of these channels which can be controlled by the channel width or by the illumination of surrounding non-excitabile medium. In our simulations we considered different values of  $\phi_p$  for non-excitabile areas, whereas  $\phi_a = 0.007$  characterizing excitabile regions has been fixed for dendrites and the neuron body. Simulation results shown on Fig. 23b indicate that the properties of chemical neurons are very sensitive with respect to changes in  $\phi_p$ . The thick line marks the values of  $\phi_p$  for which the output signal appears. For the parameters used when  $\phi_p$  is slightly below 0.047377 any single excitation produces an output signal. On the other hand if  $\phi_p > 0.047397$  then even the combined excitation of all inputs is not sufficient to excite the neuron. In between those two limiting values we observe all other thresholds; i. e., an output excitation as the result of two or three combined inputs. Therefore, by applying the proper illumination level the structure shown in Fig. 23a can work as a four input McCulloch-Pitts neuron with the required threshold. A similar high sensitivity of the neuron properties on  $\phi_a$  is also observed. It means that neuron properties can be controlled with tiny changes in system parameters.

The chemical neuron illustrated in Fig. 23a can be used to program signal processing with pulses of excitation. If we set the illuminations such that any two input pulses produce the output and use one of the inputs to control the device, then if the control pulse is present the device performs the

OR operation on the other inputs. If there is no control pulse, it calculates an alternative on conjunctions of all pairs of channels.

The geometry of a network constructed with neurons can be easily controlled if the switchable channels illustrated on Fig. 21b are used to establish or cut connections between processing elements. The state of the memory that controls the channel may depend on the state of the network through a feedback mechanism what allows for network training. In the chemical programming described above pulses of concentration of the same reagent are used to store and process information, and to program the medium. Therefore, the output signal may be directly used to change the network geometry. External programming factors like illumination or a temperature field (Adamatzky 2005) are difficult for applications in three dimensional structures. The pulse based programming seems to be easier if the proper geometry of switchable channels and of the feedbacks is introduced.

At the first look it seems that a practical realization of a network built of chemical neurons may be difficult. However, the geometry of a considered neuron looks quite similar to structures of phases formed in a multicomponent system. For example, the diamond structure in an oil-water-surfactant system, which spontaneously appears at certain thermodynamic conditions (Babin and Ciach 2003), has a form of centers linked with the four nearest neighbors. If the reactants corresponding for excitability are soluble in water, but not in oil then the water rich phase forms the structure of excitabile channels and processing elements just as the result of thermodynamic conditions. Within a certain range of parameters, such a structure is thermodynamically stable. This means that the network has an auto-repair ability and robustness against unexpected destruction. The subexcitability of a channel is related to its diameter, so the required value can be obtained by selecting the right composition of the mixture and the conditions at which the phase transition occurs. Moreover, the structure is three dimensional, which allows for a higher density of processing elements than that obtained with the classical two-dimensional techniques, for example lithography.

## Perspectives and Conclusions

### Perspectives

In this article we have described a number of simple devices constructed with structured excitable chemical media which process information coded in excitation pulses. All the considered systems process information in an unconventional (non-von Neumann) way; i.e., without an external clock or synchronizing signal that controls the sequence of operations. On the other hand, in many cases the right timing of performed operation is hidden in the geometrical distribution and sizes of excitable regions. The described devices can be used as building blocks for more complex systems that process signals formed of excitation pulses. Some of the discussed devices can be controlled with spikes. Therefore, there is room for programming and learning. However, the further development of applications of structured excitable medium for information processing along this line seem to follow the evolution of classical electronic computing. In our opinion it would be more interesting to step away from this path and learn more about the potential offered by excitable media.

It would be interesting to study new types of excitable media suitable for information processing. Electrical analogs of reaction-diffusion systems (Unconventional Computing, Novel Hardware for) seem promising, because they are more robust than the wetware based on liquid BZ media. In such media, spikes propagate much faster and the spatial scale can be much reduced if compared with typical chemical systems. They seem to be promising candidates for the hardware in geometrically oriented problems and direct image processing (Adamatzky et al. 2005). However, two interesting properties of chemical reaction-diffusion systems are lost in their electrical analogs.

First, chemical information processing systems, unlike the electronic ones, integrate two functions: the chemical reactivity and the ability to process information. Having in mind the excitable oxidation of CO on a Pt surface and the potential application of this medium for information processing (Gorecka and Gorecki 2005), we can think of catalysts that are able to monitor their activity and

report it. Second, the media described in Unconventional Computing, Novel Hardware for are two-dimensional. Most of the systems discussed in this article can be realized in three dimensions. The use of two dimensional media in chemical experiments is mainly related with significant difficulties in observations of three dimensional chemical excitations (Luengviriya et al. 2006). Potential application of phase transitions for channel structure generation described in the previous section should increase the interest in information processing with three dimensional medium.

Studies on more effective methods of information coding are important for the further development of reaction-diffusion computing. The most simple translation of chemistry into the language of information science based on the equivalence between the presence of a spike and the TRUE logic value is certainly not the most efficient. It can be expected that information interpreted within multi-valued logic systems is more suitable for transformations with the use of structured media (Motoike and Adamatzky 2005). As an example, let us consider three-valued logic encoded by excitation pulses in the following way: the lack of a pulse represents a logical FALSE (F), two pulses separated by time  $\delta t$  correspond at the logical TRUE (T), and one pulse is interpreted as a “nonsense” ( $\star$ ). Let us assume that two signals coded as described above are directed to the inputs of the AND gate illustrated on Fig. 9 and that they are fully synchronized at the input. The gap between the input and output channels is adjusted such that an activator diffusing from a single traveling pulse is not sufficient to generate a new pulse in the output channel, but the excitation resulting from the collision of facing pulses at the junction point exceeds the threshold and the output pulse is generated. Moreover, let us assume that the system is described by the dynamics for which signals are transformed as illustrated in Fig. 7a (for example FitzHugh-Nagumo dynamics) and that the time difference between spikes  $\delta t$  is selected such that the second spike can pass the gap. The output signal appears when spikes from both input channels are in coincidence or when the second spike from the same input arrives. As a result the device performs

the following function within 3-valued logic (Motoike and Adamatzky 2005):

$\mathcal{F}_1$	$T$	$F$	$\star$
$T$	$T$	$\star$	$\star$
$F$	$\star$	$F$	$F$
$\star$	$\star$	$F$	$\star$

The same operation, if performed on a classical computer would require a procedure that measures time between spikes. The structured excitable medium performs it naturally provided that the time  $\delta t$  is adjusted with the properties of the medium and the geometry of the gap. Of course, similar devices that process variables of n-valued logic can be also constructed with structured excitable media.

## Conclusions

In the article we have presented a number of examples that should convince the reader that structured excitable media can be used for information processing. The future research will verify if this branch of computer science is fruitful. It would be important to find new algorithms that can be efficiently executed using a structured excitable medium. However, the ultimate test for the usefulness of ideas should come from biology. It is commonly accepted that excitable behavior is responsible for information processing and coding in living organisms (Agmon-Snir et al. 1998; Häusser et al. 2000; Kindzelskii and Petty 2003; Rambidi 2005). We believe that studies on chemical information processing will help us to better understand these problems. And, although at the moment computing with a homogeneous excitable medium seems to offer more applications than that with the structured one, we believe the proportions will be reversed in the future. After all, our brains are not made of a single piece of a homogeneous excitable medium.

**Acknowledgments** The research on information processing with structured excitable medium has been supported by the Polish State Committee for Scientific Research project 1 P03B 035 27.

## Bibliography

### Primary Literature

- Adamatzky A (2004) Collision-based computing in Belousov-Zhabotinsky medium. *Chaos Soliton Fractal* 21(5):1259–1264
- Adamatzky A (2005) Programming reaction-diffusion processors. In: Banatre JP, Fradet P, Giavitto JL, Michel O (eds) LNCS, vol 3566. Springer, pp 47–55
- Adamatzky A, De Lacy Costello B (2002) Experimental logical gates in a reaction-diffusion medium: the XOR gate and beyond. *Phys Rev E* 66:046112
- Adamatzky A, De Lacy CB, Asai T (2005) Reaction-diffusion computers. Elsevier, Amsterdam
- Agladze K, Aliev RR, Yamaguchi T, Yoshikawa K (1996) Chemical diode. *J Phys Chem* 100:13895–13897
- Agladze K, Magome N, Aliev R, Yamaguchi T, Yoshikawa K (1997) Finding the optimal path with the aid of chemical wave. *Phys D* 106:247–254
- Agladze K, Tóth Á, Ichino T, Yoshikawa K (2000) Propagation of chemical pulses at the boundary of excitable and inhibitory fields. *J Phys Chem A* 104: 6677–6680
- Agmon-Snir H, Carr CE, Rinzel J (1998) The role of dendrites in auditory coincidence detection. *Nature* 393:268–272
- Amemiya T, Ohmori T, Yamaguchi T (2000) An Oregonator-class model for photoinduced behavior in the  $\text{Ru}(\text{bpy})_3^{2+}$ -Catalyzed Belousov-Zhabotinsky reaction. *J Phys Chem A* 104:336–344
- Armstrong GR, Taylor AF, Scott SK, Gaspar V (2004) Modelling wave propagation across a series of gaps. *Phys Chem Chem Phys* 6:4677–4681
- Babin V, Ciach A (2003) Response of the bicontinuous cubic D phase in amphiphilic systems to compression or expansion. *J Chem Phys* 119:6217–6231
- Bertram M, Mikhailov AS (2003) Pattern formation on the edge of chaos: mathematical modeling of CO oxidation on a Pt(110) surface under global delayed feedback. *Phys Rev E* 67:036207
- Dolnik M, Marek M (1991) Phase excitation curves in the model of forced excitable reaction system. *J Phys Chem* 95:7267–7272
- Dolnik M, Finkeova I, Schreiber I, Marek M (1989) Dynamics of forced excitable and oscillatory chemical-reaction systems. *J Phys Chem* 93:2764–2774
- Dolnik M, Marek M, Epstein IR (1992) Resonances in periodically forced excitable systems. *J Phys Chem* 96:3218–3224
- Epstein IR, Showalter K (1996) Nonlinear chemical dynamics: oscillations, patterns, and chaos. *J Phys Chem* 100:13132–13147
- Feynman RP, Allen RW, Heywould T (2000) Feynman lectures on computation. Perseus Books, New York
- Field RJ, Noyes RM (1974) Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction. *J Chem Phys* 60:1877–1884
- Field RJ, Koros E, Noyes RM (1972) Oscillations in chemical systems. II. Thorough analysis of temporal

- oscillation in the bromate-cerium-malonic acid system. *J Am Chem Soc* 94:8649–8664
- Finkeova I, Dolnik M, Hrudka B, Marek M (1990) Excitable chemical reaction systems in a continuous stirred tank reactor. *J Phys Chem* 94:4110–4115
- FitzHugh R (1960) Thresholds and plateaus in the Hodgkin-Huxley nerve equations. *J Gen Physiol* 43: 867–896
- FitzHugh R (1961) Impulses and physiological states in theoretical models of nerve membrane. *Biophys J* 1: 445–466
- Gaspar V, Bazsa G, Beck MT (1983) The influence of visible light on the Belousov-Zhabotinskii oscillating reactions applying different catalysts. *Z Phys Chem (Leipzig)* 264:43–48
- Ginn BT, Steinbock B, Kahveci M, Steinbock O (2004) Microfluidic systems for the Belousov-Zhabotinsky reaction. *J Phys Chem A* 108:1325–1332
- Gorecka J, Gorecki J (2003) T-shaped coincidence detector as a band filter of chemical signal frequency. *Phys Rev E* 67:067203
- Gorecka J, Gorecki J (2005) On one dimensional chemical diode and frequency generator constructed with an excitable surface reaction. *Phys Chem Chem Phys* 7:2915–2920
- Gorecka J, Gorecki J (2006) Multiargument logical operations performed with excitable chemical medium. *J Chem Phys* 124:084101. (1–5)
- Gorecka J, Gorecki J, Igarashi Y (2007) One dimensional chemical signal diode constructed with two non-excitable barriers. *J Phys Chem A* 111:885–889
- Gorecki J, Gorecka JN (2006) Information processing with chemical excitations – from instant machines to an artificial chemical brain. *Int J Unconv Comput* 2:321–336
- Gorecki J, Kawczynski AL (1996) Molecular dynamics simulations of a thermochemical system in bistable and excitable regimes. *J Phys Chem* 100:19371–19379
- Gorecki J, Yoshikawa K, Igarashi Y (2003) On chemical reactors that can count. *J Phys Chem A* 107:1664–1669
- Gorecki J, Gorecka JN, Yoshikawa K, Igarashi Y, Nagahara H (2005) Sensing the distance to a source of periodic oscillations in a nonlinear chemical medium with the output information coded in frequency of excitation pulses. *Phys Rev E* 72:046201. (1–7)
- Haken H (2002) Brain dynamics. In: Springer series in synergetics. Springer, Berlin
- Häusser M, Spruston N, Stuart GJ (2000) Diversity and dynamics of dendritic signaling. *Science* 290:739–744
- Hertz J, Krogh A, Palmer RG (1991) Introduction to the theory of neural computation. Addison-Wesley, Redwood City
- Ichino T, Igarashi Y, Motoike IN, Yoshikawa K (2003) Different operations on a single circuit: field computation on an excitable chemical system. *J Chem Phys* 118:8185–8190
- Igarashi Y, Gorecki J, Gorecka JN (2006) Chemical information processing devices constructed using a nonlinear medium with controlled excitability. *Lect Note Comput Sci* 4135:130–138
- Kapral R, Showalter K (1995) Chemical waves and patterns. Kluwer, Dordrecht
- Kindzelskii AL, Petty HR (2003) Intracellular calcium waves accompany neutrophil polarization, formylmethionylleucylphenylalanine stimulation, and phagocytosis: a high speed microscopy study. *J Immunol* 170:64–72
- Krischer K, Eiswirth M, Ertl GJ (1992) Oscillatory CO oxidation on Pt(110): modelling of temporal self-organization. *J Chem Phys* 96:9161–9172
- Krug HJ, Pohlmann L, Kuhnert L (1990) Analysis of the modified complete Oregonator accounting for oxygen sensitivity and photosensitivity of Belousov-Zhabotinskii systems. *J Phys Chem* 94:4862–4866
- Kuramoto Y (1984) Chemical oscillations, waves, and turbulence. Springer, Berlin
- Kusumi T, Yamaguchi T, Aliev RR, Amemiya T, Ohmori T, Hashimoto H, Yoshikawa K (1997) Numerical study on time delay for chemical wave transmission via an inactive gap. *Chem Phys Lett* 271:355–360
- Lázár A, Noszticzius Z, Försterling H-D, Nagy-Ungvári Z (1995) Chemical pulses in modified membranes I. Developing the technique. *Physica D* 84:112–119
- Luengviriya C, Storb U, Hauser MJB, Müller SC (2006) An elegant method to study an isolated spiral wave in a thin layer of a batch Belousov-Zhabotinsky reaction under oxygen-free conditions. *Phys Chem Chem Phys* 8:1425–1429
- Manz N, Müller SC, Steinbock O (2000) Anomalous dispersion of chemical waves in a homogeneously catalyzed reaction system. *J Phys Chem A* 104:5895–5897
- Maselko J, Reckley JS, Showalter K (1989) Regular and irregular spatial patterns in an immobilized-catalyst Belousov-Zhabotinsky reaction. *J Phys Chem* 93:2774–2780
- Mikhailov AS, Showalter K (2006) Control of waves, patterns and turbulence in chemical systems. *Phys Rep* 425:79–194
- Morozov VG, Davydov NV, Davydov VA (1999) Propagation of curved activation fronts in anisotropic excitable media. *J Biol Phys* 25:87–100
- Motoike IN, Adamatzky A (2005) Three-valued logic gates in reaction-diffusion excitable media. *Chaos, Solitons Fractals* 24:107–114
- Motoike I, Yoshikawa K (1999) Information operations with an excitable field. *Phys Rev E* 59:5354–5360
- Motoike IN, Yoshikawa K (2003) Information operations with multiple pulses on an excitable field. *Chaos, Solitons Fractals* 17:455–461
- Motoike IN, Yoshikawa K, Iguchi Y, Nakata S (2001) Real-time memory on an excitable field. *Phys Rev E* 63:036220. (1–4)
- Murray JD (1989) Mathematical biology. Springer, Berlin
- Nagahara H, Ichino T, Yoshikawa K (2004) Direction detector on an excitable field: field computation with coincidence detection. *Phys Rev E* 70:036221. (1–5)

- Nagai Y, Gonzalez H, Shrier A, Glass L (2000) Paroxysmal starting and stopping of circulatory pulses in excitable media. *Phys Rev Lett* 84:4248–4251
- Nagumo J, Arimoto S, Yoshizawa S (1962) An active pulse transmission line simulating nerve axon. *Proc IRE* 50:2061–2070
- Noszticzius Z, Horsthemke W, McCormick WD, Swinney HL, Tam WY (1987) Sustained chemical pulses in an annular gel reactor: a chemical pinwheel. *Nature* 329:619–620
- Plaza F, Velarde MG, Arecchi FT, Boccaletti S, Ciofini M, Meucci R (1997) Excitability following an avalanche-collapse process. *Europhys Lett* 38:85–90
- Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007) Numerical recipes, 3rd edn. The art of scientific computing. Available via <http://www.nr.com>
- Rambidi NG (2005) Biologically inspired information processing technologies: reaction-diffusion paradigm. *Int J Unconv Comp* 1:101–121
- Rambidi NG, Maximychev AV (1997) Towards a biomolecular computer. Information processing capabilities of biomolecular nonlinear dynamic media. *BioSyst* 41:195–211
- Rambidi NG, Yakovenchuk D (1999) Finding paths in a labyrinth based on reaction-diffusion media. *BioSyst* 51:67–72
- Rambidi NG, Yakovenchuk D (2001) Chemical reaction-diffusion implementation of finding the shortest paths in a labyrinth. *Phys Rev E* 63:026607
- Rovinsky AB (1986) Spiral waves in a model of the ferroin catalyzed Belousov-Zhabotinskii reaction. *J Phys Chem* 90:217–219
- Rovinsky AB, Zhabotinsky AM (1984) Mechanism and mathematical model of the oscillating bromate-ferroin-bromomalonic acid reaction. *J Phys Chem* 88:6081–6084
- Sielewiesiuk J, Gorecki J (2001a) Chemical impulses in the perpendicular junction of two channels. *Acta Phys Pol B* 32:1589–1603
- Sielewiesiuk J, Gorecki J (2001b) Logical functions of a cross junction of excitable chemical media. *J Phys Chem A* 105:8189–8195
- Sielewiesiuk J, Gorecki J (2002a) Chemical waves in an excitable medium: their features and possible applications in information processing. In: Klonowski W (ed) Attractors, signals and synergetics. 1 st European interdisciplinary school on nonlinear dynamics for system and signal analysis euroattractor 2000, Warsaw, 6–15 June 2000. Pabst, Lengerich, pp 448–460
- Sielewiesiuk J, Gorecki J (2002b) On complex transformations of chemical signals passing through a passive barrier. *Phys Rev E* 66:016212
- Sielewiesiuk J, Gorecki J (2002c) Passive barrier as a transformer of chemical signal frequency. *J Phys Chem A* 106:4068–4076
- Steinbock O (2002) Excitable front geometry in reaction-diffusion systems with anomalous dispersion. *Phys Rev Lett* 88:228302
- Steinbock O, Kettunen P (1996) Chemical clocks on the basis of rotating pulses. Measuring irrational numbers from period ratios. *Chem Phys Lett* 251:305–308
- Steinbock O, Toth A, Showalter K (1995a) Navigating complex labyrinths – optimal paths from chemical waves. *Science* 267:868–871
- Steinbock O, Kettunen P, Showalter K (1995b) Anisotropy and spiral organizing centers in patterned excitable media. *Science* 269:1857–1860
- Steinbock O, Kettunen P, Showalter K (1996) Chemical pulse logic gates. *J Phys Chem* 100:18970–18975
- Suzuki R (1967) Mathematical analysis and application of iron-wire neuron model. *IEEE Trans Biomed Eng* 14:114–124
- Suzuki K, Yoshinobu T, Iwasaki H (1999) Anisotropic waves propagating on two-dimensional arrays of Belousov-Zhabotinsky oscillators. *Jpn J Appl Phys* 38:L345–L348
- Suzuki K, Yoshinobu T, Iwasaki H (2000) Unidirectional propagation of chemical waves through microgaps between zones with different excitability. *J Phys Chem A* 104:6602–6608
- Taylor AF, Armstrong GR, Goodchild N, Scott SK (2003) Propagation of chemical waves across inexcitable gaps. *Phys Chem Chem Phys* 5:3928–3932
- Toth A, Showalter K (1995) Logic gates in excitable media. *J Chem Phys* 103:2058–2066
- Toth A, Horvath D, Yoshikawa K (2001) Unidirectional wave propagation in one spatial dimension. *Chem Phys Lett* 345:471–474
- Volford A, Simon PL, Farkas H, Noszticzius Z (1999) Rotating chemical waves: theory and experiments. *Physica A* 274:30–49
- Yoshikawa K, Nagahara H, Ichino T, Gorecki J, Gorecka JN, Igarashi Y (2009) On chemical methods of direction and distance sensing. *Int J Unconv Comput* 5:53–65

## Books and Reviews

- Hjelmfelt A, Ross J (1994) Pattern recognition, chaos, and multiplicity in neural networks of excitable systems. *Proc Natl Acad Sci U S A* 91:63–67
- Nakata S (2003) Chemical analysis based on nonlinearity. Nova, New York
- Rambidi NG (1998) Neural network devices based on reaction-diffusion media: an approach to artificial retina. *Supramol Sci* 5:765–767
- Storb U, Müller SC (2004) Scroll waves. In: Scott A (ed) Encyclopedia of nonlinear sciences. Routledge, Taylor and Francis Group, New York, pp 825–827



---

## Novel Hardware for Unconventional Computing

Tetsuya Asai  
Hokkaido University, Sapporo, Japan

### Article Outline

Glossary  
Definition of the Subject  
Introduction  
Constructing Electrical Analog of Reaction-Diffusion Systems  
Digital CMOS Reaction-Diffusion Chips  
Analog CMOS Reaction-Diffusion Chip  
Reaction-Diffusion Computing Devices Based on Minority-Carrier Transport in Semiconductors  
Single-Electron Reaction-Diffusion System  
Expanding Circular Pattern  
Rotating Spiral Pattern  
Dividing-and-Multiplying Pattern  
Collision-Based RD Computers  
Collision-Based Reaction-Diffusion Computing for Digital LSIs  
Future Directions  
Bibliography

### Glossary

**Analog circuit** An electronic circuit that operates with currents and voltages that vary continuously with time and have no abrupt transitions between levels. Since most physical quantities, e.g., velocity and temperature, vary continuously, as does audio, an analog circuit provides the best means of representing them.

**Current mirror** A circuit that copies single input current to single (or multiple) output

nodes. Two types of current mirrors exist: nMOS for current sinks and pMOS for current sources. Combining both types of current mirrors, one can invert a direction of currents, e.g., sink to source or source to sink.

**Digital circuit** An electronic circuit that can take on only a finite number of states. Binary (two-state) digital circuits are the most common. The two possible states of a binary circuit are represented by the binary digits, or bits, 0 and 1. The simplest forms of digital circuits are built from logic gates, the building blocks of the digital computer.

**Diode** A device that allows current flow only in one direction. Chemical diode allows for propagation of chemical waves only in one direction.

**Flip-flop circuit** A synchronous bistable device where the output changes state only when the clock input is triggered. That is, changes in the output occur in synchronization with the clock.

**Floating-gate transistor** A device consisting of a control gate, floating gate, and the thin oxide layer; when floating gate is given an electrical charge, the charge is trapped in the insulating thin oxide layer. The transistors are used as nonvolatile storage devices because they store electrical charge for a long time without powering.

**LSI, large-scale integrated circuit** An electronic circuit built on a semiconductor substrate, usually one of single-crystal silicon. It contains from 100 to 1,000 transistors. Some LSI circuits are analog devices; an operational amplifier is an example. Other LSI circuits, such as the microprocessors used in computers, are digital devices.

**Minority-carrier transport** A physical phenomenon in forwardly biased semiconductor *p-n* junctions. Minority carriers are generated in both areas of *p*- and *n*-type semiconductors. For *p*-type semiconductors, the minority carriers are electrons, while they are holes in *n*-type semiconductors. Once minority carriers are generated, they diffuse among the

semiconductor and finally disappear by the recombination of electrons and holes.

**nMOS FET** Abbreviation of *n*-type metal-oxide semiconductor field-effect transistor, where semiconductor is negatively charged so the transistors are controlled by movement of electrons; these transistors have three modes of operation: cutoff, triode, and saturation (active).

**pMOS FET** A device which works by analogy to nMOS FET, but the transistors are moved on and off by movement of electron vacancies.

**Single-electron circuit** An electrical circuit that is functionally constructed by controlling movements of single electrons. Single-electron circuit consists of tunneling junctions, and electrons are controlled by using physical phenomena called the Coulomb blockade.

## Definition of the Subject

Natural systems give us examples of amorphous, unstructured devices, capable of fault-tolerant information processing, particularly with regard to the massive parallel spatial problems that digital processors have difficulty with. For example, reaction-diffusion (RD) chemical systems have the unique ability to efficiently solve combinatorial problems with natural parallelism (Adamatzky 2001). In liquid-phase parallel RD processors (RD chemical computers), both the data and the results of the computation are encoded as concentration profiles of the reagents. The computation is performed via the spreading and interaction of the wave fronts. In experimental chemical processors, data are represented by local disturbances in the concentrations, and computation is accomplished via the interaction of waves caused by the local disturbances.

The RD chemical computers operate in parallel since the chemical medium's micro-volumes update their states simultaneously, and the molecules diffuse and react in parallel. We see a similar parallelism in cellular automata (CA). Various RD systems can be modeled in terms of CA, including the Belousov-Zhabotinsky (BZ) reaction (Adamatzky 2001; Gerhardt et al. 1990), the Turing system (Young 1984), a precipitating BZ

system for computation of Voronoi diagram (Adamatzky 1994, 1996), and so on. A two-dimensional CA is particularly well suited for the coming generation of massively parallel machines, in which a very large number of separate processors act in parallel. If an elemental processor in the CA is constructed from a smart processor and photosensor, various CA algorithms can easily be used to develop intelligent image sensors.

Implementing RD systems in hardware has several advantages. Hardware RD systems are very useful in simulating RD phenomena, even if the phenomena never occur in nature. This implies that a hardware system is a possible candidate for developing an artificial RD system that is superior to a natural system. For instance, hardware RD systems can operate at much faster speeds than actual RD systems. The velocity of chemical waves in a BZ reaction is  $O(10^{-2})$  m/s (Tóth et al. 1994), while that of a hardware RD system will be over a million times faster than that of the BZ reaction, independent of system size. This property is useful for developers of RD applications because every RD application benefits from high-speed operations. These properties encouraged us to develop novel hardware for unconventional (RD-based) computing.

## Introduction

This entry presents an overview of the semiconductor implementation of reaction-diffusion (RD) computers in large-scale integrated (LSI) circuits for unconventional computing. There, we see how to model RD processes in LSI circuits and discuss several designs of RD digital chips, based on cellular automaton models of RD and excitable systems. Feasibility of an RD digital chip is demonstrated in the construction of a Voronoi diagram and decomposition of images. The entry concludes with analog RD chips, where closer to physical reality nonlinear characteristics of chemical systems are employed. We see designs of RD chips based on Oregonator, Turing, and so on. Moreover, functionality of analog RD chips in feature extraction and finger-print reconstruction tasks is exemplified.

A RD chip consists of (i) reaction circuits that emulate elementary interactions between neurons (or chemical substances) and (ii) diffusion devices that imitate synapses (or chemical diffusion of the substances). RD chips were mostly designed by digital, analog, or mixed-signal complementary metal-oxide-semiconductor (CMOS) circuits of cellular neural networks (CNNs) or cellular automata (CA). Electrical cell circuits were designed to implement several CA and CNN models of RD systems (Adamatzky et al. 2004; Asai et al. 2002, 2005b; Matsubara et al. 2004), as well as fundamental RD equations (Asai et al. 2004, 2005c; Daikoku et al. 2002; Karahaliloglu and Balkir 2005; Serrano-Gotarredona and Linares-Barranco 2003). Each cell is arranged on a 2D square or hexagonal grid and is connected with adjacent cells through coupling devices that transmit a cell's state to its neighboring cells, as in conventional CAs. For instance, an analog-digital hybrid RD chip (Asai et al. 2002) was designed for emulating a conventional CA model for BZ reactions (Gerhardt et al. 1990). A precipitating BZ system for computation of Voronoi diagram (Adamatzky 1994, 1996) was also implemented on an analog-digital hybrid RD chip (Asai et al. 2005b). A full-digital RD processor (Matsubara et al. 2004) was also designed on the basis of a multiple-valued CA model, called *excitable lattices* (Adamatzky 2001). Furthermore, a RD CA processor for complex image processing has been proposed (Asai et al. 2005a). It performs quadrilateral-object extraction based on serial and parallel CA algorithms. An analog cell circuit was also designed to be equivalent to spatial-discrete Turing RD systems (Daikoku et al. 2002). A full-analog RD chip that emulates BZ reactions has also been designed and fabricated (Asai et al. 2005c). Furthermore, blueprints of non-CMOS RD chips have been designed, i.e., an RD device based on minority-carrier transport in semiconductor devices (Asai et al. 2004). In the following sections, we see how to construct an artificial RD system on *solid-state media* and to develop some applications using the solid-state RD system that could cope with conventional digital computers.

## Constructing Electrical Analog of Reaction-Diffusion Systems

The behavior of RD systems, or the spatiotemporal patterns of chemical concentration, can be expressed by the reaction-diffusion equation, a partial differential equation with chemical concentrations as variables:

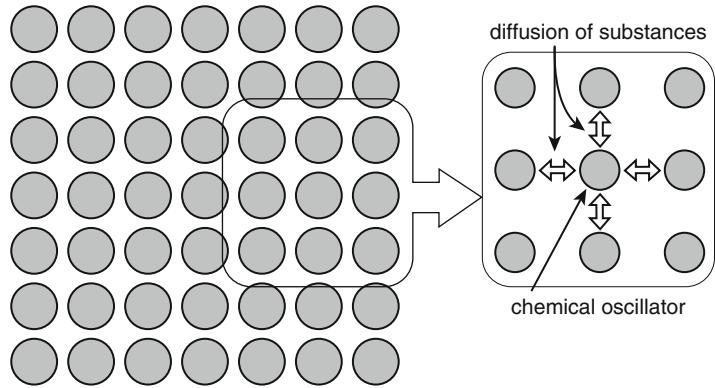
$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f}(\mathbf{u}) + D\Delta\mathbf{u}, [\mathbf{u} = (u_1, u_2, u_3, \dots)], \quad (1)$$

where  $t$  is time,  $\mathbf{u}$  is the vector of chemical concentrations,  $u_i$  is the concentration of the  $i$ th substance, and  $D$  is the diagonal matrix of diffusion coefficients. Nonlinear function  $\mathbf{f}(\mathbf{u})$  is the reaction term that represents the reaction kinetics of the system. Spatial derivative  $D\Delta\mathbf{u}$  is the diffusion term that represents the change of  $\mathbf{u}$  due to the diffusion of the substance. A greater number of variables results in more complex dynamics and a more complicated dissipative structure. A simple reaction-diffusion system with few variables, however, will still exhibit dynamics similar to biological activity.

An RD system can be considered an aggregate of coupled chemical oscillators, or a chemical cellular automaton, as described in Fig. 1. Each oscillator represents the local reaction of chemical substances and generates nonlinear dynamics  $d\mathbf{u}/dt = \mathbf{f}(\mathbf{u})$  that corresponds to reaction kinetics in Eq. 1. The oscillator interacts with its neighbors through nonlocal diffusion of substances; this corresponds to the diffusion term in Eq. 1 and produces dynamics  $d\mathbf{u}/dt = D\Delta\mathbf{u}$ . Because of diffusion, all oscillators correlate with one another to generate synchronization and entrainment. Consequently, the system as a whole produces orderly dissipative structures on a macroscopic level. The size of each oscillator, or the size of the local space in which chemical concentrations are roughly uniform, depends on the diffusion coefficients and reaction velocities in the system. It is several micrometers in diameter in many liquid RD systems; therefore, even a tiny RD system in a test tube contains millions of oscillators.

### Novel Hardware for Unconventional Computing,

**Fig. 1** Simplified model of RD systems, consisting of many chemical oscillators. Each oscillator has variables corresponding to chemical concentrations  $u_1, u_2, u_3, \dots$  in Eq. 1 and interacts with its neighbors through diffusion of substances



An electrical analog of RD systems can be created by using electrical oscillation circuits instead of chemical oscillators and coupling these circuits with one another in a way that imitates diffusion. Variables are the electrical potential of nodes in the oscillation circuits in this electrical RD system. The system will produce electrical dissipative structures, i.e., orderly spatiotemporal patterns of node potentials, under appropriate conditions.

The key to building an electrical RD system is to integrate a large number of oscillation circuits on a chip with coupling subcircuits. A large arrangement of oscillators (e.g.,  $1,000 \times 1,000$  or more) is needed to generate complex, varied dissipative structures as observed in chemical RD systems. Oscillators constructed by micro- or nanoscale circuits are thus useful to achieve such large-scale integration. Such circuits can generate nonlinear oscillation through a simple circuit structure, so it can effectively be used in producing small oscillators for electrical RD systems.

## Digital CMOS Reaction-Diffusion Chips

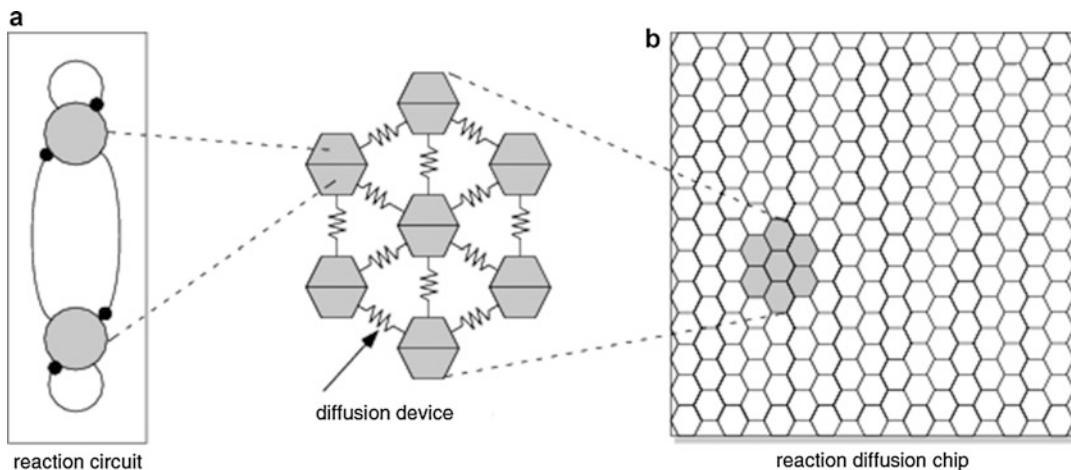
### A Reaction-Diffusion Circuit Based on Cellular Automaton Processing Emulating the Belousov-Zhabotinsky Reaction

The Belousov-Zhabotinsky (BZ) reaction provides us important clues to control 2D phase-lagged stable synchronous patterns in excitable medium. Because of the difficulty in computing RD systems in large systems using conventional digital processors, a cellular automaton

(CA) circuit that emulates the BZ reaction was proposed (Asai et al. 2002). In the circuit, a two-dimensional array of parallel processing cells, shown in Fig. 2, is responsible for the emulation, and its operation rate is independent of the system size. The operations of the CA circuit were demonstrated by using a simulation program with integrated circuit emphasis (SPICE). In the circuit's initial state, cells adjacent to inactive cells were in a refractory period (step 0 in Fig. 3). The inactive cells adjacent to the white bar in Fig. 3 were suppressed by adjacent cells in the refractory period (cells in the white bar). The inactive cells then entered an active, inactive, or refractory period, depending on the degree of the refractory condition. When the inactive cells were in an active or inactive period, the tip of the bar rotated inward (step 4–8 in Fig. 3), resulting in the generation of the modelock (spiral patterns) typically observed in the BZ reaction (step 40 in Fig. 3). A hexagonal distortion of the propagating waves was generated by interactions between adjacent cells. These results indicated that the RD chip could be easily integrated into existing digital systems and can be used to clarify RD systems, aiming at developing further novel applications.

### Reaction-Diffusion Chip Implementing Excitable Lattices with Multiple-Valued Cellular Automata

A RD chip was fabricated based on a multiple-valued CA model of excitable lattices (Matsubara et al. 2004). The experiments confirmed the expected operations, i.e., excitable wave



**Novel Hardware for Unconventional Computing, Fig. 2** Basic construction of RD chip

propagation and annihilation. One could obtain a binary stream from the common output wire by selecting each cell sequentially. Using a conventional displaying technique, the binary stream was reconstructed on a 2D display. Figure 4 shows snapshots taken from the recorded movie. Each dot represents an excitatory cell where EXC is logical “1.” In the experiment, the supply voltage was set at 5 V, and the system clock was set at low frequency (2.5 Hz) so that “very slow” spatiotemporal activities could be observed visually (the low frequency was used only for the visualization and was not the upper limit of the circuit operation). Pinspot lights were applied to several cells at top-left and bottom-right corners of the chip. The circuit exhibited the expected results; i.e., two excitable waves of excited cells triggered by the corner cells propagated toward the center and disappeared when they collided. This result suggests that if we use a more microscopic process and a large number of cells were implemented, we would observe the same complex (BZ-like) patterns, as observed in the original excitable lattices (Adamatzky 2001).

#### Silicon Implementation of a Chemical Reaction-Diffusion Processor for Computation of Voronoi Diagram

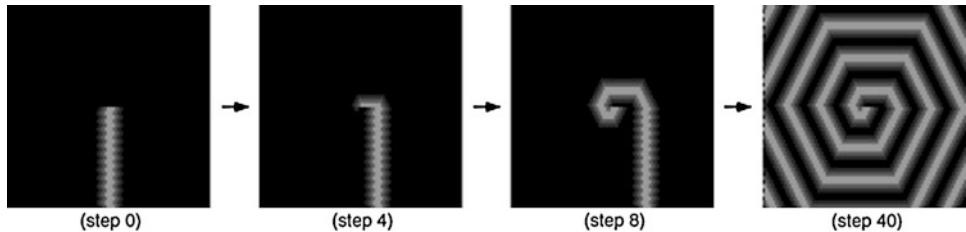
RD chemical systems are known to realize sensible computation when both data and results of the computation are encoded in concentration profiles

of chemical species; the computation is implemented via spreading and interaction of either diffusive or phase waves, while a silicon RD chip is an electronic analog of the chemical RD systems.

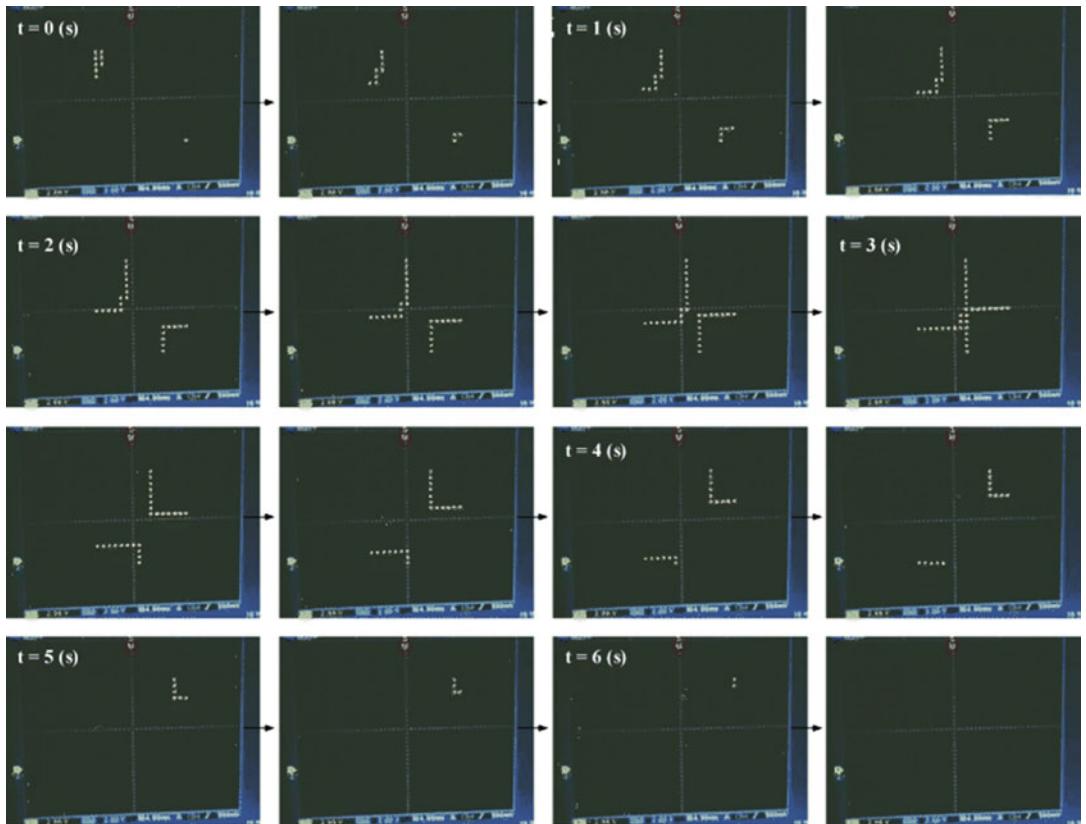
A prototype RD chip implementing a chemical RD processor for a well-known NP-complete problem of computational geometry – computation of a Voronoi diagram – was fabricated. Here we see experimental results for fabricated RD chips and compare the accuracy of information processing in silicon analogs of RD processors and their experimental “wetware” prototypes (Asai et al. 2005b). Figures 5 and 6 show examples of skeleton operation of a T- and “+”- shaped images. As initial images, a glass mask was prepared where “T” and “+” areas were exactly masked. Therefore, cells under the “T” and “+” areas are initially resting, and the rest are initially excited. At its equilibrium, skeletons of “T” and “+” were successfully obtained.

#### A Quadrilateral-Object Composer for Binary Images with Reaction-Diffusion Cellular Automata

A CA LSI architecture that extracted quadrilateral objects from binary images was proposed in Asai et al. (2005a) with a serial combination of parallel CA algorithms, based on RD chemical systems model. Each cell in the CA was implemented by a simple digital circuit called



**Novel Hardware for Unconventional Computing, Fig. 3** Excitatory modelock operations of RD circuit (Asai et al. 2002)

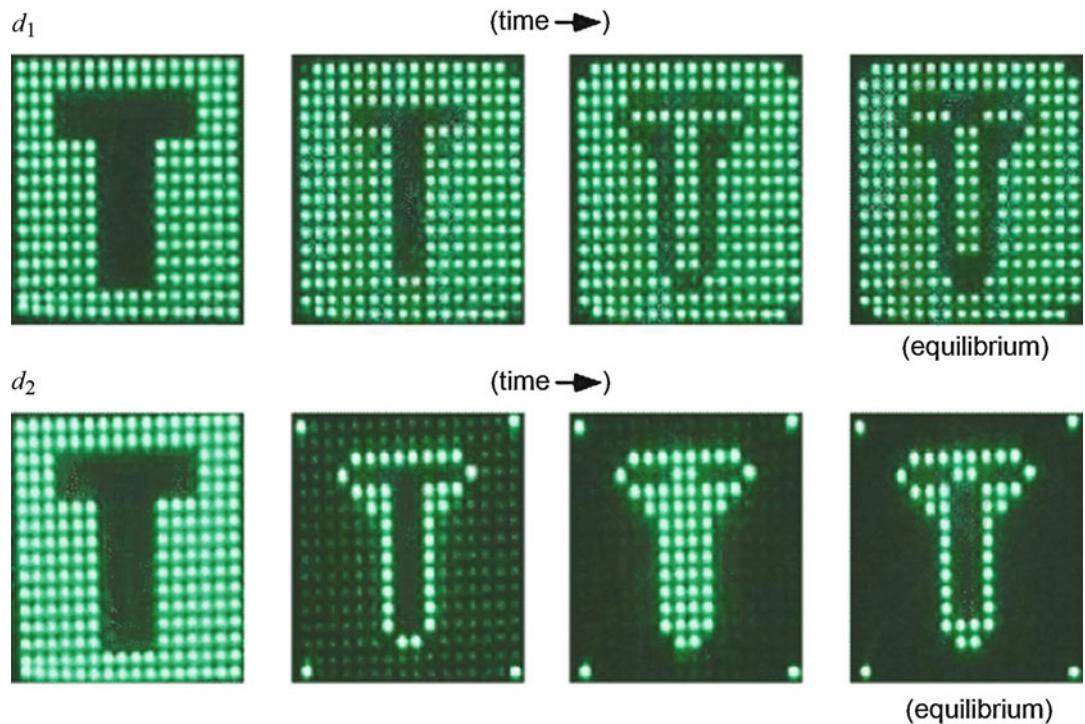


**Novel Hardware for Unconventional Computing, Fig. 4** Snapshots of recorded movie obtained from fabricated RD chip (Matsubara et al. 2004). This chip can operate much faster than real chemical RD systems, even when the system clock frequency is  $O(1)$  Hz, and is much easier to use in various experimental environments.

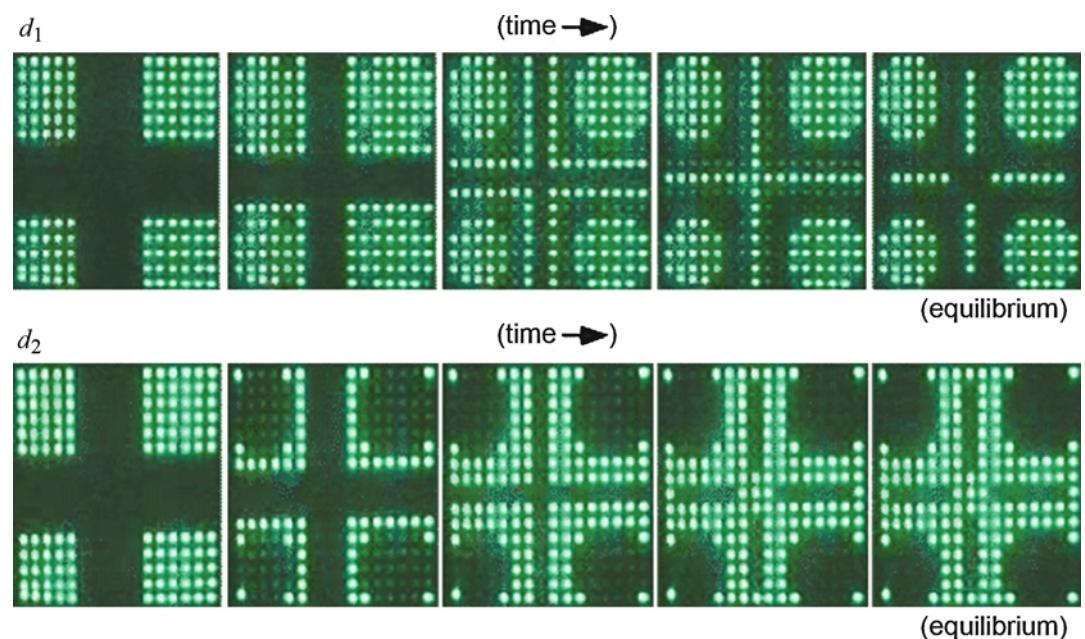
an elemental processor. The CA LSI can be constructed by a large number of elemental processors and their controllers operating in serial and parallel.

Therefore, the chip should encourage RD application developers who use such properties of excitable waves to develop unconventional computing schemes, e.g., chemical image processing, pattern recognition, path planning, and robot navigation

Figure 7a demonstrates object extraction for a natural image. The image was quantized and given to the CA LSI. Figure 7b show the results. The maximum boxes were correctly detected in



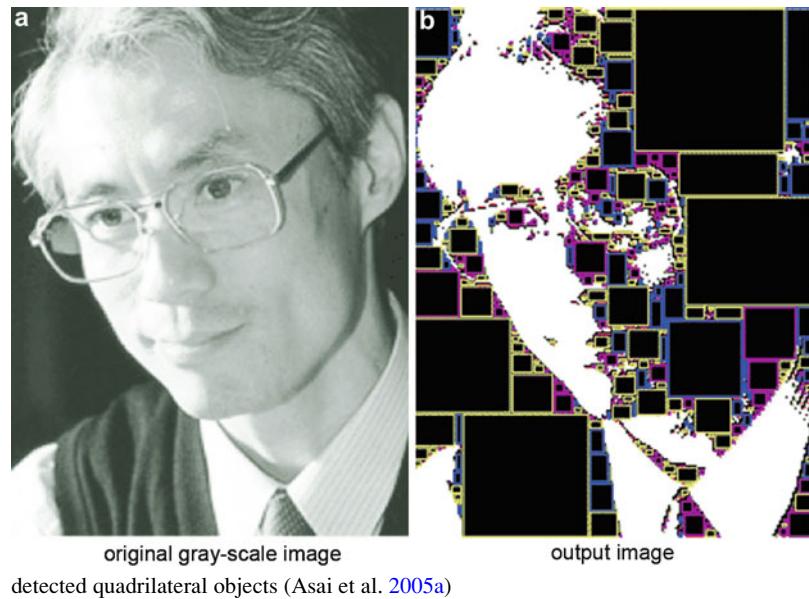
**Novel Hardware for Unconventional Computing, Fig. 5** Skeleton operation with “T” shape (Asai et al. 2005b)



**Novel Hardware for Unconventional Computing, Fig. 6** Skeleton operation with “+” shape (Asai et al. 2005b)

# Novel Hardware for Unconventional Computing,

**Fig. 7** Simulation results for  $181 \times 238$  image, **a** original image and **b** mixed image of quantized image given to CA LSI and



order, as predicted. The input bitmap image that consisted of **181 × 238** pixels was decomposed of 1,020 quadrilateral objects. The bitmap image occupied 43,078 bits (5,385 bytes) on memory, while the objects used 4,080 bytes (8-bit address of 4 corners × 1,020 boxes). The important thing is not discussing the compression rate between bitmap images and extracted objects, but that bitmap images were represented by small number of vector objects, which facilitates picture drawing in terms of the drawing speed if we have variable box window.

One of the most important application targets for the proposed chip is a computer-aided design (CAD) system for LSIs. Conventional LSI CAD tools use polygons to represent device structures. However, recent LSIs include not only polygon patterns but also graphical patterns, consisting of large number of dots, usually imported from image files such as JPEGs, to implement complex analog structures. In the mask manufacturing process, exposing a large number of dot patterns is quite a time-consuming task. Recently, electron beam (EB) lithography systems that can expose wide areas through a quadrilateral window have been produced on a commercial basis. The proposed LSI can produce efficient stream files from binary image files that can easily be handled by the new

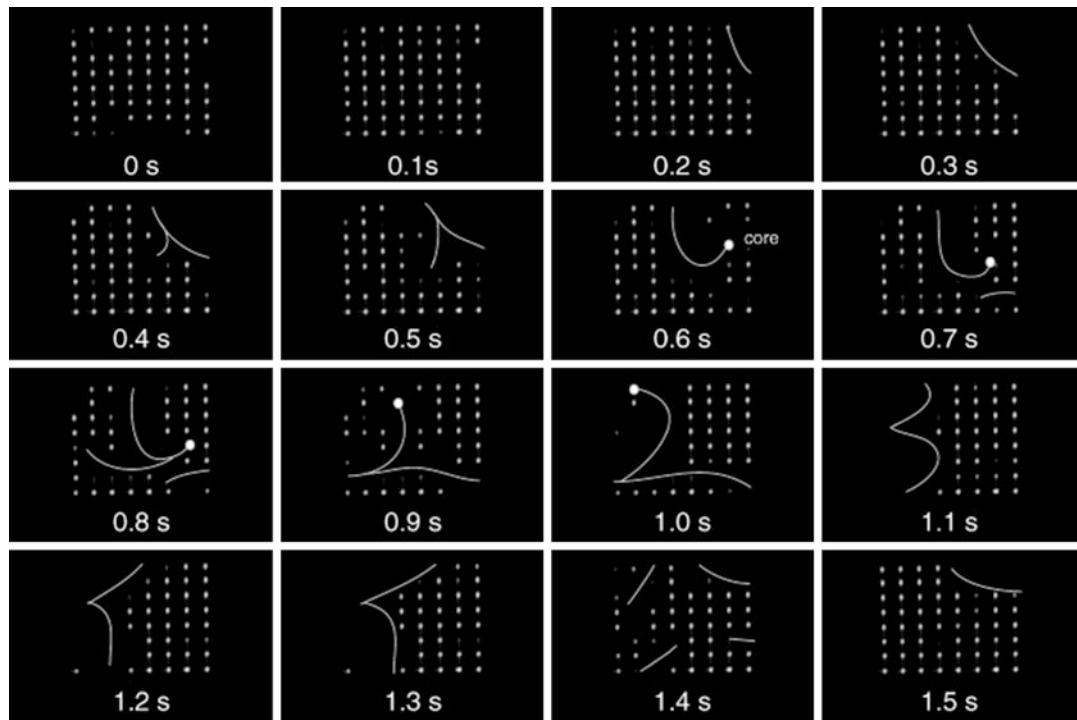
EB systems, by developing simple software that converts the box format, produced by the proposed LSI, to a conventional stream format.

## Analog CMOS Reaction-Diffusion Chip

## Analog Reaction-Diffusion Chip with Hardware Oregonator Model

Silicon devices that imitate the autocatalytic and dissipative phenomena of RD systems were developed (Asai et al. 2005c). Numerical simulations and experimental results revealed that an RD device could successfully produce concentric and spiral waves in the same way as natural RD systems. These results encouraged us to develop new applications based on natural RD phenomena using hardware RD devices.

Figure 8 presents an example where some spiral (modelock) patterns of cell clusters were observed. Snapshots were taken with at intervals of 100 ms. The active-cell clusters and cores of the spirals are superimposed with the figure by white curves and white circles, respectively. Although observing “beautiful” spirals as in chemical RD systems is difficult because of the small number of cells, the appearance and disappearance of small sections of spiral waves were successfully observed. RD devices and circuits are useful not



**Novel Hardware for Unconventional Computing, Fig. 8** Spiral patterns on RD chip (weak connections between cells) (Asai et al. 2005c)

only for hardware RD systems but also for constructing modern neuro-chips. The excitatory and oscillatory behaviors of an RD device and circuit are very similar to actual neurons that produce sequences of identically shaped pulses in time, called *spikes*. Recently, Fukai demonstrated that an inhibitory network of spiking neurons achieves robust and efficient neural competition on the basis of a novel timing mechanism of neural activity (Fukai 1996). A network with such a timing mechanism may provide an appropriate platform to develop analog LSI circuits and could overcome problems with analog devices, namely, their lack of precision and reproducibility.

### Striped and Spotted Pattern Generation on RD Cellular Automata

A novel RD model that is suitable for LSI implementation and its basic LSI architecture was proposed in Suzuki et al. (2007).

The model employs linear diffusion fields of activators and inhibitors and a discrete transition

rule after diffusion. Image-processing LSI circuits based on pattern formation in RD systems were developed. Continuous diffusion fields and an analog state variable were introduced to improve the Young's local activator-inhibitor model (Young 1984).

A model pattern diagram was produced on a 2D parameter space through extensive numerical simulations. The spatial frequency and form (striped or spotted) could be controlled with only two parameters. Theoretical analysis of the one-dimensional model proved that (i) spatial distribution given by a periodic square function is stable at the equilibrium and (ii) the spatial frequency is inversely proportional to the square root of a diffusion coefficient of the inhibitors.

A basic circuit for the proposed model was designed, i.e., an RD LSI based on the analog computing method where the concentration of chemicals was represented by a two-dimensional voltage distribution and the cell voltage was diffused step by step. By mimicking two diffusion

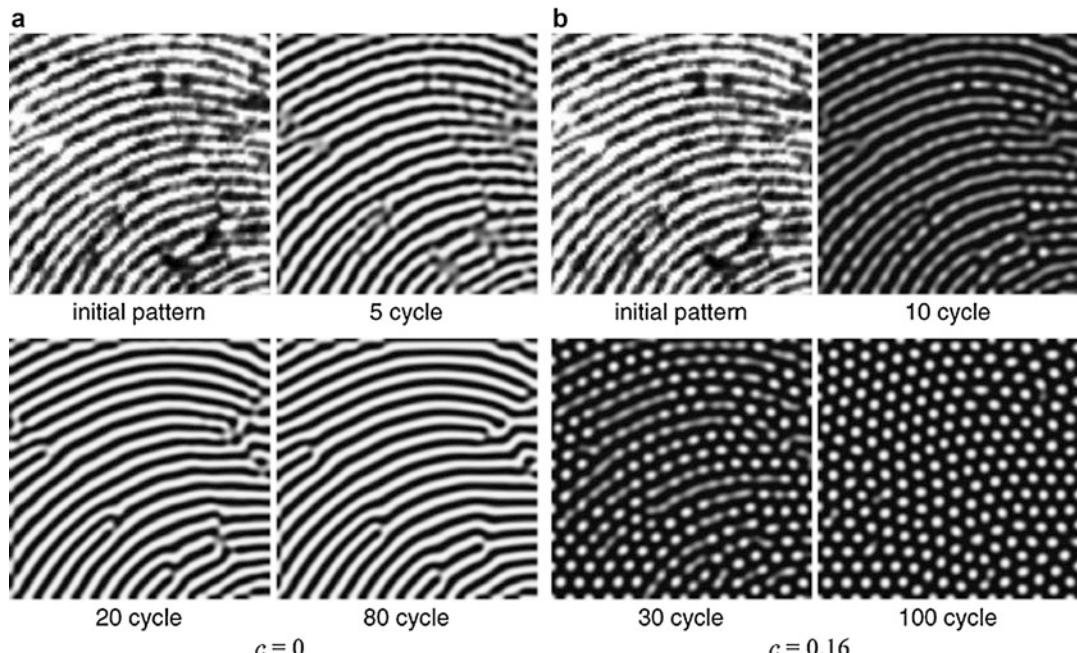
fields with the proposed model in one diffusion circuit on the LSI, one can reduce the area of the unit cell circuit.

Figure 9a has snapshots of pattern formation in the circuit. One can estimate that the system produces striped patterns from the theory (Suzuki et al. 2007). Therefore, a fingerprint pattern was used as an initial input. Noisy local patterns were repaired by their surrounding striped patterns, as time increased. The circuit required 50 cycles (8,000 clocks) to reach equilibrium. Figure 9b shows the results for spot pattern generation. The same initial input as in Fig. 9a was given to the circuit. As expected from the theory, spotted patterns were obtained. The pattern formation process was the same as in Fig. 9a where noisy local spots were restored by surrounding global spotted patterns. Therefore, this circuit would be suitable for restoring regularly arranged spotted patterns such as polka dot patterns. The system took 100 cycles (16,000 clocks) until it reached equilibrium to restore the spotted patterns.

## Reaction-Diffusion Computing Devices Based on Minority-Carrier Transport in Semiconductors

A massive parallel computing device was designed (Asai et al. 2004) based on principles of information processing in RD chemical media (Adamatzky 2000, 2001) (Figs. 10 and 11). This device imitates autocatalytic and dissipative phenomena of the chemical RD systems; however, comparing to real chemical medium, the semiconductor analog of RD computers functions much faster. Operational characteristics of the RD silicon devices and feasibility of the approach on several computational tasks are shown in Asai et al. (2004). The results indicate that the proposed RD device will be a useful tool for developing novel hardware architectures based on RD principles of information processing.

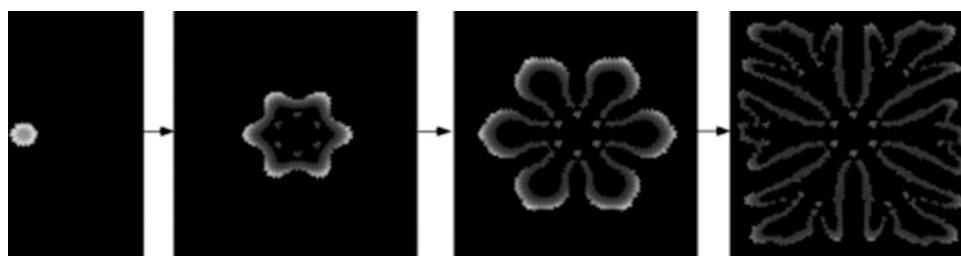
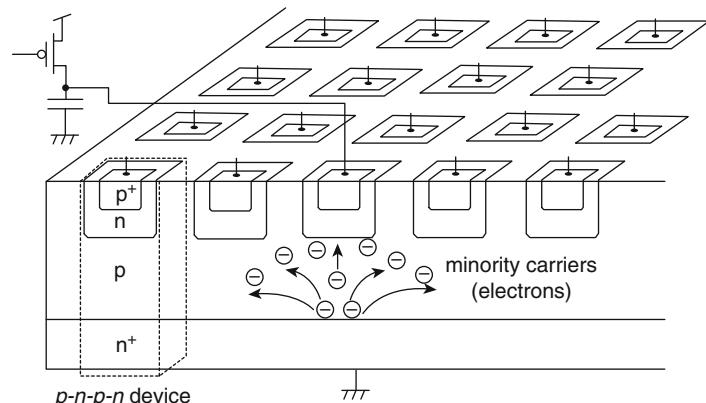
Practical value of RD chemical systems is significantly reduced by low speed of traveling waves which makes real-time computation senseless. One of the cost-efficient options to overcome the speed limitations of RD



**Novel Hardware for Unconventional Computing, Fig. 9** Snapshots of pattern formation from initial fingerprint image (Suzuki et al. 2007)

**Novel Hardware for Unconventional Computing,**

**Fig. 10** Construction of two-dimensional RD device with vertical  $p-n-p-n$  device (Asai et al. 2004)



**Novel Hardware for Unconventional Computing, Fig. 11** Simulation results of RD device producing multiplicating patterns (Asai et al. 2004)

computers while preserving unique features of wave-based computing is to implement RD chemical computers in silicon. The velocity of traveling wave fronts in typical reaction-diffusion systems, e.g., BZ reaction, is  $10^{-2}$  m/s (Tóth et al. 1994), while that of a hardware RD system will be over a million times faster than that of the BZ reaction, independent of system size (Adamatzky et al. 2005). The increase in speed will be indispensable for developers of RD computers. Moreover, if an RD system is implemented in integrated circuits, then we would be able to artificially design various types of RD spatiotemporal dynamics and thus develop parallel computing processors for novel applications. Basing on experimental evidences of RD-like behavior, namely, traveling current density filaments (Niedernostheide et al. 1994), in  $p-n-p-n$  devices, a novel type of semiconductor RD computing device, where minority carriers diffuse as chemical species and reaction elements are represented by  $p-n-p-n$  diodes, was proposed.

### Single-Electron Reaction-Diffusion System

This section introduces a single-electron device that is analogous to the RD system. This electrical RD device consists of a two-dimensional array of single-electron nonlinear oscillators that are combined with one another through diffusive coupling. The device produces animated spatio-temporal patterns of node voltages, e.g., a rotating spiral pattern similar to that of a colony of cellular slime molds and a dividing-and-multiplying pattern that reminds us of cell division. A method of fabricating actual devices by using self-organized crystal growth technology is also described. The following is an excerpt from Oya et al. (2005). For details, see the reference.

Figure 12 shows an electrical oscillator constructed by a single-electron circuit. It consists of tunneling junction  $C_j$  and high resistance  $R$  connected in series at node 1 and biased by positive voltage  $V_{dd}$ . This circuit is an elementary component of single-electron circuits known as

the single-electron transistor (SET) cell (see Gravert and Devoret 1992 for detailed explanation). A SET cell only has a single variable, voltage  $V_1$  of node 1, but it can be oscillatory or excitatory in operation – which is indispensable in creating RD systems – because the node voltage can produce a discontinuous change because of electron tunneling. In continuous variable systems such as chemical reaction systems, two or more variables are needed for oscillatory and excitatory operations.

The SET cell operates as a nonlinear oscillator at low temperatures at which the Coulomb blockade effect occurs. It is oscillatory (astable) if  $V_{dd} > e/(2C_j)$  ( $e$  is elementary charge) and produces nonlinear oscillation in voltage at node 1

(Fig. 13a). The node voltage gradually increases as junction capacitance  $C_j$  is charged through resistance  $R$  and then drops discontinuously because of electron tunneling through the junction, again gradually increasing to repeat the same cycles. In contrast, the oscillator is excitatory (monostable) if  $V_{dd} < e/(2C_j)$  and produces single-pulse operation excited by an external trigger (Fig. 13b). A modified Monte Carlo method is used for the simulation. Kuwamura and his colleagues (1994) have given details of this method. Also see Appendix in Yamada et al. (2001). For constructing electrical RD systems, oscillatory oscillators and excitatory ones, or both, can be used.

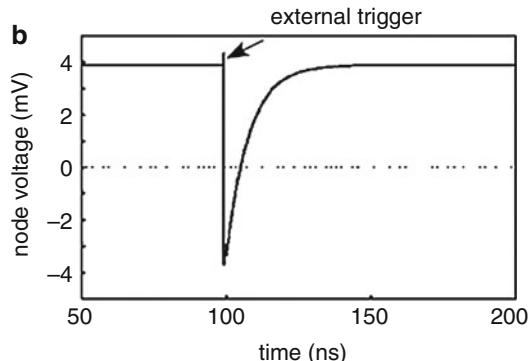
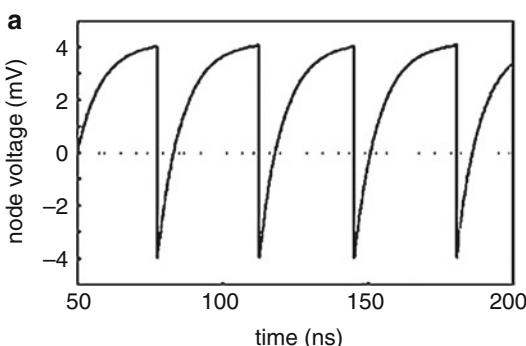
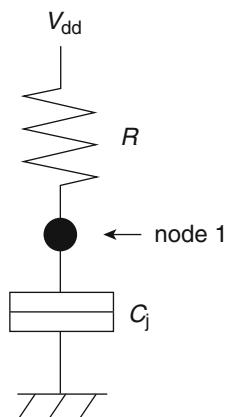
The oscillator exhibits discontinuous, probabilistic kinetics resulting from electron tunneling. The kinetics is given in the form of

$$\frac{dV_1}{dt} = \frac{V_{dd} - V_1}{RC_j} - \frac{e}{C_j} \delta\left(V_1 - \frac{e}{2C_j} - \Delta V\right),$$

where  $\delta(\cdot)$  represents a discontinuous change in node voltage caused by electron tunneling. Probabilistic operation arises from the stochastic nature of tunneling; i.e., a time lag (a waiting time) exists between when junction voltage exceeds tunneling threshold  $e/(2C_j)$  and when tunneling actually occurs. This effect is represented by delay term  $\Delta V$  in the equation. Because the value of  $\Delta V$  has probabilistic

### Novel Hardware for Unconventional Computing,

**Fig. 12** Single-electron oscillator (a SET cell) consisting of tunneling junction  $C_j$ , high resistance  $R$  connected at node 1, and positive bias voltage  $V_{dd}$



### Novel Hardware for Unconventional Computing,

**Fig. 13** Operation of the oscillator. Waveforms of node voltage are shown for (a) self-induced oscillation and (b) monostable oscillation, simulated with following set of parameters: tunneling junction capacitance  $C_j = 20 \text{ aF}$ ,

tunneling junction conductance =  $1 \mu\text{S}$ , high resistance  $R = 400 \text{ M}\Omega$ , and zero temperature. Bias voltage  $V_{dd} = 4.2 \text{ mV}$  for self-induced oscillation and  $V_{dd} = 3.8 \text{ mV}$  for monostable oscillation

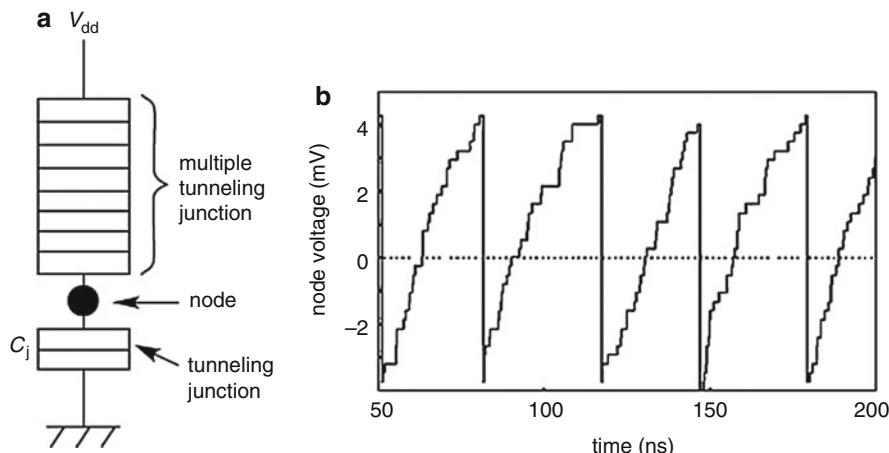
fluctuations in every tunneling event and cannot be expressed in analytical form, so Monte Carlo simulation is necessary for studying the behavior of the oscillator.

In fabricating actual oscillators, a high resistance of hundreds of mega-ohms or more is not easy to implement on an LSI chip. A better way is to use a multiple tunneling junction, i.e., a series of many tunneling junctions, instead of high resistance (Fig. 14a). This structure also enables oscillatory and excitatory operations to be obtained because sequential electron tunneling through a multiple tunneling junction has a similar effect to current flowing at high resistance (Fig. 14b). In the following sections, however, the high-resistance SET cell (Fig. 12) is used to construct electrical RD systems because less computing time is required in simulating RD operation. One can expect that the knowledge obtained from high-resistance RD systems will be able to be applied to RD systems consisting of multiple-junction oscillators.

To construct RD systems, oscillators have to be connected with one another so that they will interact through “diffusive” coupling to generate synchronization and entrainment. To do this, the oscillators are connected by means of

intermediary oscillation cells and coupling capacitors. Figure 15a illustrates the method of connection with a one-dimensional chain of oscillators. The oscillators (SET cells denoted by A1, A2, . . . , with their nodes represented by closed circles) are connected with their neighboring oscillators through intermediary oscillation cells (SET cells denoted by B1, B2, . . . , with their nodes represented by open circles) and coupling capacitors C. One can use an excitatory SET cell biased with a negative voltage  $-V_{ss}$  as the intermediary oscillation cell. An excitatory SET cell biased with a negative voltage  $-V_{ss}$  is used here as the intermediary oscillation cell.

When electron tunneling occurs in an oscillator in this structure, the node voltage of the oscillator changes from positive to negative, and this induces, through coupling capacitor C, electron tunneling in an adjacent intermediary cell. The induced tunneling changes the node voltage of the intermediary cell from negative to positive, and this induces electron tunneling in an adjacent oscillator. In this way, electron tunneling is transmitted from one oscillator to another along the oscillator chain. There is a time lag between two tunneling events in two neighboring oscillators as if these oscillators interacted through diffusion.



#### Novel Hardware for Unconventional Computing,

**Fig. 14** Single-electron oscillator with a multiple tunneling junction: (a) circuit configuration and (b) simulated self-induced oscillation. Parameters are capacitance of single tunneling junction  $C_j = 10 \text{ aF}$ , conductance of the

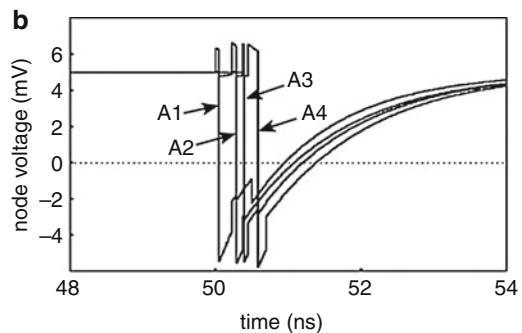
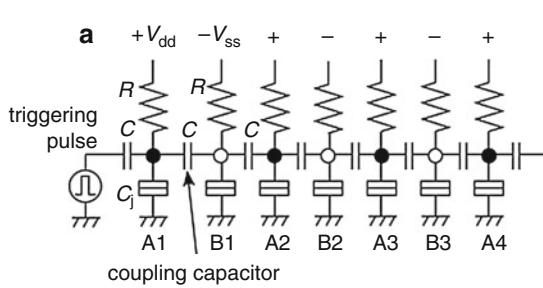
single tunneling junction =  $1 \mu\text{S}$ , and 30 tunneling junctions in the multiple tunneling junction; capacitance and conductance of a tunneling junction in the multiple tunneling junction are  $300 \text{ aF}$  and  $50 \text{ nS}$ , bias voltage  $V_{dd} = 8.6 \text{ mV}$ , and zero temperature

This phenomenon is not diffusion itself and cannot be expressed in the form  $D\Delta \mathbf{u}$  in Eq. 1 but can be used as a substitute for diffusion.

The transmission of tunneling with delay is illustrated in Fig. 15b with simulated results for a chain of excitatory oscillators with intermediary cells. Electron tunneling was induced in the leftmost oscillator by a triggering pulse, and it was transmitted to the right along the chain with delay. In other words, an excitation wave of

tunneling traveled to the right along the chain. Its delay in traveling from one oscillator to a neighbor has probabilistic fluctuations because of the stochastic nature of tunneling, but this is not a problem for applications to RD systems.

An electrical RD system can be constructed by connecting oscillators into a network by means of intermediary cells and coupling capacitors (Fig. 16). Each oscillator is connected to its neighboring four oscillators by means of four



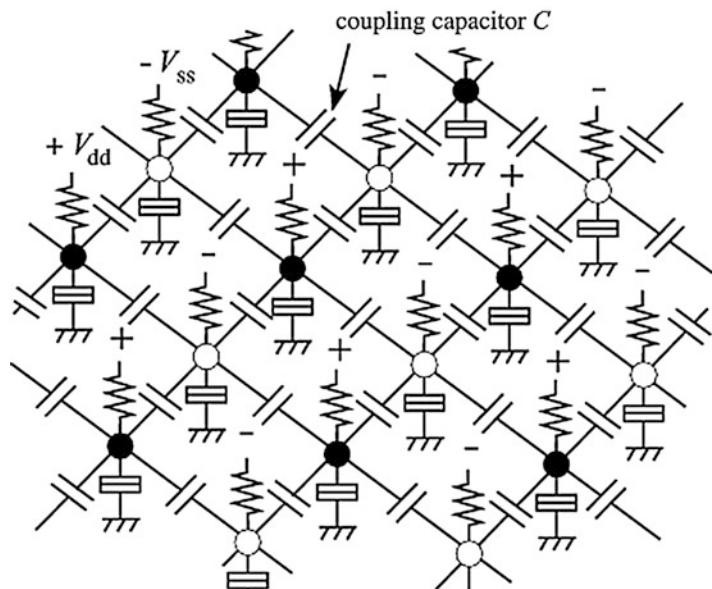
#### Novel Hardware for Unconventional Computing,

**Fig. 15** Diffusive connection of oscillators. (a) One-dimensional chain of oscillators ( $A_1, A_2, \dots$ ) with intermediary cells ( $B_1, B_2, \dots$ ) and coupling capacitors  $C$ . For study of the transmission of tunneling, a triggering pulse generator is connected to the left end. (b) Transmission of tunneling through the chain of excitatory oscillators. The waveform of node voltage is plotted for each oscillator.

A triggering pulse was applied to the leftmost oscillator, and tunneling started at the oscillator to transmit along the chain with delay. Jumps in curves  $A_1$ – $A_4$  result from electron tunneling in oscillators  $A_1$ – $A_4$ . Simulated with a set of parameters:  $C_j = 10 \text{ aF}$ ,  $C = 2 \text{ aF}$ ,  $R = 77 \text{ M}\Omega$ , tunneling junction conductance =  $5 \mu\text{S}$ ,  $V_{dd} = 5 \text{ mV}$ ,  $-V_{ss} = -5 \text{ mV}$ , and zero temperature

#### Novel Hardware for Unconventional Computing,

**Fig. 16** Two-dimensional RD system consisting of the network of single-electron oscillators. Each oscillator (*closed-circle* node) is connected with four neighboring oscillators by means of four intermediary cells (*open-circle* nodes) and coupling capacitors



intermediary cells and coupling capacitors. This is a two-dimensional RD system. A three-dimensional RD system can also be constructed in a similar way by arranging oscillators into a cubic structure and connecting each oscillator with its six neighboring oscillators by means of six intermediary cells and coupling capacitors.

In the single-electron RD system, the node voltage of each oscillator changes temporally as the oscillators operate through mutual interactions. Consequently, a two-dimensional spatio-temporal pattern of the node voltages is produced on the RD system. Since this voltage pattern corresponds to the dissipative structure in chemical RD systems, it can be called an “electrical dissipative structure.”

A variety of electrical dissipative structures are produced from different sets of system parameters. To understand the behavior of an electrical RD system entirely, a phase diagram for the system must be drawn, i.e., a diagram that depicts – in the multidimensional space of system parameters – what kind of dissipative structure will appear for each set of parameter values. However, a phase diagram for the RD system cannot be drawn without a long numerical computer simulation because its reaction-diffusion kinetics cannot be expressed in analytical form. Instead, few examples of electrical dissipative structures simulated with a few sample sets of parameter values are demonstrated.

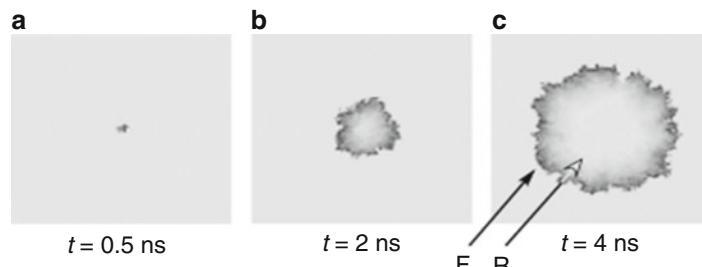
Although a single-electron RD system differs greatly from chemical RD systems in terms of reaction-diffusion kinetics, it can produce

dissipative structures similar to those of chemical RD systems. Here, three examples are exhibited, i.e., an expanding circular pattern, a rotating spiral pattern, and a dividing-and-multiplying pattern. The following will have the results simulated for a RD system consisting of  $201 \times 201$  excitatory oscillators and  $200 \times 200$  intermediary cells.

### Expanding Circular Pattern

A single-electron RD system consisting of excitatory oscillators is in a stable uniform state as it stands. Once a triggering signal is applied to an oscillator in the system, an excitation wave of tunneling starts at the oscillator and propagates in all directions to form an expanding circular pattern. This can be seen in Fig. 17; the node voltage of each oscillator is represented by a gray scale: the light shading means high voltage, and the dark means low voltage. The front F of the wave is the region where tunneling just occurred, and, therefore, the node voltage of the oscillators is at the lowest negative value. The front line is uneven or irregular because the velocity of the traveling wave fluctuated in each direction throughout the process because of the stochastic waiting time of tunneling.

After the excitation wave passed through, the node voltage of each oscillator gradually increased to return to its initial value, the positive bias voltage. This is indicated in the figure by the light shading on the rear R of the wave. If a triggering



**Novel Hardware for Unconventional Computing,**

**Fig. 17** Expanding circular pattern in the single-electron RD system. Snapshots for three time steps. Simulated with parameters: tunneling junction capacitance  $C_j = 1 \text{ aF}$ ,

tunneling junction conductance =  $1\,019 \mu\text{S}$ , high resistance  $R = 137.5 \text{ M}\Omega$ , coupling capacitance  $C = 1 \text{ aF}$ , bias voltage =  $V_{dd} = 16.5 \text{ mV}$ ,  $-V_{ss} = -16.5 \text{ mV}$ , and zero temperature

signal is applied repeatedly to one oscillator, a concentric circular wave – called a target pattern in chemical RD systems – will be generated.

### Rotating Spiral Pattern

This pattern appears when an expanding circular wave is chipped by external disturbance, thereby making an endpoint to appear in the wave front. With this endpoint acting as a center, the wave begins to curl itself to form a rotating spiral pattern (Fig. 18). The principle of curling is similar to that in chemical RD systems.

In this example, a triggering signal was applied to the middle oscillator on the left of the RD system. When an excitation wave started and expanded a little, the lower half of the wave was chipped by resetting the node voltage of oscillators to zero (Fig. 18a). After that, the RD system was left to operate freely, and a rotating spiral pattern of node voltages automatically generated as can be seen in Fig. 18b–f.

### Dividing-and-Multiplying Pattern

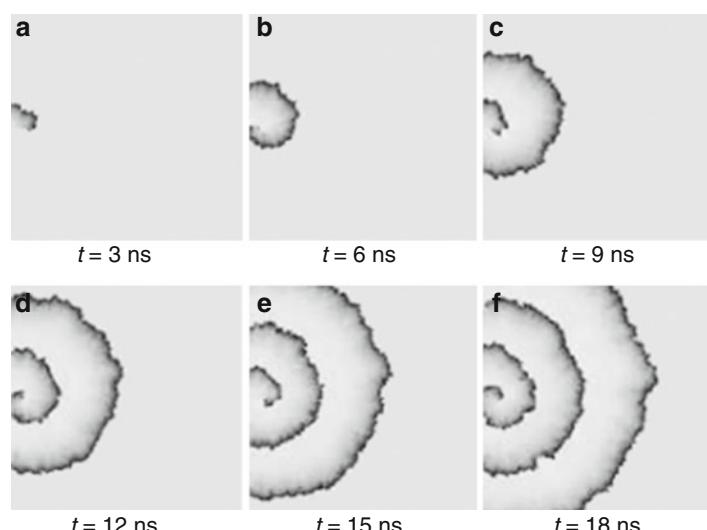
This pattern appears when the coupling between oscillators is weak (i.e., small coupling capacitance or low bias voltage). When this happens,

electron tunneling in an oscillator cannot be transmitted to all four adjacent intermediary cells; e.g., tunneling can be transmitted to the right and left cells but not to the upper and lower cells. As a result, an expanding node-voltage pattern splits into pieces, and each piece again expands to split again. This produces dividing-and-multiplying patterns (Fig. 19). The principle of division is different from that in chemical RD systems, but the behavior of created patterns is somewhat similar. In a way, we may consider that there are electrical microbes consisting of negative voltages living on the RD system, eating positive charges on nodes as food, and propagating so that they can spread all over the system.

The unit element in this RD system is a single-electron oscillator coupled with four neighbors. The multiple-tunneling-junction oscillator (Fig. 14) is preferable for this element because it can be made without high resistance, which is difficult to implement on an LSI chip. Arranging such oscillators into a two-dimensional array produces an RD system, so the next task is to fabricate many identical oscillators on a substrate. Figure 20a shows the three-dimensional and cross-sectional schematics for the structure of the device. Each oscillator consists of a conductive nanodot (minute dot) with four coupling arms, and there is a tunneling junction between

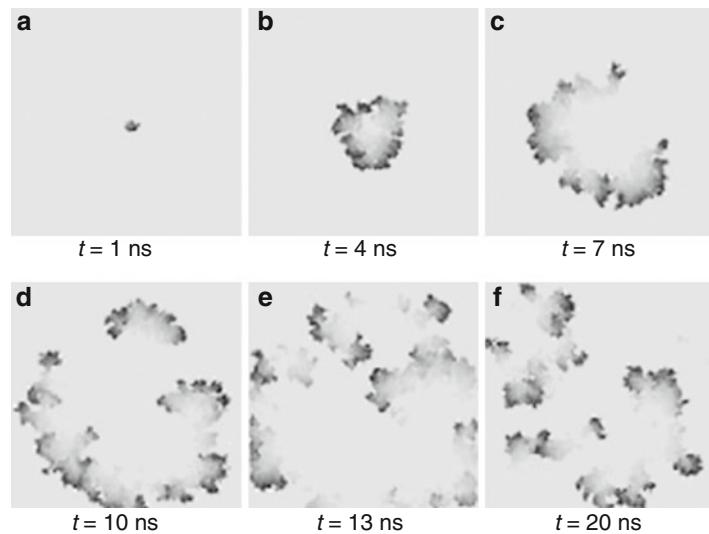
#### Novel Hardware for Unconventional Computing,

**Fig. 18** Rotating spiral pattern. Snapshots for six time steps. Simulated with the same parameters as used for Fig. 17

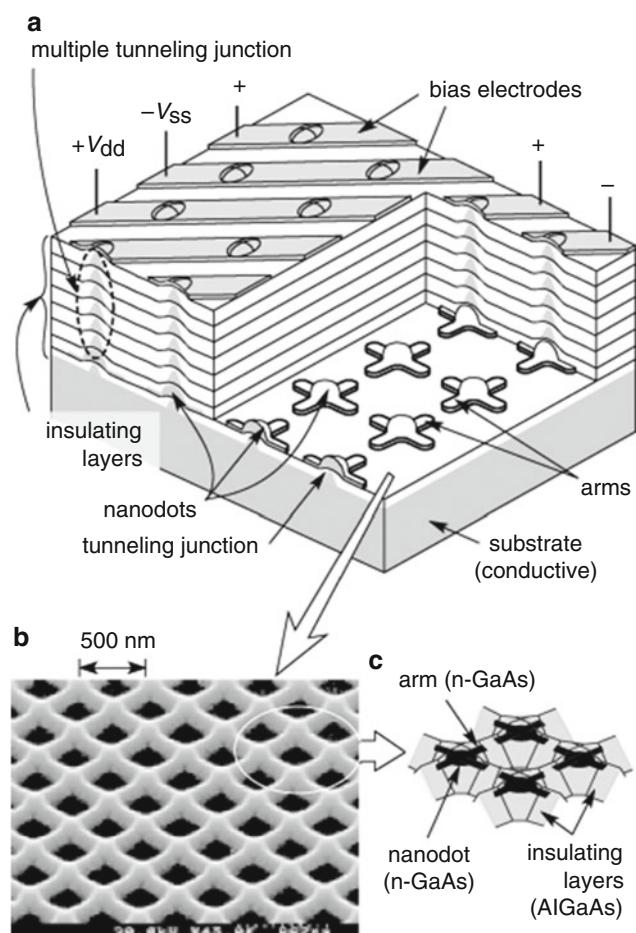


**Novel Hardware for Unconventional Computing,**

**Fig. 19** Dividing-and-multiplying pattern. Snapshots for six time steps. Simulated with the same parameters as used for Fig. 17 except that  $R = 150.5 \text{ M}\Omega$ ,  $V_{dd} = 15.8 \text{ mV}$ , and  $-V_{ss} = -15.8 \text{ mV}$


**Novel Hardware for Unconventional Computing,**

**Fig. 20** Device structure for the single-electron RD system. (a) Three-dimensional and cross-sectional schematics, (b) SEM photograph of a two-dimensional array of GaAs nanodots with coupling arms and tunneling junctions, and (c) schematic diagram of the nanodots with coupling arms



the nanodot and the conductive substrate beneath it. Many series-connected junctions run between the nanodot and a positive bias or a negative bias electrode. Capacitive coupling between neighboring oscillators can be achieved by laying their coupling arms close to each other.

The key in this construction is to prepare a large arrangement of nanodots with coupling arms and tunneling junctions. A process technology that could be used to fabricate the RD-system structure was previously proposed and demonstrated (Oya et al. 2002). This technology uses self-organized crystal growth achieved by selective area metal-organic vapor phase epitaxy (SA-MOVPE), and it can be used to fabricate GaAs nanodots with arms and tunneling junctions on a GaAs substrate by making use of the dependence of the crystal growth rate on crystal orientation (for detailed explanation, see Kumakura et al. 1995, 1997). With this technology, a nanodot with four coupling arms can be formed automatically in a self-organizing manner. This technology can also be used to automatically create the structure for multiple tunneling junctions on nanodots simply by repeating the growth of an *n*-type GaAs layer and an insulating AlGaAs layer. Using such a process, the formation of GaAs nanodots with their arms and tunneling junctions beneath them was succeeded, in the form of a two-dimensional array on a substrate (Figs. 20b, c), though the technology is not yet perfect and a complete device has not been fabricated yet. An improved process technology to form GaAs nanodots with arms and multiple tunneling junctions is now under development, where the arms and multiple tunneling junctions are arranged regularly with a smaller pitch of 100 nm or less (corresponding to 1,010 oscillators/cm<sup>2</sup>). With the improved process technology, we will be able to integrate coupled single-electron oscillators on a chip and proceed from there to develop reaction-diffusion LSIs.

## Collision-Based RD Computers

Present digital LSI systems consist of a number of combinational and sequential logic circuits as

well as related peripheral circuits. A well-known basic logic circuit is a two-input NAND circuit that consists of four metal-oxide semiconductor field-effect transistors (MOS FETs) where three transistors are on the current path between the power supply and the ground. Many complex logic circuits can be constructed by not only populations of a large number of NAND circuits but also special logic circuits with a small number of transistors (there are more than three transistors on the current path) compared with NAND-based circuits.

A straightforward way to construct low-power digital LSIs is to decrease the power supply voltage because the power consumption of digital circuits is proportional to the square of the supply voltage. In complex logic circuits, where many transistors are on the current paths, the supply voltage cannot be decreased due to stacking effects of transistors' threshold voltages, even though the threshold voltage is decreasing as LSI fabrication technology advances year by year. On the other hand, if two-input basic gates that have the minimum number of transistors (three or less) on the current path are used to decrease the supply voltage, a large number of the gates will be required for constructing complex logic circuits.

The Reed-Muller expansion (Muller 1954; Reed 1954), which expands logical functions into combinations of AND and XOR logic, enables us to design "specific" arithmetic functions with a small number of gates, but it is not suitable for arbitrary arithmetic computation. Pass transistor logic (PTL) circuits use a small number of transistors for basic logic functions, but additional level-restoring circuits are required for every unit (Song and Asada 1998). Moreover, the acceptance of PTL circuits into mainstream digital design critically depends on the availability of tools for logic, physical synthesis, and optimization. Current-mode logic circuits also use a small number of transistors for basic logic, but their power consumption is very high due to the continuous current flow in turn-on states (Alioto and Palumbo 2005). Subthreshold logic circuits where all the transistors operate under their threshold voltage are expected to exhibit ultralow power consumption, but the operation speed is

extremely slow (Soeleman et al. 2001). Binary decision diagram logic circuits are suitable for next-generation semiconductor devices such as single-electron transistors (Asahi et al. 1998; Shelar and Sapatnekar 2001), but not for present digital LSIs because of the use of PTL circuits.

To address the problems above concerning low-power and high-speed operation in digital LSIs, a method of designing logic circuits with collision-based fusion gates, which is inspired by collision-based RD computing (RDC) (Adamatzky 2002; Adamatzky et al. 2005), is described. In the following sections, we see a new interpretation of collision-based RDC, especially concerning directions and speeds of propagating information quanta. We also see basic logical functions constructed by collision-based fusion gates and discuss the number of transistors in classical and fusion-gate logic circuits (Yamada et al. 2008).

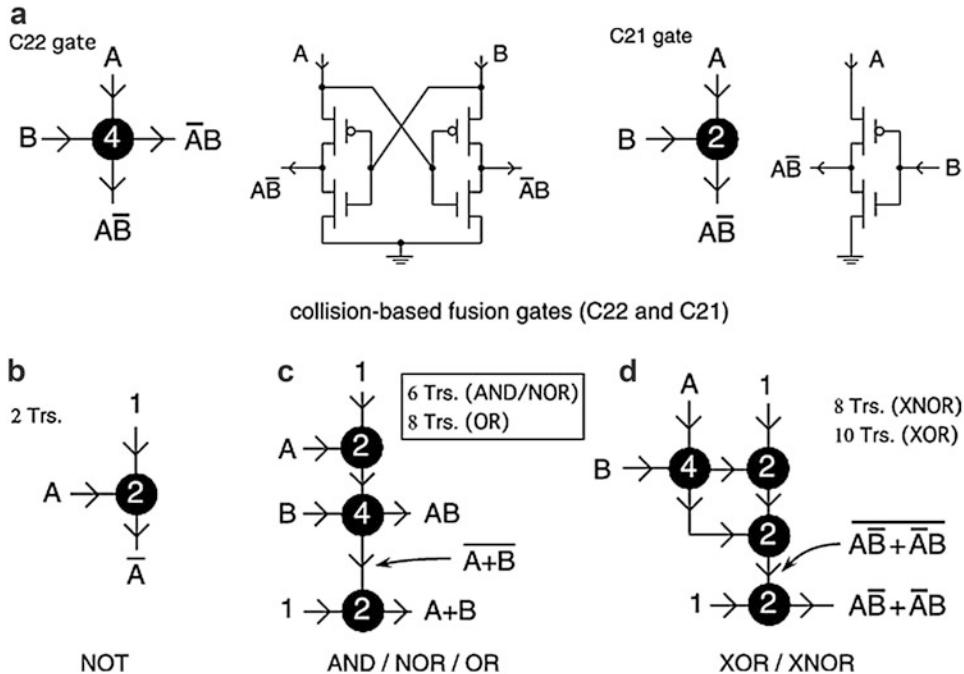
## Collision-Based Reaction-Diffusion Computing for Digital LSIs

Adamatzky proposed how to realize arithmetical scheme using wave fragments traveling in an RD medium where excitable chemical waves disappear when they collide each other (Adamatzky 2002; Adamatzky et al. 2005). His cellular automaton model mimicked localized excitable waves (wave fragments) traveling along columns and rows of the lattice and along diagonals. The wave fragments represented values of logical variables where logical operations were implemented when wave fragments collided and were annihilated or reflected as a result of the collision. One can achieve basic logical gates in the cellular automaton model and build an arithmetic circuit using the gates (Adamatzky 2002).

The cellular automaton model for basic logic gates has been implemented on digital LSIs (Adamatzky et al. 2005). Each cell consisted of several tens of transistors and was regularly arranged on a 2D chip surface. To implement a one-bit adder, for example, by collision-based cellular automata, at least several tens of cells are required to allocate sufficient space for the

collision of wave fragments (Adamatzky 2002). This implies several hundreds of transistors are required for constructing just a one-bit adder. Direct implementation of the cellular automaton model is therefore a waste of chip space, as long as the single cell space is decreased to the same degree of chemical compounds in spatially continuous RD processors.

What happens if wave fragments travel in “limited directions instantaneously”? Our possible answers to this question are depicted in Fig. 21. Figure 21a shows 2-in 2-out (C22) and 2-in 1-out (C21) units representing two perpendicular “limited” directions of wave fragments, i.e., North–South and West–East fragments. The number of MOS transistors in each unit is written inside the black circle in the figure. The input fragments are represented by values A and B where A (or B) = “1” represents the existence of a wave fragment traveling North–South (or West–East) and A (or B) = “0” represents the absence of wave fragments. When A = B = “1”, wave fragments collide at the center position (black circle) and then disappear. Thus, East and South outputs are “0” because of the disappearance. If A = B = “0”, the outputs will be “0” as well because of the absence of the fragments. When A = “1” and B = “0”, a wave fragment can travel to the South because it does not collide with a fragment traveling West–East. The East and South outputs are thus “0” and “1”, respectively, whereas they are “1” and “0”, respectively, when A = “0” and B = “1”. Consequently, logical functions of this simple “operator” are represented by  $\bar{A}B$  and  $A\bar{B}$ , as shown in Fig. 21a left. We call this operator a collision-based “fusion gate,” where two inputs correspond to perpendicular wave fragments and one (or two) output represents the results of the collision (transparent or disappear) along the perpendicular axes. Figure 21b–d represents basic logic circuits constructed by combining several fusion gates. The simplest example is shown in Fig. 21b where the NOT function is implemented by a C21 gate (two transistors). The North input is always “1”, whereas the West is the input (A) of the NOT function. The output appears on South node ( $\bar{A}$ ). Figure 21c represents a combinational



**Novel Hardware for Unconventional Computing,**  
**Fig. 21** Construction of collision-based fusion gates  
(Yamada et al. 2008). (a) Definition of 2-in 2-out (C22)

circuit of three fusion gates (two C21 gates and one C22 gate) that produces AND, NOR, and OR functions.

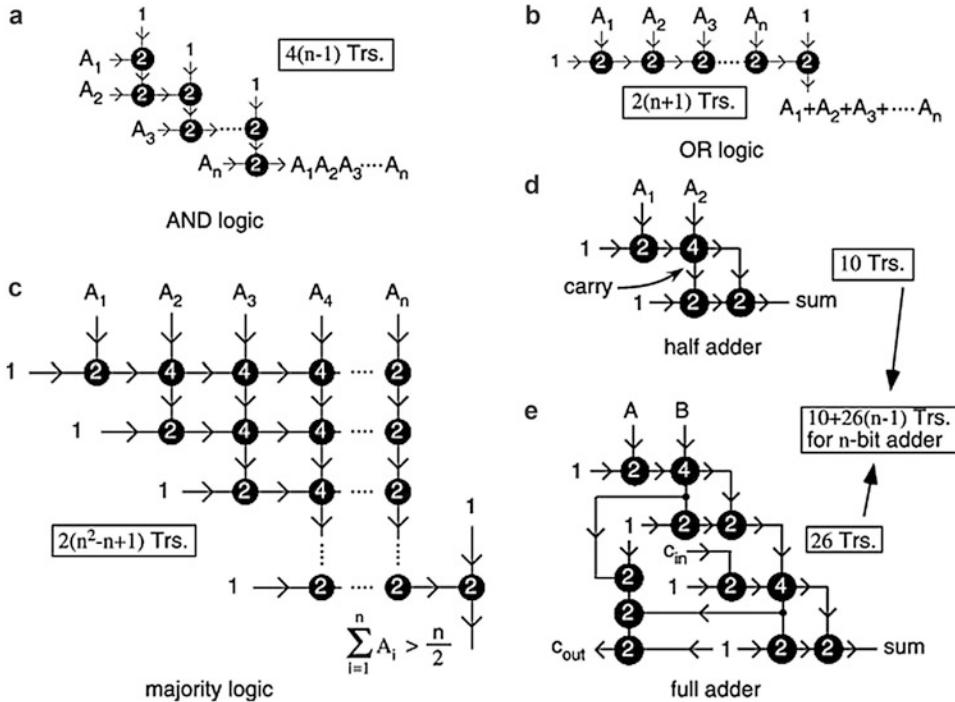
Exclusive logic functions are produced by three (for XNOR) or four (for XOR) fusion gates as shown in Fig. 21d. The number of transistors for each function is depicted in the figure (inside the white boxes).

A collision-based fusion gate receives two logical inputs ( $A$  and  $B$ ) and produces one (C21) or two (C22) logical outputs, i.e.,  $\bar{A}\bar{B}$  for C21 and  $\bar{A}B$  and  $A\bar{B}$  for C22. Unit circuits for C22 and C21 gates receive logical (voltage) inputs ( $A$  and  $B$ ) and produce these logic functions. The minimum circuit structure is based on PTL circuits where a single-transistor AND logic is fully utilized. For instance, in Fig. 21a right, a pMOS pass transistor is responsible for the  $\bar{A}\bar{B}$  function, and an additional nMOS transistor is used for discharging operations. When the pMOS transistor receives voltages  $A$  and  $B$  at its gate and drain, respectively, the source voltage approaches  $\bar{A}B$  at equilibrium. If a pMOS

and 2-in 1-out (C21) gates and their corresponding circuits: (b) NOT; (c) AND, NOR, and OR; and (d) XOR and XNOR functions

transistor is turned off, an nMOS transistor connected between the pMOS transistor and the ground discharges the output node, which significantly increases the upper bound of the operation frequency. When  $A = B = "0"$ , the output voltage is not completely zero because of the threshold voltage of the MOS transistors; however, this small voltage shift is restored to logical “0” at the next input stage. Therefore additional level-restoring circuits are unnecessary for this circuit.

Figure 22 shows constructions of multiple-input logic functions with fusion gates. In classical circuits, two-input AND and OR gates consist of six transistors. To decrease the power supply voltage for low-power operation, a small number of transistors (three or less) should be on each unit’s current path. Since each unit circuit has six transistors,  $n$ -input AND and OR gates consist of  $6(n - 1)$  transistors ( $n \geq 2$ ). On the other hand, in fusion-gate logic [(a) and (b)], an  $n$ -input AND gate consisted of  $4(n - 1)$  transistors, whereas  $2(n + 1)$  transistors were used in an  $n$ -input OR gate. Therefore, in the case of AND logic, the



**Novel Hardware for Unconventional Computing,**  
**Fig. 22** Fusion-gate architectures of multiple-input functions (Yamada et al. 2008): (a) AND, (b) OR, and

(c) majority logic gates. Half and full adders are shown in (d, e), respectively

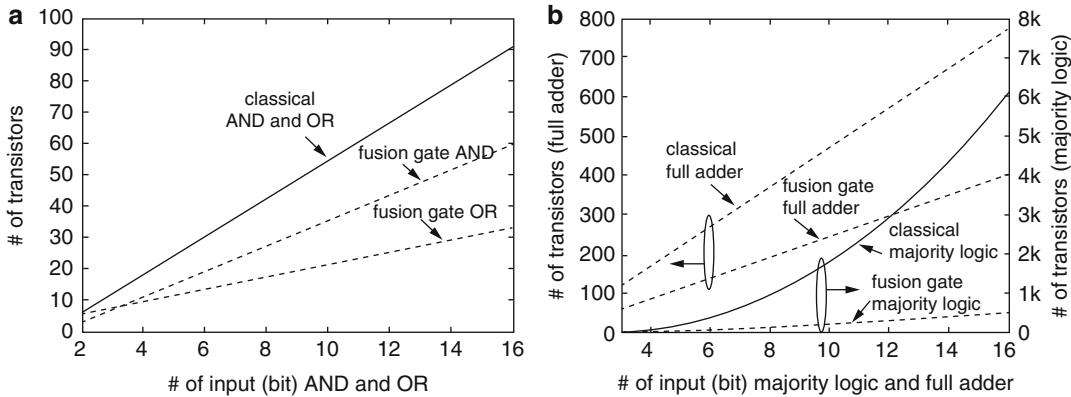
number of transistors in fusion-gate circuits is smaller than that of classical circuits. The difference will be significantly expanded as  $n$  increases. Figure 22c shows fusion-gate implementation of majority logic circuits with multiple inputs. Again, in classical circuits, the number of transistors on each unit's current path is fixed to three. For  $n$ -bit inputs ( $n$  must be an odd number larger than 3), the number of transistors in the classical circuit was  $30 + 36(n - 3)^2$ , while in the fusion-gate circuit, it was  $2(n^2 - n + 1)$ , which indicates that the collision-based circuit has a great advantage in the number of transistors. Half and full adders constructed by fusion-gate logic are illustrated in Fig. 22d and e. The number of transistors in a classical half adder was 22, while it was 10 in a fusion-gate half adder (Fig. 22d). For  $n$ -bit full adders ( $n \geq 1$ ), the number of transistors in a classical circuit was  $50n - 28$ , while it was  $26(n - 1) + 10$  in a fusion-gate circuit (Fig. 22e). Again, the fusion-gate circuit has a significantly smaller number of

transistors, and the difference will be increased as  $n$  increases.

Figure 23 summarizes the comparison of the number of transistors between classical and fusion-gate logic. The number of transistors in fusion-gate logic was always smaller than that of transistors in classical logic circuits, especially in majority logic gates.

## Future Directions

Before starting any computation, one should input data information in the RD medium. A parallel input is an essential feature of an edge-cutting parallel computing architecture. Serial inputs, so common for vast majority of massively parallel processors, dramatically decrease performance of the computing devices, particularly those operating in transducer mode, where information is constantly fed into the processor (e.g., in tasks of image processing).



**Novel Hardware for Unconventional Computing, Fig. 23** Total number of transistors in classical and collision-based (CB) multiple-input logic gates (Yamada et al. 2008)

Experimental RD chemical computers, at least in certain case, may well have analogs of parallel inputs. It has been demonstrated widely that by applying light of varying intensity, we can control excitation dynamic in BZ medium (Beato and Engel 2003; Flessels et al. 1998; Grill et al. 1996; Petrov et al. 1997), wave velocity (Schebesch and Engel 1998), and pattern formation (Wang 2001). Of particular interest are experimental evidences of light-induced back propagating waves, wave front splitting, and phase shifting (Yoneyama 1996); we can also manipulate medium's excitability by varying intensity of the medium's illumination (Brandtstadter et al. 2000). In fact, optical input of data information has been already used at the beginning of RD research (Kuhnert et al. 1989). This was proved to be particularly important in experiments in image processing in BZ-medium-based computing devices (Kuhnert et al. 1989; Rambidi 1998; Rambidi and Yakovenchuk 2001; Rambidi et al. 2002). We are not aware of any rigorous experimental results demonstrating a possibility of optical inputs of RD semiconductor devices; there is however a simulation-related evidence of a possibility of optical parallel inputs. This entry (Mohajerzadeh et al. 1994) discusses particulars of a photo-response, photon-induced generation of electron-hole pairs in *p-n-p-n* devices, depending on primary color components in the stimulating input because elementary processors can also act as color sensors.

There exists a possibility of parallel optical outputs on semiconductor devices. Technologies for integrating optoelectronic devices and electronic circuitry are fully developed, but limited hybrid integration is available commercially at present. An important problem of such integration is that pursuing it involves simultaneous development of sophisticated technologies for optoelectronic devices (III-V semiconductors) and silicon integrated circuits. Indeed, recent development in optoelectronic integrated circuits (OEICs) enables us to implement light-emitting devices (LEDs) on silicon substrate by controlling defects at III-V/silicon interface. For example, Furukawa et al. demonstrated that lattice-matched and defect-free GaPN epilayers can be grown on silicon with a thin GaP buffer layer (Furukawa et al. 2001). The task is enormous, and as a practical matter only small-scale OEICs have been demonstrated. While the integration levels of III-V OEICs have remained low, the degree of integration in commercial GaAs integrated circuits has reached LSI levels in recent years. These advances offer a route to achieving much higher levels of optoelectronic integration through epitaxial growth of III-V heterostructures on GaAs-based LSI electronics.

Most experimental prototypes of chemical excitable computing devices suffer from difficulties with representation of results of wave interaction. This is because being excited a medium's micro-volume becomes refractory and then

recovers back to the resting state. Excitation wave fronts usually annihilate in the result of collision.

Therefore, experimental excitable RD computers require some external devices, like digital camera, to record their spatiotemporal dynamics. Thus, e.g., to compute a collision-free path around obstacles in thin-layer BZ medium, one must record snapshots of BZ medium's activity and then analyze this series of snapshots to extract results of the computation (Adamatzky and De Lacy Costello 2002b; Agladze et al. 1997). This is the cost we pay for reusable (because the excitable medium eventually returns to resting state) RD processors. Another option of preserving results of computation may be to employ non-excitatory RD media, where a precipitate is formed (or do not form) in the result of diffusion wave interaction with a substrate (or competition of several wave fronts for the substrate) (Adamatzky and De Lacy Costello 2002a, 2003; Adamatzky and Tolmachiev 1997; De Lacy Costello and Adamatzky 2003). Precipitation is an analog of infinite memory. The feature is priceless and however makes experimental prototypes simply disposable; in contrast to excitatory media, the precipitate-forming media can be used just once. RD semiconductor computers, because they are essentially man-made devices, may allow us to combine reusability and rich space-time dynamics of excitable RD media with low post-processing costs of precipitate-forming RD media. This can be done by embedding a lattice of oscillatory semiconductor elements into a lattice of excitatory semiconductor elements. The excitatory elements (EEs) will form a substrate to support traveling excitation waves, while oscillatory elements (OEs) will play a role of rewritable memory. For example, to represent sites of wave front collision, we must adjust an activation threshold of OEs in such manner that front of a single wave will not trigger OEs; however, when two or more wave fronts collide, the OEs at the sites of collision are triggered and continue to oscillate even when the wave fronts annihilate.

What computational tasks can be realized in RD semiconductor devices? Most primitive

operations of image processing (see overview in Adamatzky 2001), e.g., detection of contour and enhancement, are straightforwardly mapped onto the silicon architecture. Silicon implementation of RD (precipitate formation based) algorithms for computational geometry – Voronoi diagram (Adamatzky and Tolmachiev 1997; De Lacy Costello and Adamatzky 2003; De Lacy Costello et al. 2004) and skeletonization (Adamatzky et al. 2002) – requires embedding of oscillatory elements in the lattice of excitatory *p-n-p-n* devices; the oscillating elements will represent bisectors of a Voronoi diagram and segments of a skeleton.

Universal computation can be realized in RD semiconductor devices using two approaches. Firstly, by employing collision-based mode of computation in excitable media (Adamatzky 1998), where functionally complete sets of Boolean gates are implemented by colliding waves. In collision-based mode, however, we must force the medium to be in a sub-excitatory regime which may pose some problems from fabrication point of view. Secondly, we can “physically” embed logical circuits, namely, their diagram-based representations, into the excitable medium (Steinbock et al. 1995; Tóth and Showalter 1995). In this we employ particulars of photo-response of *p-n-p-n* elements and project the logical circuit onto the medium as pattern of heterogeneous illumination.

## Bibliography

- Adamatzky A (1994) Reaction-diffusion algorithm for constructing discrete generalized Voronoi diagram. *Neural Netw World* 6:635–643
- Adamatzky A (1996) Voronoi-like partition of lattice in cellular automata. *Math Comput Model* 23:51–66
- Adamatzky A (1998) Universal dynamical computation in multi-dimensional excitable lattices. *Int J Theor Phys* 37:3069–3108
- Adamatzky A (2000) Reaction-diffusion and excitable processors: a sense of the unconventional. *Parallel Distrib Comput Theor Pract* 3:113–132
- Adamatzky A (2001) Computing in nonlinear media and automata collectives. Institute of Physics Publishing, Bristol
- Adamatzky A (ed) (2002) Collision-based computing. Springer, London

- Adamatzky A, De Lacy Costello BPJ (2002a) Experimental logical gates in a reaction-diffusion medium: the XOR gate and beyond. *Phys Rev E* 66:046112
- Adamatzky A, De Lacy Costello BPJ (2002b) Collision-free path planning in the Belousov-Zhabotinsky medium assisted by a cellular automaton. *Naturwissenschaften* 89:474–478
- Adamatzky A, De Lacy Costello BPJ (2003) On some limitations of reaction-diffusion computers in relation to Voronoi diagram and its inversion. *Phys Lett A* 309:397–406
- Adamatzky A, Tolmachev D (1997) Chemical processor for computation of skeleton of planar shape. *Adv Mater Opt Electron* 7:135–139
- Adamatzky A, De Lacy Costello B, Ratcliffe NM (2002) Experimental reaction-diffusion pre-processor for shape recognition. *Phys Lett A* 297:344–352
- Adamatzky A, Arena P, Basile A, Carmona-Galán R, De Lacy Costello B, Fortuna L, Frasca M, Rodríguez-Vázquez A (2004) Reaction-diffusion navigation robot control: from chemical to VLSI analogic processors. *IEEE Trans Circ Syst I* 51:926–938
- Adamatzky A, De Lacy Costello B, Asai T (2005) Reaction-diffusion computers. Elsevier, Amsterdam
- Agladze K, Magome N, Aliev R, Yamaguchi T, Yoshikawa K (1997) Finding the optimal path with the aid of chemical wave. *Phys D* 106: 247–254
- Alioto M, Palumbo G (2005) Model and design of bipolar and MOS current-mode logic: CML, ECL and SCL digital circuits. Springer, Berlin
- Asahi N, Akazawa M, Amemiya Y (1998) Single-electron logic systems based on the binary decision diagram. *IEICE Trans Electron* E81-C:49–56
- Asai T, Nishimiya Y, Amemiya Y (2002) A CMOS reaction-diffusion circuit based on cellular-automaton processing emulating the Belousov-Zhabotinsky reaction. *IEICE Trans Fundam* E85-A:2093–2096
- Asai T, Adamatzky A, Amemiya Y (2004) Towards reaction-diffusion computing devices based on minority-carrier transport in semiconductors. *Chaos Solitons Fractals* 20:863–876
- Asai T, Ikebe M, Hirose T, Amemiya Y (2005a) A - quadrilateral-object composer for binary images with reaction-diffusion cellular automata. *Int J Parallel Emergen Distrib Syst* 20:57–68
- Asai T, De Lacy Costello B, Adamatzky A (2005b) Silicon implementation of a chemical reaction-diffusion processor for computation of Voronoi diagram. *Int J Bifur Chaos* 15:3307–3320
- Asai T, Kanazawa Y, Hirose T, Amemiya Y (2005c) Analog reaction-diffusion chip imitating the Belousov-Zhabotinsky reaction with hardware Oregonator model. *Int J Unconv Comput* 1:123–147
- Beato V, Engel H (2003) Pulse propagation in a model for the photosensitive Belousov-Zhabotinsky reaction with external noise. In: Schimansky-Geier L et al (eds) Noise in complex systems and stochastic dynamics. *Proc SPIE* 5114:353–362
- Brandstädter H, Braune M, Schebesch I, Engel H (2000) Experimental study of the dynamics of spiral pairs in light-sensitive Belousov-Zhabotinskii media using an open-gel reactor. *Chem Phys Lett* 323:145–154
- Daikoku T, Asai T, Amemiya Y (2002) An analog CMOS circuit implementing Turing's reaction-diffusion model. *Proc Int Symp Nonlinear Theor Appl* 809–812
- De Lacy Costello B, Adamatzky A (2003) On multitasking in parallel chemical processors: experimental findings. *Int J Bifur Chaos* 13:521–533
- De Lacy Costello B, Adamatzky A, Ratcliffe N, Zanin AL, Liehr AW, Purwins HG (2004) The formation of Voronoi diagrams in chemical and physical systems: experimental findings and theoretical models. *Int J Bifur Chaos* 14:2187–2210
- Flessels JM, Belmonte A, Gáspár V (1998) Dispersion relation for waves in the Belousov-Zhabotinsky reaction. *J Chem Soc Faraday Trans* 94:851–855
- Fukai T (1996) Competition in the temporal domain among neural activities phase-locked to subthreshold oscillations. *Biol Cybern* 75:453–461
- Furukawa Y, Yonezu H, Ojima K, Samonji K, Fujimoto Y, Momose K, Aiki K (2001) Control of N content of GaPN grown by molecular beam epitaxy and growth of GaPN lattice-matched to Si(100) substrate. *Jpn J Appl Phys* 41:528–532
- Gerhardt M, Schuster H, Tyson JJ (1990) A cellular automaton model of excitable media. *Phys D* 46:392–415
- Gravert H, Devoret MH (1992) Single charge tunneling – Coulomb blockade phenomena in nanostructures. Plenum, New York
- Grill S, Zykov VS, Müller SC (1996) Spiral wave dynamics under pulsatory modulation of excitability. *J Phys Chem* 100:19082–19088
- Karahaliloglu K, Balkir S (2005) Bio-inspired compact cell circuit for reaction-diffusion systems. *IEEE Trans Circ Syst II Expr Brief* 52:558–562
- Kuhnert L, Agladze KL, Krinsky VI (1989) Image processing using light-sensitive chemical waves. *Nature* 337:244–247
- Kumakura K, Nakakoshi K, Motohisa J, Fukui T, Hasegawa H (1995) Novel formation method of quantum dot structures by self-limited selective area metalorganic vapor phase epitaxy. *Jpn J Appl Phys* 34:4387–4389
- Kumakura K, Motohisa J, Fukui T (1997) Formation and characterization of coupled quantum dots (CQDs) by selective area metalorganic vapor phase epitaxy. *J Cryst Growth* 170:700–704
- Kuwamura N, Taniguchi K, Hamakawa C (1994) Simulation of single-electron logic circuits. *IEICE Trans Electron* J77-C-II:221–228
- Matsubara Y, Asai T, Hirose T, Amemiya Y (2004) Reaction-diffusion chip implementing excitable lattices with multiple-valued cellular automata. *IEICE Electron Express* 1:248–252
- Mohajerzadeh S, Nathan A, Selvakumar CR (1994) Numerical simulation of a p-n-p-n color sensor

- for simultaneous color detection. *Sensors Actuators A* 44:119–124
- Muller DE (1954) Application of boolean algebra to switching circuit design and to error detection. *IRE Trans Electr Comput EC-3*:6–12
- Niedernosteide FJ, Kreimer M, Kukuk B, Schulze HJ, Purwins HG (1994) Travelling current density filaments in multilayered silicon devices. *Phys Lett A* 191:285–290
- Oya T, Asai T, Fukui T, Amemiya Y (2002) A majority-logic nanodevice using a balanced pair of single-electron boxes. *J Nanosci Nanotechnol* 2:333–342
- Oya T, Asai T, Fukui T, Amemiya Y (2005) Reaction-diffusion systems consisting of single-electron circuits. *Int J Unconv Comput* 1:123–147
- Petrov V, Ouyang Q, Swinney HL (1997) Resonant formation in a chemical system. *Nature* 388:655–657
- Rambidi NG (1998) Neural network devices based on reaction-diffusion media: an approach to artificial retina. *Supramol Sci* 5:765–767
- Rambidi NG, Yakovenchuk D (2001) Chemical reaction-diffusion implementation of finding the shortest paths in a labyrinth. *Phys Rev E* 63:0266071–0266076
- Rambidi NG, Shamayaev KE, Peshkov GY (2002) Image processing using light-sensitive chemical waves. *Phys Lett A* 298:375–382
- Reed IS (1954) A class of multiple-error-correcting codes and their decoding scheme. *IRE Trans Inf Theory PGIT* 4:38–49
- Schebesch I, Engel H (1998) Wave propagation in heterogeneous excitable media. *Phys Rev E* 57: 3905–3910
- Serrano-Gotarredona T, Linares-Barranco B (2003) Log-domain implementation of complex dynamics reaction-diffusion neural networks. *IEEE Trans Neural Netw* 14:1337–1355
- Shelar RS, Sapatnekar SS (2001) BDD decomposition for the synthesis of high performance PTL circuits. *Workshop Notes IEEE IWLS 2001* 298–303
- Soeleman H, Roy K, Paul BC (2001) Robust subthreshold logic for ultra-low power operation. *IEEE Trans VLSI Syst* 9:90–99
- Song M, Asada K (1998) Design of low power digital VLSI circuits based on a novel pass-transistor logic. *IEICE Trans Electron* E81-C:1740–1749
- Steinbock O, Toth A, Showalter K (1995) Navigating complex labyrinths: optimal paths from chemical waves. *Science* 267:868–871
- Suzuki Y, Takayama T, Motoike IN, Asai T (2007) Striped and spotted pattern generation on reaction-diffusion cellular automata: theory and LSI implementation. *Int J Unconv Comput* 3:1–13
- Tóth Á, Showalter K (1995) Logic gates in excitable media. *J Chem Phys* 103:2058–2066
- Tóth Á, Gáspár V, Showalter K (1994) Propagation of chemical waves through capillary tubes. *J Phys Chem* 98:522–531
- Wang J (2001) Light-induced pattern formation in the excitable Belousov-Zhabotinsky medium. *Chem Phys Lett* 339:357–361
- Yamada T, Akazawa M, Asai T, Amemiya Y (2001) Boltzmann machine neural network devices using single-electron tunneling. *Nanotechnology* 12:60–67
- Yamada K, Asai T, Hirose T, Amemiya Y (2008) On digital LSI circuits exploiting collision-based fusion gates. *Int J Unconv Comput* 4:45–59
- Yoneyama M (1996) Optical modification of wave dynamics in a surface layer of the Mn-catalyzed Belousov-Zhabotinsky reaction. *Chem Phys Lett* 254:191–196
- Young DA (1984) A local activator-inhibitor model of vertebrate skin patterns. *Math Biosci* 72:51–58



---

## Thermodynamics of Computation

H. John Caulfield and Lei Qian  
Fisk University, Nashville, TN, USA

### Article Outline

Glossary  
Definition of the Subject  
Introduction  
The Thermodynamics of Digital Computers  
Analog and Digital Computers  
Natural Computing  
Quantum Computing  
Optical Computing  
Cellular Array Processors  
Conclusions  
Future Directions  
Bibliography

### Glossary

**Analog computing** An analog computer is often negatively defined as a computer that is not digital. More properly, it is a computer that uses quantities that can be made proportional to the amount of signal detected. That analogy between a real number and a physical property gives the name “analog.” Many problems arise in analog computing, because it is so difficult to obtain a large number of distinguishable levels and because noise builds up in cascaded computations. But, being unencoded, it can always be run faster than its digital counterpart.

**Computing** Any activity with input/output patterns mapped onto real problems to be solved.

**Digital computing** The name derives from digits (the fingers). Digital computing works

with discrete items like fingers. Most digital computing is binary – using 0 and 1. Because signals are restored to a 1 or a 0 after each operation, noise accumulation is not much of a problem. And, of course, digital computers are much more flexible than analog computers that tend to be rather specialized.

**Entropy** A measure of the disorder or unavailability of energy within a closed system.

**First law of thermodynamics** Also known as the law of *energy conservation*. It states that the energy remains constant in an isolated system.

**Quantum computer** A quantum computer is a computing device that makes direct use of quantum mechanical phenomena such as superposition and entanglement. Theoretically, quantum computers can achieve much faster speed than traditional computers. It can solve some NP hard or even exponentially hard problem within linear time. Due to many technical difficulties, no practical quantum computer using entanglement has been built up to now.

**Second law of thermodynamics** The second law of thermodynamics asserts that the *entropy* of an isolated system never decreases with time.

**The Church-Turing thesis** A combined hypothesis about the nature of computable functions. It states that *any function that is naturally regarded as computable is computable by a Turing machine*. In the early twentieth century, various computing models such as Turing machine,  $\lambda$ -calculus, and recursive functions are invented. It was proved that Turing machine,  $\lambda$ -calculus, and recursive functions are equally powerful. Alonzo Church and Alan Turing independently raised the thesis that any *naturally computable function* must be a recursive function or, equivalently, be computed by a Turing machine or be a  $\lambda$ -definable function. In other words, it is not possible to build a computing device that is

more powerful than those machines with simplest computing mechanisms. Note that Church-Turing thesis is not a provable or refutable conjecture because “*naturally computable function*” is not a rigorous definition. There is no way to prove or refute it. But it has been accepted by nearly all mathematicians today.

**Thermodynamics** Literally accounting for the impact of temperature and energy flow. It is a branch of physics that describes how temperature, energy, and related properties affect behavior of an object or event.

**Turing machine** An abstract computing model that was invented by Alan Turing in 1936, long before the first electronic computer was invented, to serve as an idealized model for computing. A Turing machine has a tape that is unbounded in both directions, a read-write head and a finite set of instructions. At each step, the head may modify the symbol on the tape right under the head, change the state of the head, and then move on the tape one unit to the left or right. Although extremely simple, Turing machines can solve any problem that can be solved by any computers that could possibly be constructed (see **Church-Turing thesis**).

## Definition of the Subject

Thermodynamics is a very general field that helps to define the physics limits that impose on all stuff. When people study an abstract computing model such as a Turing machine, thermodynamics does not impact their behavior because nothing physical has been done. However, real computers must be made of some physical material, material that has a temperature. The physical stuff must have at least two readily distinguishable states that can be interpreted to two different values. In the process of computing, the stuff should be able to change the state under control. The change of the state must obey the laws of thermodynamics. For example, there is a minimum amount of energy associated with a bit of information. Information is limited by the

entropy change  $k \log 2$  and the minimum energy is  $kT \log 2$ , where  $k$  is the Boltzmann’s constant and  $T$  is the absolute temperature of the device and its surroundings. Mysteriously, information and entropy seem linked even though one is physical and the other is conceptual. Those limits are vital to know, as it allows designers to stop seeking improvements as they reach or near those limits. At the same time, thermodynamics indicates where computers do not come near what is possible. Such aspects of computers represent potentially important fields to explore.

## Introduction

Thermodynamics – the science of the changes in energy over time – is one of the most fundamental sciences. Arguably, the first and second laws of thermodynamics are the scientific principles most scientists would label as the most fundamental of all the laws of science. No one ever expects them to be violated. We will explore the following aspects of the relationship between thermodynamics and computing.

- The energetics and corresponding temporal properties of classical digital computers
- Possible modifications such as conservative computing, analog computing, and quantum computing
- Thermodynamically inspired computing

Unsurprisingly, we will find that those topics are all related.

Before we can do any of those things, however, we must review thermodynamics itself and do so in terms suited for this discussion.

## Thermodynamics

Thermodynamics originated as a set of general rules characterizing the effects of system parameters such as energy and temperature over time. Somewhat later, thermodynamics came to be partitioned into two broad classes – the classical thermodynamics (now called equilibrium

thermodynamics as it predicts the final system parameters that result from some perturbing effect) and nonequilibrium thermodynamics (what happens if energy input to a system continues over time so equilibrium is never obtained).

The fact that thermodynamics applies to quantities averaged over the system and deals only with probabilistic results means that no physical law is broken if entropy within the system decreases due to either of two reasons: the occurrence of an improbable event that is allowable but fleeting or the occurrence of a state that is not reachable by equilibrium thermodynamics but is reachable with continuing input of energy. The latter is an example of nonequilibrium thermodynamics. This also connects nonequilibrium thermodynamics with topics such as nonlinear dynamics, chaos, emergence, and so forth. Two concepts from nonlinear dynamics are especially interesting: attractor states and emergence. Attractor states are system states such that if the state of the system is near a given attractor (within its “attractor basin”), successive states tend to move toward that attractor. Some of the attractor states in systems far from equilibrium can be highly organized. The properties of highly organized nonequilibrium thermodynamic states can be fixed conditions, periodic events, or even more complex situations called strange attractors that are somewhat predictable and necessarily chaotic in the technical sense of that word. There is generally a control parameter such as power input which determines the system performance. For some values systems often develop excessive sensitivity to the values of the control parameter leading to long-term unpredictability of the deterministic result – a condition called chaos. Systems that are too unorganized like an ideal gas are not very interesting and form a behavior viewpoint. Systems that are fully organized like a crystal also have limited interest. But between those extremes lie regions of high complexity and great interest. One of the favorite and well-justified sayings of complexity theorists is that life arises on the edge of chaos. You are a system that shows spectacular organization if kept away from equilibrium with continuing

input of energy. The properties that arise far from equilibrium are said to be emergent and are generally unpredictable until discovered experimentally. A general feature of such organized complexity in systems maintained well away from equilibrium is self-organization. That is, the order observed does not have to be created by a scientist. Instead it arises spontaneously under the correct circumstances.

Turing machines can be accounted for by conventional thermodynamics, but some computer such as neural networks and cellular array processors can become complex enough to exhibit interesting and useful emergent properties that must be discussed in terms of nonlinear thermodynamics. Such self-organized complex processor behavior has also been called “synergistic.”

The most complex computer we know is the human brain. It not only arises on the edge of chaos but also uses and consumes chaos. Over-regular temporal behavior of the brain is called epilepsy and is not a desirable state.

Two of the first and most influential people to explore nonequilibrium thermodynamics were Warren Weaver and Ilya Prigogine.

Weaver (1948) discussed three ways people can understand the world: simple systems, statistical methods (conventional thermodynamics), and nonequilibrium organized complexity [thermodynamics]. Organized complexity is compatible with classical thermodynamics from an overall perspective but can be achieved locally in space and time.

Prigogine’s Nobel Prize-winning work on nonequilibrium thermodynamics is explained beautifully in his popular books (Nicolis and Prigogine 1977; Prigogine and Stengers 1984; Prigogine et al. 1986).

Let us now review very briefly what thermodynamics deals with and why it is important to computers.

Temperature,  $T$ , is a measure of the driving force for energy. There can be no systematic energy flow from a lower-temperature region to a higher-temperature region.

Entropy,  $S$ , is a measure of the disorder. The second law of thermodynamics, perhaps science’s most sacred law, says that entropy never

decreases. Indeed, it nearly always increases. Note that entropy is essentially an abstract quantity. Indeed, it is the negative of information as defined by Shannon. Therefore, information is sometimes called negentropy – negative entropy. There is a metaphysical puzzle here that has been more ignored than solved. How does a mathematical concept like information come to govern physics as does entropy?

Boltzmann's constant  $k$  symbolizes the relationship between information and physics. It appears in purely physical equations such as the ideal gas law. It also appears in equations about information. Leo Szilard suggested that writing a memory does not produce entropy; it is possible to store information into a memory without increasing entropy. However, entropy is produced in every case that the memory is *erased*. It turns out that the (minimum) entropy created by erasing one bit is given by

$$S \text{ per erased bit} = k \ln 2,$$

and the number  $\ln 2 \approx 0.69$  is the natural logarithm of 2. What's going on here? How are information and the physical world related?

Information relates to the answers to yes or no questions. Entropy is physical entity not necessarily related to questions and answers. Yet the mathematics of both is the same. Information is the negative of entropy. What are we to make of this metaphysically? One view (originating in the mind and writing of John Archibald Wheeler) is that information is primary. The slogan is “It from Bit.” Wheeler's metaphysics is extremely appealing, because it requires no external scaffolding. But it strikes many as not quite convincing. There is an opposite approach that begins with stuff and energy. The physical world is made of relationships between parts. The universe is self-organized and self-defined. Stuff is differentiation (yes and no) within the whole. Another view is that physics (stuff, energy, motion) is primary. Information in a brain or a computer is present only if some stuff is used and some energy is expended in generating, transmitting, or storing it. In that case, information relates to the probability that a given record would appear by chance if all records were

equally probable. The less likely the chance appearance, the more information is present. In this way, information ties back to Bayes' theorem. Neither caricature of a position is complete enough to be fair, but perhaps this footnote will stimulate some readers to dig deeper. This is not a metaphysics chapter, so we take no side in this dispute. But, some of us are physicists and physicists are drawn to physics by the hope of finding answers not just equations. We cannot help asking such questions, so we think noting this unresolved metaphysical problem is appropriate. Note also that no metaphysical problem can ever be frilly resolved, as the criteria for truth of metaphysics lie in the domain of metametaphysics (Ockham's razor, etc.) and so on ad infinitum.

There is a fundamental relationship between the change  $\Delta E$  in the physical quantity energy and the change  $\Delta S$  in the mathematical quantity entropy, namely:

$$\Delta E = T \Delta S.$$

What is remarkable about thermodynamics is that it tells us nothing about how things actually work. It tells us instead about what cannot be done. It is the science of fundamental limits. It is, in that sense, a negative field. Physicists sometimes have to work very hard to find the detailed ways nature enforces these limits in any particular problem.

When energy ceases to have any net flow, the system is said to be at equilibrium. Until the latter part of the twentieth century, thermodynamics meant near-equilibrium behavior. Far from equilibrium, very interesting things can occur. Order can arise from disorder at least locally. The second law is satisfied by creating even more disorder somewhere in the universe. Life exists far from equilibrium and seems to increase order. Indeed, it does – locally. But it must take energy from its environment and cause a global increase in energy.

The laws of thermodynamics define and thus forbid magic – something from nothing (the first law) or effects without causes (the second law). We will interpret all magic forbidding laws as “thermodynamic” even if they apply to nonphysical things such as information.

## Computer Equivalents of the First and Second Laws

We propose two laws of information inspired by the two laws of thermodynamics just described.

**First law: Information is not increased in a computer.** A computer cannot create information. The output result is strictly determined by the data and the instructions. If we were smart enough and had unlimited resources (time and memory), we would know that  $1 + 2 + 3 + 4 + 5 = 15$ . We only need a computer, because computer can compute faster and more reliably. We may not have known the answer now, but we could if we had enough time, memory, and patience. The result is not new but derived information.

**Second law: The energy required to destroy one bit of information is not smaller than the temperature  $T$  multiplied by some minimum amount of information  $\Delta S$ .** This second law applies for conservative as well as dissipative logic – terms to be described later. This second law, like the second law of thermodynamics, deals with irreversible acts.

## The Thermodynamics of Digital Computers

Just as all digital computers are equivalent to a Turing machine, all computations in digital computers can be thought of as being performed by Boolean logic gates. If we understand the thermodynamics of a Boolean logic gate, we understand the thermodynamics of any digital computer.

All readers of this chapter will recognize that Boolean logic gates such as *AND*, *OR*, *NOR*, and so forth convert two one-bit inputs into a single one-bit output. Immediately, we notice:

- A Boolean logic gate has less information out than we put in.
- Such a logic gate is irreversible.

From the two laws of computing, we conclude:

- Since information must be conserved, one bit of information must somehow have been converted into some noninformative state.
- The amount of energy required is related to the information content of a single bit multiplied by the operating temperature  $T$ .

The person who first realized and determined the amount of energy needed to destroy one bit of information is

$$\Delta E = kT\log 2$$

was an IBM scientist named Landauer. Here  $k$  is the Boltzmann constant. The average energy of a molecule at temperature  $T$  is about  $kT$ , so this makes some intuitive sense as well.

This is really a very small amount of energy – molecules of that energy are striking you right now and you can't even feel them. Two things make this energy remark important. First, most current gates require millions of  $kTs$  per operation. This is no accident. It is required as we will see later when we discuss analog computing. Second, the number of operations per second is now in the billions. So trillions of  $kTs$  are dissipated in current computers each second. They get very hot and consume a great amount of power.

Landauer and fellow IBM physicist Bennett then took a totally unexpected and brilliant direction of research. What would happen, they asked, if we could use logic gates that do not destroy information? Then there would be no thermodynamic minimum energy requirement for such a gate's operation. Perhaps the energy cost of computing could be reduced or even eliminated.

A trivial example from arithmetic will illustrate these concepts for readers not familiar with them.

Here is a conventional (dissipative) arithmetic expression:

$$2 + 3 = 5.$$

It would be more informative to write

$$2 + 3 \Rightarrow 5.$$

If we know the data (2, 3) and the instructions (+), the result (5) is determined. But this is irreversible. If we know the result (5) and the instruction to be inverted (+), we cannot recover the data (2, 3). The data could have been (3, 2), (1, 4), etc. Here is an information conserving arithmetic expression:

$$2, 3^{\circledR} 3, 2.$$

The operator  $^{\circledR}$  means reverse the order of the data. Such an operation is clearly conservative of information. And, clearly, it is reversible:

$$3, 2^{\circledR} 2, 3.$$

An  $^{\circledR}$  gate would have no inevitable energy price. It destroys no information (our first law is satisfied).

Here is a not such trivial example. Consider the operation:

$$a, b, \odot a + b, a - b$$

This operation is also conservative of information because with  $a + b$  and  $a - b$ , one can easily recover the original  $a$  and  $b$ .

A more complicate example is cryptography. A sequence  $p$  of characters is encrypted (by, say DES algorithm) to  $s = \text{enc}_k(p)$  (assume the key  $k$  is fixed). It is obvious that  $p$  can be recovered from  $s$ .

Since that time, a number of conservative, reversible logic gates have been proposed and implemented. We can say some things in general about such gates.

- They will have equal numbers of inputs and outputs (usually three).
- The changes effected by the gates must either rearrange the data (like the  $^{\circledR}$  operator in our arithmetic example) or mix them (like the  $\odot$  in the second example). All electronic gates so far proposed involve rearrangements. In optical logic, reversible mixing is common.

Stringing conservative logic gates together can allow us to produce sought after results.

Inevitably, however, we will also have computed things we do not care about – called garbage.

Garbage is inevitable in reversible computing, because the components must retain the information normally destroyed by a conventional Boolean logic gate. Reversibility eliminates one problem (the inevitable dissipation of information) but creates another (the inevitable appearance of garbage bits).

Nevertheless, we must detect the desired answer, and that must cost at least  $kT \log 2$ . The zero-energy possibility applies only to the logic operations, not the entering of data into the system or the extraction of data from the system. Failure to understand this has caused more than a few to proclaim that zero-energy logic operations are impossible.

We enter data at the input ports and wait for an answer to reach the output ports. As the system is reversible, not all occurrences lead in the “right direction.” Temperature-dependent random walk will eventually produce the answer. But with a little energy bias in the forward direction, we will get the answer faster at the cost of expending energy – the more energy, the faster the results are likely to arrive. One might expect that something akin to the energy-time uncertainty principle might apply here, so very low energy gates might require a very long time to operate. Indeed, that has usually been the case. An exception will be noted in the case of certain optical elements.

## Analog and Digital Computers

Because digital computers and computation have been so successful, they have influenced how we think about both computers as machines and computation as a process – so much so, it is difficult today to reconstruct what analog computing was all about... It is a history in which digital machines can do things ‘better’ and ‘faster’ than other machines... However, what is at stake here are not matters of speed or precision Rather, it is an argument about what can be rendered and understood through a machine that does computation. (Nyce 1996)

All computers involve the operation of some physical system and the interpretation of

physically measurable quantities in numerical terms. The most obvious interpretation is

$$N = cQ.$$

The number  $N$  is the measured quantity  $Q$  multiplied by some conversion coefficient. In classical physics,  $Q$  is a continuous variable. Computers using such numbers are called analog computers.

Analog computers have several profound disadvantages relative to digital computers.

- **Sensitivity.** By this we mean the ability to distinguish small difference in large numbers. Analog computers tend to have trouble being able to distinguish, say, 47 from 48. Digital computers only have to distinguish between 0 and 1. It encodes bigger numbers in terms of 0s and 1s.
- **Inability to compute complex or even real numbers in most cases,** because most useful quantities  $Q$  are usually nonnegative. This is usually “solved” by encryption of real or complex numbers as multiple nonnegative numbers.
- **Cumulative errors.** When the output from one analog computation is used as an input to another, the errors are carried forward and accumulated. This effect is made even worse by the fact that some problems amplify the input errors significantly. In linear algebra problems, do instance, that error amplification factor is called the condition number of the matrix. That number is never less than 1 and often exceedingly large (ill-conditioned matrices) or even infinite (singular matrices).
- **Inflexibility.** This chapter was written using a word processor in a digital computer. Making an analog word processor would require almost inconceivably complex.

Some distinct advantages come from being continuous in space or time (Orponen 1997).

In addition, it is widely believed that anything an analog computer can compute can also be computed by a digital computer. In fact, that belief is a corollary of what is called the strong

Church-Turing thesis. It asserts that anything that can be computed in any way (including analog computations) can also be computed by a Turing machine. The more widely accepted form of the Church-Turing thesis simply asserts that anything that is sensibly computable by any computer can be computed by a Turing machine. There are some who believe they have discovered exceptions to the Church-Turing thesis and many more who doubt that. This is not the place to examine that controversy. It is mentioned here for two reasons.

First, it suggests that analog computers cannot do anything a digital computer cannot do. There is no advantage in principle. There may be practical advantages to analog computing when it suffices as we have already suggested, though. But there is much more to computing than simply getting the right answer. Speed, cost, and power consumption are examples. Sometimes, analog computing can win on those criteria.

Second, the Church-Turing thesis is thermodynamic in the generalized sense that it sets presumably immutable constraints but does not provide information on how those constraints are enforced.

So why would anyone bother with analog computers? There are many reasons. Here are some obvious ones:

- A binary gate is an analog gate followed by a nonlinear operator (a quantizer) that sets low signals to 0 and high signals to 1 or vice versa. So the difference between analog and binary computing lies in two things – the number of quantization levels (2 for binary and many more for analog) and how often measurements and corrections are made (after each operation for binary computers and after the whole computation for analog). So, we can argue that all computers are analog at their hearts.
- But encryption in terms of many binary operations, they are inevitably slower and more power hungry than digital computers. We could digitize to more than two values. It takes far fewer ternary values than binary values to represent a large number, but the more quantization levels, the more probable

it is that we will make an error. It is generally assumed that two is the optimum number overall.

But there are more subtle effects that sometimes favor analog over digital.

Consider the representation of physically measured time signal in a digital computer. The sampling theorem tells us how frequently we must sample the signal if we know its bandwidth. But suppose we do not know. We can do either of two dangerous things: apply a band-pass filter and risk losing information or guess the bandwidth and risk producing aliasing. So discretization of an analog signal has a significant probability of giving misleading results.

Iterative digital solution of linear algebra problems converge only for condition numbers limited by the digitization itself, while analog solutions have no such limit because the condition number itself is undefined.

A more general treatment can be found in Pour-El and Richards (1989).

For chaotic systems, discretization can lead to wildly different results with different levels of discretization (Grantham and Amitm 1990; Herbst and Ablowitz 1989; Ogorzalek 1997).

So what can we say about the thermodynamics of analog computers? The operation occurs in three steps:

1. **Preparation.** The apparatus is set up so that the input parameters are the ones we choose and the physics will carry out the desired transformations.
2. **Time evolution.** The physical system evolves through time.
3. **Readout.** The desired physical parameters are measured.

Preparation is necessary in digital computing as well and has not been treated in any thermodynamic discussion. Nor will we treat it here. The last two steps must somehow be governed by the two laws of computing propounded earlier. That is, the system cannot produce what we interpret as information beyond that we inserted in the preparation. The underlying physics may be

exceedingly complex, but the informatics must be as described by the first law. The readout requires some minimum energy. In a digital computer with  $L$  levels, the Landauer analysis says we must dissipate  $kT \log L$  in energy. Here is at least a qualitative explanation of the fact that we need far more than  $kT$  to operate a binary logic gate. We need to measure to far more than just two levels in order to make the binary decisions with sufficiently low error rate.

Analog computing is an emerging field, in part because of interest in what is called “natural computing.”

## Natural Computing

In serial non-Von Neumann machines, the interconnections between processors have negligible impact on system space and energy consumption. But concurrency (e.g., parallelism) requires more interconnections and more attention to their space and energy consumption. And those problems become worse as the number of concurrent processors increases. For large enough numbers of processors, interconnections are no longer negligible and can become limiting.

Two approaches have been proposed to ameliorate such problems: optical interconnection and quantum computing. Optical interconnection may be useful, because optical paths can cross in space without effect. That is, many optical paths can share the same space. Quantum computing is of interest because  $N$  entangled states can probe  $2^N$  paths. Both avoid massive interconnection with wires.

Richard Feynman anticipated both approaches. One of Feynman’s goals in suggesting quantum computing was to solve the wiring problem. Another motivation also guided him. He knew that the accuracy with which the results of a quantum experiment could be predicted depended on the number of path integrals evaluated, so (in a sense) each natural event in quantum mechanics is equivalent to infinitely many calculations. The only way to calculate quantum events with perfect accuracy would be to use a quantum computer.

Let us describe a natural computation. We begin with a physical system  $P$  that operates on

an input  $I_N$  to produce an output  $O_N$ . It does this naturally without human assistance.

Now suppose we wish to use that physical process P to calculate some result. If there is a computer operation C that operates on an input  $I_C$  to produce an output  $O_C$ , and if there are mappings,

$$\begin{aligned} M1 : I_N &\leftrightarrow I_C \\ M2 : O_N &\leftrightarrow O_C \end{aligned}$$

then we can declare that the P and C are congruent and thus compute the same results if the diagram in Fig. 1 commutes.

That is, if nature in doing what it will do automatically allows us to establish the starting state  $I_N$  (state preparation) in a way that can be viewed using M1 as representing the computer input  $I_C$  and if the output  $O_N$  of the natural process and if M2 carries that to the very output that a conventional computer produces with input, then the two processes are congruent and we have a natural computer.

Certainly, the most widely studied form of natural computing at this moment is the quantum computer.

## Quantum Computing

The behaviors of matter in very small scale obey laws of quantum mechanics, which are so different from what we usually experience as to be

### Thermodynamics of Computation,

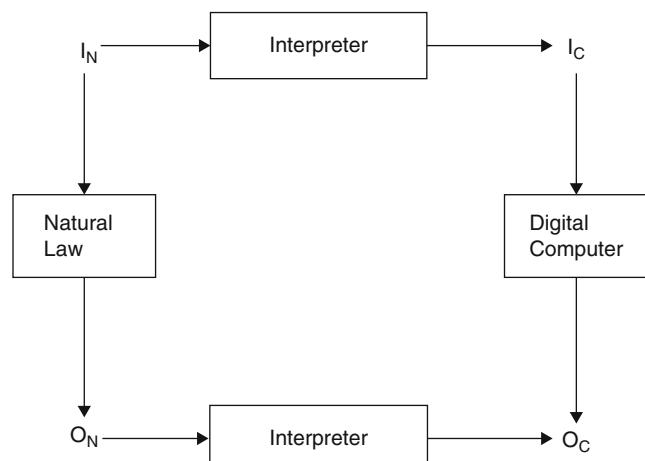
**Fig. 1** Nature converts input  $I_N$  into output  $O_N$  by natural law. The digital computer performing some fixed algorithm converts input  $I_C$  into output  $O_C$ . If, for all allowable  $I_N$ , we have a rule that converts it to an equivalent  $I_C$  and if the output of the natural computer  $I_N$  can be converted to the output  $I_C$ , then we have in the physical operator a “natural computer”

essentially beyond our understanding. We can write the equations and perform the experiments but the results inevitably seem weird. Despite this, it is possible to harness some of these effects for computing. Quantum computers usually utilize superposition (a principal of quantum mechanism which claims that while the state of any object is unknown, it is actually in all possible states simultaneously, as long as we don't measure it) and quantum entanglement (a quantum mechanical phenomenon in which the quantum states of two or more objects have to be described with reference to each other, even though the individual objects may be spatially separated) to perform computation (Nielsen and Chuang 2000; Bouwmeester et al. 2001).

In traditional digital computers, information is carried by numerous binary digits, which are called bits. In quantum computing, information is carried by particles which carry quantum states. Such quantum logic bit is called qubit. A qubit can be expressed by

$$|i\rangle = a|0\rangle + b|1\rangle$$

where  $|0\rangle$  and  $|1\rangle$  are two orthonormal states. The coefficients a and b are the normalized probability amplitudes (in general, complex-valued numbers) of states  $|0\rangle$  and  $|1\rangle$  that satisfy the condition  $|a|^2 + |b|^2 = 1$ . These two orthonormal states can be polarization in orthogonal directions, two oppositely directed spins of atoms, etc. It



seems that a qubit carries partially information of  $|0\rangle$  and partially information of  $|1\rangle$ . However, when a qubit is measured (which is necessary if we want to read the output), the qubit “collapses” to either state 0 or state 1 with the probabilities  $|a|^2$  or  $|b|^2$ . This process is irreversible. Whence a qubit collapses to a state, there is no way to recover the original state. On the other hand, it is also not possible to clone a quantum state without destroying the original one.

A very weird but extremely useful phenomenon in quantum world is *entanglement*. It is the coherence among two or more objects even though these objects are spatially separated. Consider a 2-qubit state:  $(|0\rangle|0\rangle + |0\rangle|1\rangle + |1\rangle|0\rangle + |1\rangle|1\rangle)/\sqrt{2}$ . It can be decomposed to  $((|0\rangle + |1\rangle)/\sqrt{2})((|0\rangle + |1\rangle)/\sqrt{2})$ . That means we have 50% probability to read  $|0\rangle$  and 50% probability to read  $|1\rangle$  when we measure the first qubit. We will get same probability for the second qubit. The measured value of the first qubit is independent of that of the second qubit. However, consider the state:  $1/\sqrt{2}(|0\rangle|1\rangle - |1\rangle|0\rangle)$ . It is easy to prove that one cannot decompose the state into a form  $(a|0\rangle + b|1\rangle)(c|0\rangle + d|1\rangle)$ . Actually, if we measure the first qubit, we have 50% probability to read  $|1\rangle$  and 50% chance to read  $|0\rangle$ . However, if we read  $|0\rangle$  when the first qubit is measured, we *always* read  $|1\rangle$  for the second qubit; if we read  $|1\rangle$  for the first qubit, we *always* read  $|0\rangle$  for the second qubit. The measurement to one of them will affect the state of the other qubit immediately no matter how far they are from each other.

Consider a quantum system with  $n$  qubits. If there is no entanglement, it can be decomposed into the product of  $n$  1-qubit systems or  $2n$  independent numbers. However, if entanglement is allowed, then there are  $2^n$  independent coefficients and therefore can hold  $2^n$  independent values. So it can operate tremendous number of values in parallel. It is possible to solve exponentially hard or NP complete problems in polynomial or even linear time. In 1994, Peter Shor of AT&T devised an algorithm for factoring an integer  $N$  in  $O((\log N)^3)$  time and  $O(\log N)$  space with a quantum computer. The security of

RSA, the most widely used public-key cryptography method, is based on the assumption that factoring a large integer is not computationally feasible. A quantum computer can break it easily.

Even though quantum computers are much more powerful than traditional digital computers in terms of speed, it still obeys Church-Turing thesis. In other words, no quantum computer can solve a problem that a traditional computer cannot. Quantum computers can still be simulated by traditional computers. The difference is that a traditional computer may have to spend millions of years to solve a problem that a quantum computer can solve in seconds. Therefore, the first law of computing – information cannot increase in a computer – is still true because a quantum computer has the same power as a traditional computer in terms of computability.

Just as Boolean logic gates play a central role in traditional computers, quantum logic gates are the heart of quantum computers. Unlike Boolean logic gates, all quantum logic gates are reversible. More precisely, a quantum logic gate can be represented by a unitary matrix  $M$  (an  $n \times n$  matrix  $M$  is unitary if  $M^*M = I_n$  where  $M^*$  is the conjugate transpose of  $M$ ). So if  $M$  is the matrix that represents the quantum logic gate,  $M^*$  represents the reverse operation of the gate.

While it is possible to define quantum logic gates for any number of qubits, the most common quantum logic gates operate on spaces of one or two qubits. For example, Hadamard gate and phase shifter gates are single-qubit gate and CNOT (controlled-NOT) gate is a two-qubit gate. With Hadamard gates, phase shifter gates, and CNOT gates, one can do any operation that can be done by a quantum computer. We call sets of quantum gates like this by *universal quantum gates*.

Because quantum logic gates are all reversible, quantum computers do not destroy any information until outputs are measured. Therefore, quantum computers may consume less energy than traditional computers in theory.

While quantum computers have tremendous power and may consume much less energy, they are extremely difficult to be built. A quantum computer must maintain coherence among its

qubits long enough to perform an algorithm. Preventing qubits from interacting with the environment (which will cause decoherence) is extremely hard. We can get around the difficulty by giving up the entanglements. However, the quantum computer is no longer as powerful if we do so. Without entanglements, the state space of  $n$  qubits only has  $2n$  dimensions rather than  $2^n$  dimensions. Exponentially hard or NP complete problems cannot be computed by these computers within polynomial time (assume  $P \neq NP$ ) (Caulfield and Qian 2006). However, these computers use reversible logic gates and therefore have advantage in energy consuming.

In conclusion, even though quantum computers use a quite different model, two laws of information still apply to them.

## Optical Computing

Optical computing has been studied for many decades (Caulfield et al. 2006). This work has led to conclusions that seem obvious in retrospect, namely, that:

- Optical computing can never compete with electronic computing for general purpose computing.
- Optical computing has legitimate roles in niches defined by the characteristic that the information being processed is already in the optical domain, e.g., in optical communication or optical storage.
- Two types of complexity arise. One is the complexity of the optical effects themselves, e.g., optics often has three or more activities going on at each location, so if the processes are nonlinear (e.g., photorefractives or cross talk among active elements), then complexity factors are likely to arise (Caulfield et al. 1999).

This is nonequilibrium thermodynamics at work. Not to be confused with this are the savings in temporal coherence accomplished by use of

space, time, and fan-in complexity to consume computational complexity (Caulfield et al. 1991; Caulfield 1992).

Optical computing is limited by thermodynamics in a unique sense. Let us suppose that we wish to measure an output signal that is either 1 or 0 in amplitude. Either for speed or power consumption, we seek to measure as few photons as possible to decide which of those two states (0 or 1) is present. Because the photons do not (without encouragement) come evenly spaced in time of arrival, we must detect many photons to be sure that no detected photon means a logical zero rather than a statistical extreme when a logical 1 is intended. With normal statistics, it takes about 88 detected photons to distinguish between what was intended as a logical 1 and what was intended as a zero. It is possible to make a light source with substantially uniform emission rates. This is called squeezed light. Naively, squeezing can overcome that problem to an amazing extent, but the squeezed state is very fragile, and many of the central activities in an optical computer destroy the order squeezing put in before it can be detected (Shamir et al. 1991).

## Thermodynamically Inspired Computing

The approach of systems to equilibrium may become a model for computing. The most prominent of these are called simulated annealing, the Hopfield net, and the Boltzmann machine. Basically, they are ways to encode a quantity to be minimized as a sort of pseudo energy and then simulating some energy minimization activity. And, of course, hybridization of these methods makes the clean division among them impossible.

Annealing of materials is an ancient art. Domains in a metal sword, for instance, may be randomly disposed to one another, making them brittle. The goal of annealing is to get them aligned, so they behave more like a single crystal and thus hardened. This is done by a method sometimes called “heat and beat.” The material is heated to a high temperature where domain changes are fairly easy and then beat with a hammer to encourage such movement. Then the material is cooled to a little lower temperature where one hopes the gains so far are not

rerandomized and beat again. The art is in the cooling schedule and in the beating. In thermodynamic terms, we are trying to get the sword in its minimum energy state where no beating against the opponent will cause it to relax into a lower energy state. Unlike more familiar optimization methods such as steepest descent or Newton-Raphson that find only a local extremum (one near the starting state), simulated annealing aims at finding the global extremum independently of starting state. It does this by pure imitation of annealing. Some monotonic figure of merit for the problem is chosen. It might be the RMS difference between the output vector of a matrix-vector multiplication and a target vector. Minimizing it is analogous to minimizing the sword's energy. Driving it to or near zero by adjusting the input vector is a way of solving the ubiquitous  $\mathbf{Ax} = \mathbf{b}$  problem for  $\mathbf{x}$ . In early stages, big random steps (chosen from a thermodynamic expression governing probabilities as a function of energy and "temperature") are used. The perturbed system is then "cooled" in the sense of allowing the starting fictitious  $T$  value to decrease slightly, and the process is repeated. This is very computer intensive but very effective. More information can be found in many places including Kirkpatrick et al. (1983), Cerny (1985), and Das (2005).

A Hopfield net (Hopfield 1982) is very similar to simulated annealing in that it uses guided stochastic methods to minimize a user-defined energy (figure of merit). It is based on adjusting connections of a rather simple binary-connected neural network by picking a starting state and adjusting one interconnection at a time.

A Boltzmann machine is a more thermodynamically inspired version of a Hopfield network (Hinton and Sejnowski 1986).

One more thermodynamically inspired means for computation needs to be mentioned, namely, stochastic resonance (Benzi et al. 1983). It is a creative use of noise. Noise is usually thought of as the enemy of signal. But if in some nonlinear operation the signal is so small that it is not useful, adding a certain amount of noise to the input may help. Indeed, there is an optimal amount of noise for each case. Too little or too

much noise is both bad, but there is a noise value in resonance with the problem which gives optimal results.

This is not the place for detailed discussions of these thermodynamically inspired methods. Rather, our intent is to show that thermodynamics has impacted not only physical computers but also the way those computers are used to achieve certain goals.

## Cellular Array Processors

Cellular array processors (some prefer "cellular automata") are discrete binary-valued elements in fixed locations or arrays (Ilachinski 2001). Usually started with random inputs, they can be observed to produce quite interesting space-time histories as each cell is updated according to a fixed rule using its current value and those of its immediate neighbors. Under some circumstances, those space-time patterns simply terminate or generate chaos. Often, however, the system self-organizes to produce highly organized patterns. One "application" is the so-called artificial life which began with the immensely popular set of evolution rules for a two-dimensional array called "the Game of Life." For the few who have not played with it, we urge you to do so. An online version is available free at <http://www.bitstorm.org/gameoflife/>. Many brilliant computer scientists have become enthralled with it. For instance, Stephen Wolfram analyzed it in terms of thermodynamics (Wolfram 1983).

## Conclusions

### Thermodynamics

- Governs all computing
- Has inspired conservative, reversible computing
- Is not easy to apply to non-Turing computers such as analog and quantum
- Has directly inspired some optimization methods

In the end, computers are at least partially physical in the classical sense, so they are ultimately limited by thermodynamics. The presence of many virtual operations in some forms of quantum computing can reduce the price of physicality dramatically but cannot eliminate it (Kupiec and Caulfield 1991).

## Future Directions

Classical equilibrium thermodynamics places limits on what can be accomplished under different conditions.

More importantly, it tells us when we are approaching a limit state. Things often get exceedingly difficult in such cases, so effort might be better spent where there is greater room for improvement.

Optical computing may allow the reaching of the energy required for reversible computing soon. As there is no fundamental limit but negative energy is not defined, we can say that zero-energy computing is the limit. Doing the computation by interferometry of beams passing through passive optics requires no energy. At last, we will have an example of how the choice of medium can be critical. That kind of zero-energy computation is not possible for electronics. Early versions of zero-energy gates have already been described and made (Caulfield et al. 2007a, b). That work might not have been undertaken except for the “permission” for zero-energy systems granted by thermodynamics.

## Bibliography

### Primary Literature

- Benzi R, Parisi G, Sutera A, Vulpiani A (1983) A theory of stochastic resonance in climatic change. *SIAM J Appl Math* 43:565–578
- Bouwmeester D, Ekert A, Zeilinger A (2001) The physics of quantum information. Springer, Berlin
- Caulfield HJ (1992) Space – time complexity in optical computing. *Multidim Syst Sig Process* 2:373–378
- Caulfield HJ, Qian L (2006) The other kind of quantum computing. *Int J Unconv Comput* 2(3):281–290

- Caulfield HJ, Brasher JD, Hester CF (1991) Complexity issues in optical computing. *Opt Comput Process* 1:109–113
- Caulfield HJ, Kukhtarev N, Kukhtareva T, Schamschula MP, Banarjee P (1999) One, two, and three-beam optical chaos and self organization effects in photorefractive materials. *Mater Res Innov* 3:194–199
- Caulfield HJ, Vikram CS, Zavalin A (2006) Optical logic redux. *Opt Int J Light Electron Opt* 117:199–209
- Caulfield HJ, Soref RA, Qian L, Zavalin A, Hardy J (2007a) Generalized optical logic elements – GOLEs. *Opt Commun* 271:365–376
- Caulfield HJ, Soref RA, Vikram CS (2007b) Universal reconfigurable optical logic with silicon-on-insulator resonant structures. *Photonics Nanostruct* 5:14–20
- Cerny V (1985) A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *J Optim Theory Appl* 45:41–51
- Das A, Chakrabarti BK (eds) (2005) Quantum annealing and related optimization methods, vol 679, Lecture Notes in Physics. Springer, Heidelberg
- Grantham W, Amitm A (1990) Discretization chaos – feedback control and transition to chaos. In: Control and dynamic systems, vol34. Advances in control mechanics. Pt. 1 (A91-50601 21–63). Academic, San Diego, pp 205–277
- Herbst BM, Ablowitz MJ (1989) Numerically induced chaos in the nonlinear Schrödinger equation. *Phys Rev Lett* 62:2065–2068
- Hinton GE, Sejnowski TJ (1986) Learning and relearning in Boltzmann machines. In: Rumelhart DE, McClelland JL, the PDP Research Group (eds) Parallel distributed processing: explorations in the microstructure of cognition vol 1. Foundations. Cambridge MIT Press, Cambridge, pp 282–317
- Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. *Proc Natl Acad Sci U S A* 79:2554–2558
- Ilachinski A (2001) Cellular automata: a discrete Universe. World Scientific, Singapore
- Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. *Science* 220(4598):671–680
- Kupiec SA, Caulfield HJ (1991) Massively parallel optical PLA. *Int J Opt Comput* 2:49–62
- Nicolis G, Prigogine I (1977) Self-organization in nonequilibrium systems. Wiley, New York
- Nielsen MA, Chuang IL (2000) Quantum computation and quantum information. Cambridge University Press, New York
- Nyce JM (1996) Guest editor’s introduction. *IEEE Ann Hist Comput* 18:3–4
- Ogorzalek MJ (1997) Chaos and complexity in nonlinear electronic circuits. *World Sci Ser Nonlinear Sci Ser A* 22
- Orponen P (1997) A survey of continuous-time computation theory. In: Du DZ, Ko KI (eds) Advances in algorithms, languages, and complexity. Kluwer, Dordrecht, pp 209–224
- Pour-El MB, Richards JI (1989) Computability in analysis and physics. Springer, Berlin

- Prigogine I, Stengers I (1984) Order out of chaos. Bantam Books, New York
- Prigogine I, Stengers I, Toffler A (1986) Order out of chaos: man's new dialogue with nature. Flamingo, London
- Shamir J, Caulfield HJ, Crowe DG (1991) Role of photon statistics in energy-efficient optical computers. *Appl Opt* 30:3697–3701
- Weaver W (1948) Science and complexity. *Am Sci* 36:536–541
- Wolfram S (1983) Statistical mechanics of cellular automata. *Rev Mod Phys* 55:601–644
- Bernard W, Callen HB (1959) Irreversible thermodynamics of nonlinear processes and noise in driven systems. *Rev Mod Phys* 31:1017–1044
- Bub J (2002) Maxwell's Demon and the thermodynamics of computation. arXiv:quant-ph/0203017
- Casti JL (1992) Reality rules. Wiley, New York
- Deutsch D (1985) Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc R Soc Lond Ser A Math Phys Sci* 400:97–117
- Karplus WJ, Soroka WW (1958) Analog methods: computation and simulation. McGraw Hill, New York
- Leff HS, Rex AF (eds) (1990) Maxwell's demon: entropy, information, computing. Princeton University Press, Princeton
- Zurek WH (1989) Algorithmic randomness and physical entropy. *Phys Rev Abstr* 40:4731–4751

### Books and Reviews

- Bennett CH (1982) The thermodynamics of computation a review. *Int J Theor Phys* 21:905–940



## Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications

Evgeny Katz  
Department of Chemistry and Biomolecular  
Science, Clarkson University, Potsdam, NY, USA

### Article Outline

#### Abbreviations

- Introduction
- Enzyme-Based Logic Gates and Short Logic Circuits
- Decreasing Noise in the Enzyme-Based Logic Systems: Filters Producing Sigmoid Response Functions
- Interfacing of the Enzyme Logic Gates with Various Signal-Transducing Systems
- Digital Biosensors Based on Enzyme Logic Gates
- Conclusions and Perspectives
- Bibliography

#### Keywords

Biocomputing; Enzymes; Logic gates; Transduction methods; Bioanalytical techniques

### Abbreviations

Abs	Optical absorbance
ABTS	2,2'-Azino- <i>bis</i> (3-ethylbenzothiazoline-6-sulphonic acid) (chromogenic substrate used to follow peroxidase activity)
ABTS <sub>ox</sub>	Oxidized ABTS (colored product)
Ac	Acetic acid
AcCh	Acetylcholine

This chapter is based partially on recently published review articles (Katz 2015, 2017; Katz et al. 2017) with the copyright permissions from Elsevier, Wiley-VCH, and Springer. The compiled text was updated and edited.

AcChE	Acetylcholinesterase (enzyme; EC 3.1.1.7)
ADH	Alcohol dehydrogenase (enzyme; EC 1.1.1.1)
ADP	Adenosine 5'-diphosphate
AFM	Atomic force microscope (microscopy)
AGS	Amyloglucosidase (enzyme; EC 3.2.1.3)
Ala	Alanine (amino acid)
Ald	Acetaldehyde
ALT	Alanine transaminase (enzyme; EC 2.6.1.2)
Asc	Ascorbic acid
ATP	Adenosine 5'-triphosphate
BuCh	Butyrylcholine
ChO	Choline oxidase (enzyme; EC 1.1.3.17)
CK	Creatine kinase (enzyme; EC 2.7.3.2)
CN	4-Chloro-1-naphthol
CN-ox	CN insoluble oxidized product
Crt	Creatine
DC	Direct current
DHA	dehydroascorbic acid (product of oxidation of ascorbic acid)
Diaph	Diaphorase (enzyme; EC 1.8.1.4)
DNA	Deoxyribonucleic acid
EIS	Electrolyte–insulator–semiconductor
Et-O-Ac	Ethyl acetate
EtOH	Ethanol
FET	Field-effect transistor
Frc	Fructose
G6PDH	Glucose 6-phosphate dehydrogenase (enzyme; EC 1.1.1.49)
GDH	Glucose dehydrogenase (enzyme; EC 1.1.1.47)
Glc	Glucose
Glc6P	Glucose-6-phosphate
Glc6PA	Gluconate-6-phosphate acid (product of Glc6P oxidation)
GlcA	Gluconic acid
Glu	Glutamate (amino acid, salt form)
GluOx	Glutamate oxidase (enzyme; EC 1.4.3.11)

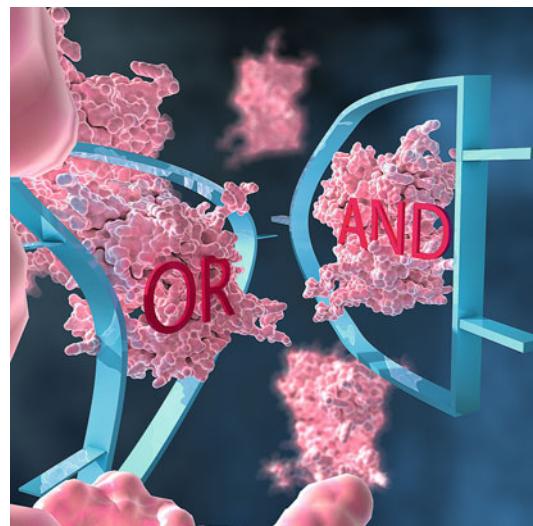
GOx	Glucose oxidase (enzyme; EC 1.1.3.4)	$R_{et}$	Electron transfer resistance (measured by Faradaic impedance spectroscopy)
HK	Hexokinase (enzyme; EC 2.7.1.1)		
HRP	Horseradish peroxidase (enzyme; EC 1.11.1.7)	RNA	Ribonucleic acid
Inv	Invertase (enzyme; EC 3.2.1.26)	SPR	Surface plasmon resonance
$I_p$	Peak current (measured with cyclic voltammetry)	TBI	Traumatic brain injury
IR	Infrared	TMB	3,3',5,5'-Tetramethylbenzidine (chromogenic substrate used to follow peroxidase activity)
ITO	Indium tin oxide (electrode)	UV	Ultraviolet
Lac	Lactate	$V_a$	Alternative voltage applied between the conducting support and reference electrode of the EIS devise
LDH	Lactate dehydrogenase (enzyme; EC 1.1.1.27)		
LSPR	Localized surface plasmon resonance	$V_{bias}$	Constant (bias) voltage applied between the conducting support and reference electrode of the EIS devise
Luc	Luciferase (enzyme from ATP assay kit, Sigma-Aldrich)		
Lucif	Luciferin		
MP-11	Microperoxidase-11	$V_{FB}$	Flat band voltage of the EIS device
MPh	Maltose phosphorylase (enzyme; EC 2.4.1.8)	W/O	Water-in-oil Pickering emulsion
$NAD^+$	Nicotinamide adenine dinucleotide		
NADH	Nicotinamide adenine dinucleotide reduced	$\alpha$ -KTG	$\alpha$ -Ketoglutaric acid
NPs	Nanoparticles		
O/W	Oil-in-water Pickering emulsion		
P2VP	Poly(2-vinylpyridine)		
P4VP	Poly(4-vinylpyridine)		
PB	Prussian blue		
PEO	Poly(ethylene oxide)		
PEP	Phospho(enol)pyruvic acid (or phosphoenolpyruvate in the form of salt)		
Pi	Inorganic phosphate		
PK	Pyruvate kinase (enzyme; EC 2.7.1.40)		
Ppy	Polypyrrole		
Ppy-ox	Polypyrrole-oxidized state		
Ppy-red	Polypyrrole-reduced state		
PQQ	Pyrroloquinoline quinone		
PS	Polystyrene		
Pyr	Pyruvate		
QCM	Quartz crystal microbalance		
R	Reflectance measured by SPR		
$R_{cell}$	Ohmic resistance measured in a bulk solution in an electrochemical cell		
RE	Reference electrode		

## Introduction

Exponential development of computing systems based on silicon materials and binary algorithms formulated as the “Moore’s law” (Moore 1998) is coming to the end being limited by further component miniaturization and by the speed of operation. In order to keep fast progress in computer technology, novel ideas from area of unconventional computing (Calude et al. 2009) are needed, including research in quantum computing (Mermin 2007) and biologically inspired molecular computing (Katz 2012a). Molecular computing systems, generally motivated by mimicking natural biological information processing, are not necessarily based on biomolecules and could be represented by synthetic molecules with signal-controlled switchable properties (de Silva 2007; Katz 2012b; Pischel 2010; Szacilowski 2008, 2012). However, despite progress achieved in assembling synthetic molecular systems performing basic Boolean operations and simple computations, these systems have limited

complexity, and further increase of their complexity is very challenging (Pischel et al. 2013). A new advance in the development of molecular information systems has been achieved with the use of biomolecular species (Benenson 2012; Katz 2012a), borrowing some ideas from systems biology (Alon 2007). Indeed, assembling very sophisticated systems from biomolecules, using their natural bioaffinity and complementarity, is much easier than increasing complexity of synthetic organic molecules with switchable functions. The biomolecular computing systems mimicking various information processing steps were assembled from DNA/RNA (Ezziane 2006; Stojanovic and Stefanovic 2011; Stojanovic et al. 2014), oligopeptides (Ashkenasy et al. 2011), proteins (Unger and Moult 2006), enzymes (Katz 2015; Katz and Privman 2010), and even whole biological cells (Rinaudo et al. 2007). The obvious advantage of the biomolecular computing systems is their ability to be integrated in artificially designed complex-reacting processes (Privman and Katz 2015) mimicking multistep information processing networks, also operating in biological environment (Kahan et al. 2008). It should be noted, however, that the complexity of the biocomputing systems is still far from reaching the level of the silicon-based electronic systems. While in a long perspective, a “biocomputer” might become a reality (Benenson 2009), particularly for some special applications, e.g., for solving complex combinatorial problems (Adleman 1994), the present level of technology does not allow any practical use of biomolecular systems for real computational applications. Presently developed biomolecular computing systems are represented by logic elements mimicking their electronic counterparts, thus filling a toolbox for future applications. However, researchers (and even more funding agencies) are anxious to achieve some practical results already in the near future. For achieving any practical result soon, some other applications, different from making a biocomputer, should be considered using the biomolecular systems with a limited complexity.

One of the immediate possible applications for biomolecular logic systems is a special kind of biosensing where the multiple input signals are logically processed through biochemical reactions resulting in YES/NO decisions in the binary (0,1) format (Katz et al. 2012). In this subarea of biomolecular logic systems, practical results are already possible at the present level of the system complexity, particularly for biomedical applications (Wang and Katz 2010, 2011), e.g., for logic analysis of injury biomarkers (Halámková et al. 2010a, b; Halámková et al. 2012; Zhou et al. 2011). This chapter is concentrated on enzyme-based logic systems (Katz and Privman 2010), introducing readers to the design of logic gates and circuits, illustrating their biosensing applications and demonstrating their interfacing with various transducers.



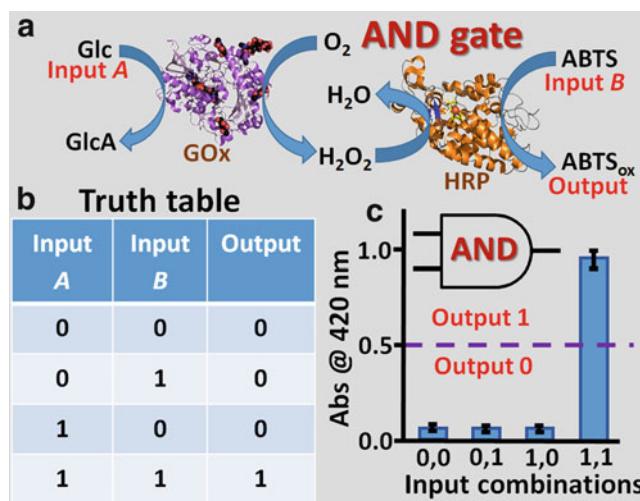
Artistic vision of the enzyme-based logic gates

## Enzyme-Based Logic Gates and Short Logic Circuits

Enzyme-based logic gates are usually realized through relatively simple enzyme-catalyzed reactions (Katz and Privman 2010). Rapid progress in enzyme-based information processing systems has resulted in the design of biocatalytic cascades

mimicking various Boolean logic gates, including AND (Bakshi et al. 2013; Baron et al. 2006; Halámk et al. 2012; Privman et al. 2013a; Strack et al. 2008a), OR (Strack et al. 2008a; Zavalov et al. 2012), NAND (Zhou et al. 2009a), NOR (Baron et al. 2006; Zhou et al. 2009a), controlled NOT (CNOT) (Moseley et al. 2014), XOR (Baron et al. 2006; Halámk et al. 2011a; Privman et al. 2010a; Strack et al. 2008a), INHIBIT (Baron et al. 2006; Strack et al. 2008a), Identity (Baron et al. 2006), and Inverter (Baron et al. 2006) gates. In order to digitalize chemical processes, the reacting species considered as logic input signals were applied at two levels of their concentrations: their physical absence (zero concentration) was defined as logic **0** input, while logic **1** input was defined as experimentally optimized and conveniently high concentration, thus allowing significant separation in the produced output signals when inputs **0** and **1** were applied in different combinations. Depending on specific needs set by applications, the input signals were defined as variable concentrations of substrates and/or cofactors reacting with enzymes (Baron et al. 2006) or different concentrations of the biocatalytic enzymes added to a “soup” of substrates/cofactors being ready to react with the enzymes (Strack

et al. 2008a). The non-variable part of the system is considered as a “machinery” operating with the variable input signals applied in various combinations. Figure 1 exemplifies the AND gate realization in an enzyme system. In this system the constant (non-variable “machinery”) part is represented by two enzymes, glucose oxidase (GOx) and horseradish peroxidase (HRP), being in a solution at specific experimentally optimized concentrations, while the logic (variable) input signals were represented by glucose (Glc) and 2,2'-azino-bis(3-ethylbenzothiazoline-6-sulphonic acid) (ABTS), Input *A* and Input *B*, respectively (Fig. 1a). The inputs were defined as logic value **0** in the absence of Glc and ABTS, while their arbitrary, experimentally optimized, concentrations were defined as logic value **1**. In the presence of Glc and ABTS (**1,1** input combination), the following biocatalytic reactions proceed. Glucose (Glc) is biocatalytically oxidized to gluconic acid (GlcA) and the concomitant reaction results in production of hydrogen peroxide ( $H_2O_2$ ); note that  $O_2$  is always present being also a part of the system “machinery.” This reaction is biocatalyzed by GOx. The produced  $H_2O_2$  reacts with ABTS to yield its oxidized color form ( $ABTS_{ox}$ ) in the reaction biocatalyzed by HRP.



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 1 (a)** A biocatalytic cascade mimicking Boolean AND logic gate. (b) Truth table corresponding to the AND logic gate. (c) The bar chart showing optical absorbance

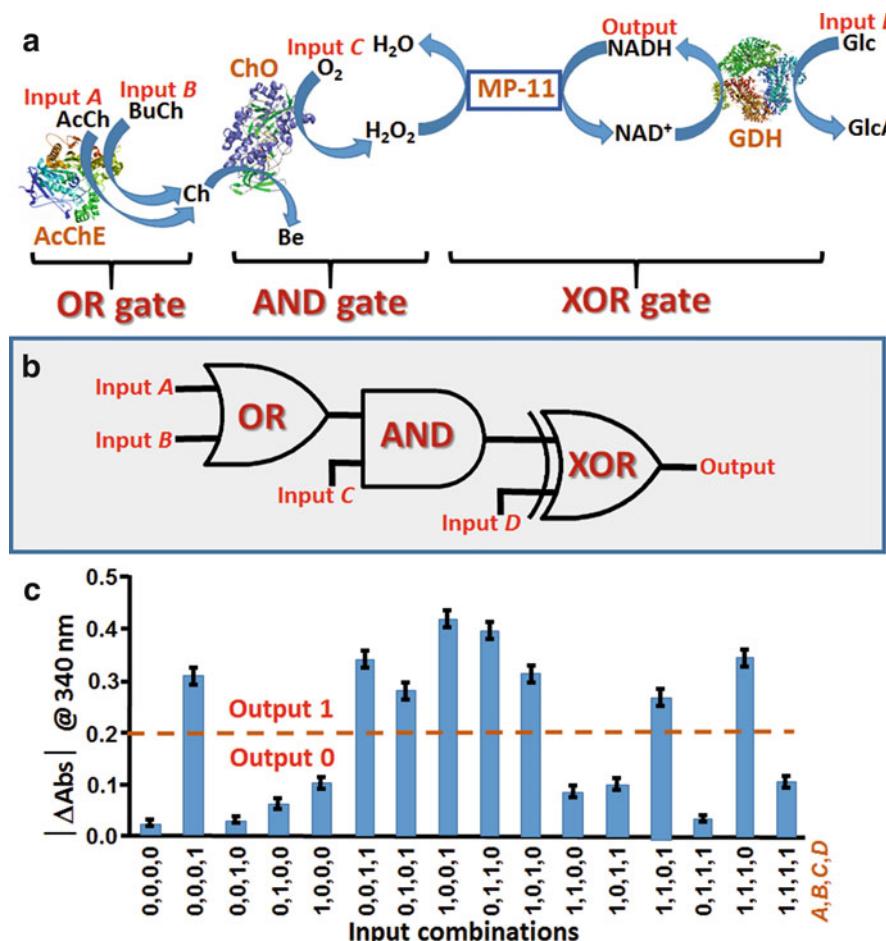
corresponding to  $ABTS_{ox}$  produced upon the application of the input signals in different combinations. The dash line separates **0** and **1** values of the output signal (Adopted with permission from Katz et al. 2017)

This two-step biocatalytic cascade proceeds only in the presence of both reacting species: Glc and ABTS. The absence of any or both of them (input signal combinations **0,0; 0,1; 1,0**) does not allow reactions to proceed till the end, thus inhibiting formation of the final colored ABTS<sub>ox</sub> product. Figure 1b shows the truth table of the process corresponding to the Boolean AND logic, while Fig. 1c shows essentially the same in the form of the experimentally observed absorbance changes produced in the course of the biocatalytic reactions in the presence of the input signals applied in various combinations: **0,0; 0,1; 1,0;** and **1,1.** The optical absorbance change at  $\lambda = 420$  nm associated with the formation of ABTS<sub>ox</sub> was considered as the output signal, being at the logic values **0** and **1** when the absorbance was measured below and above the threshold value of 0.5, respectively (Fig. 1c). Overall, the process is mimicking the Boolean AND logic gate where the output signal **1** is only achieved when the input signals are applied in **1,1** combination. All other Boolean logic gates have been realized in a similar way through different enzyme-catalyzed reactions (Katz 2015; Katz and Privman 2010).

Multistep biocatalytic cascades activated by several (more than two) input signals have been assembled to mimic logic circuits composed of concatenated logic gates (Niazov et al. 2006; Privman et al. 2009a, 2013b; Strack et al. 2008b). Different reaction steps were representing various Boolean logic gates. Figure 2a exemplifies a biocatalytic cascade mimicking the logic circuit composed of OR, AND, and XOR concatenated logic gates (Niazov et al. 2006). In the first reaction biocatalyzed by acetylcholinesterase (AcChE), two inputs represented by acetylcholine (AcCh) and butyrylcholine (BuCh) are hydrolyzed to yield choline (note that both reactions result in the formation of the same product, thus representing an OR gate). Then, produced in situ choline is oxidized in the presence of oxygen through the reaction biocatalyzed by choline oxidase (ChO) (note that this reaction proceeds only in the presence of both reacting species: choline and O<sub>2</sub>, thus representing an AND gate). The produced in situ H<sub>2</sub>O<sub>2</sub> oxidizes NADH through the reaction catalyzed by microperoxidase-11 (MP-11). Another logic input represented by glucose (Glc) reduces NAD<sup>+</sup> to yield NADH in the reaction catalyzed by glucose

dehydrogenase (GDH). Overall, the NADH/NAD<sup>+</sup> balance is preserved without changes if H<sub>2</sub>O<sub>2</sub> is not produced through the concatenated OR-AND gates, and Glc is applied at logic value **0** (meaning its absence). The NADH/NAD<sup>+</sup> ratio is also unchanged when both reactions, reducing NAD<sup>+</sup> and oxidizing NADH, are balanced, meaning appearance of H<sub>2</sub>O<sub>2</sub> through the biocatalytic cascade and application of Glc at the logic value **1.** The balanced/unbalanced reactions mimic a Boolean XOR logic gate, where the final output is defined as the absolute value of the NADH absorbance changes (note that in this definition, the absorbance increase and decrease corresponding to the NADH formation and consumption, respectively, means essentially the same). Figure 2b shows the equivalent logic circuit corresponding to the biocatalytic cascade depicted in Fig. 2a. Figure 2c demonstrates the bar chart with the experimentally obtained absorbance changes measured at 340 nm corresponding to the formation/consumption of NADH, which was considered as the final output signal produced by the enzyme system in response to the four input signals applied at 16 different variations. Due to the system complexity, the output signal appeared at different intensities, and a threshold of 0.2 was used to separate low and high signals corresponding to **0** and **1** logic values, respectively. Various reaction cascades have been designed to mimic different combinations of concatenated logic gates; however, they usually do not include more than three or four logic steps. Due to the noise formation and amplification through the reaction steps, the number of logic steps is limited, and theoretical estimation limits the system complexity by approximately ten logic steps (which have never been realized experimentally in those enzyme-biocatalyzed reactions) (Privman et al. 2008).

Figure 3a shows another interesting example of an enzyme-based biocatalytic system mimicking three concatenated AND gates used to activate an electrochemical system for releasing DNA and activating a DNA-based computing system (Mailloux et al. 2015). In other words, the system was used to integrate two different kinds of biomolecular computing systems based on enzyme and DNA reactions through an electrochemical interface. The reaction cascade was started by two

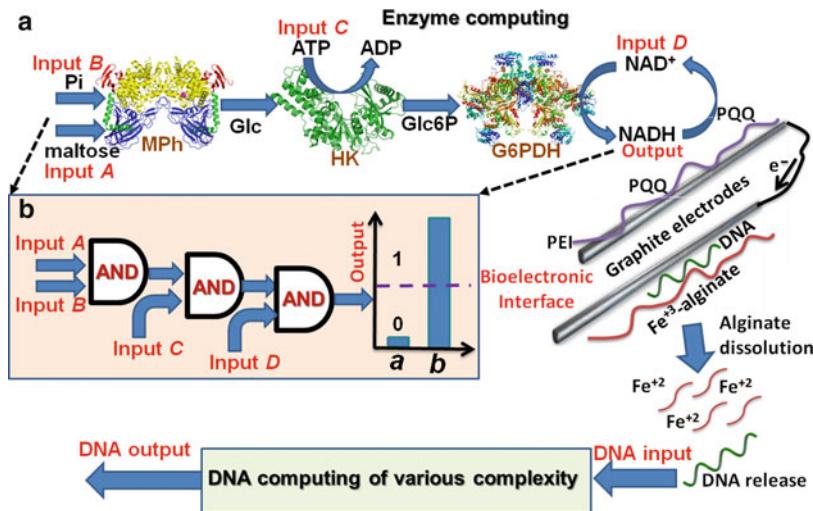


**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 2 (a)**  
A biocatalytic cascade mimicking operation of concatenated OR–AND–XOR gates activated by four biomolecular input signals. (b) Equivalent logic circuit corresponding to the biocatalytic cascade shown in (a). (c) Optical absorbance changes (absolute values, not

taking into account the directions of the changes) corresponding to the NADH concentrations obtained with the application of four input signals in 16 different combinations. The dash line separates **0** and **1** values of the output signal (A part of this figure was adapted from Niazov et al. 2006 with permission)

inputs, maltose and inorganic phosphate (Pi), reacting with maltose phosphorylase (MPH) and resulting in the production of glucose. Then, glucose was reacted with ATP (another logic input) in the reaction biocatalyzed by hexokinase (HK) to yield glucose-6-phosphate (Glc6P). Finally, Glc6P reduced NAD<sup>+</sup> (one more logic input) producing NADH in the reaction biocatalyzed by glucose 6-phosphate dehydrogenase (G6PDH). The completion of the reaction cascade was only possible in the presence of all logic inputs applied at logic value **1**, thus representing three concatenated AND

logic gates (Fig. 3b). The produced *in situ* NADH (the final output of the enzymatic reactions) was electrocatalytically oxidized at a pyrroloquinoline quinone (PQQ)-modified graphite electrode yielding a negative potential of ca. -60 mV (vs. Ag/AgCl reference). This electrode was electrically connected to another modified electrode coated with a thin film of Fe<sup>3+</sup>-cross-linked alginate with entrapped DNA. The reductive potential on the PQQ-modified electrode produced in the presence of NADH resulted in the reduction of Fe<sup>3+</sup> and dissolution



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 3 (a)** A biocatalytic cascade mimicking operation of three concatenated AND gates interfaced with a PQQ-modified graphite electrode. The NADH oxidation electrocatalyzed by PQQ resulted in the reductive dissolution of the  $\text{Fe}^{3+}$ -cross-linked alginate on the connected electrode and the concomitant release of DNA and then served as an input

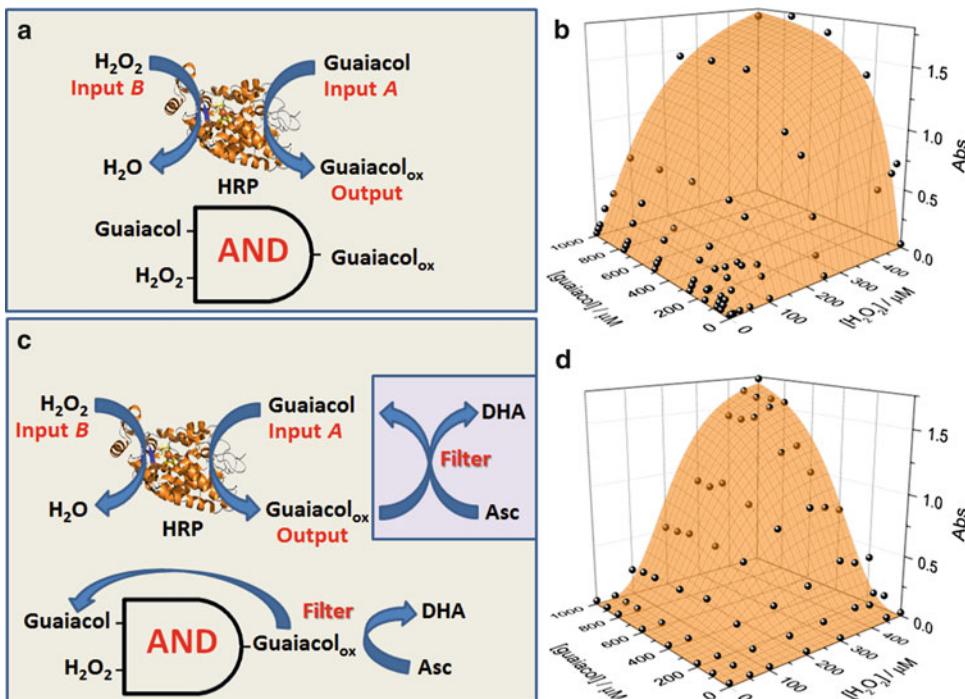
signal for the downstream DNA computing system (not shown in the figure). **(b)** Equivalent logic circuit corresponding to the biocatalytic cascade sketched in (a). The bar chart shows low (*a*) and high (*b*) output signal corresponding to the logic output **0** and **1**, respectively (A part of this figure was adapted from Mailloux et al. 2015 with permission)

of the alginate gel film, thus, resulting in the DNA release; note that produced  $\text{Fe}^{2+}$  is not capable of alginate cross-linking (Jin et al. 2012). The released DNA was used for activating various DNA-based logic gates (Guz et al. 2016; Mailloux et al. 2015).

### Decreasing Noise in the Enzyme-Based Logic Systems: Filters Producing Sigmoid Response Functions

In analog sensors/biosensors, a high slope of the response function (signal vs. analyte concentration) is a positive feature corresponding to a high sensitivity. However, this feature in digital systems with YES/NO (**1,0**) output, particularly in the areas near **0** and **1** inputs, means high analog noise produced by the digital system (Melnikov et al. 2009; Privman et al. 2008). Figure 4a shows a reaction biocatalyzed by horseradish peroxidase (HRP) where guaiacol is oxidized by  $\text{H}_2\text{O}_2$  to yield a colored product analyzed by the optical absorbance changes. The system represents a

simple realization of an AND logic gate with two reacting species operating as variable input signals. When the response function (optical absorbance) was measured for different concentrations of  $\text{H}_2\text{O}_2$  and guaiacol, the convex shape was found demonstrating a high slope of the signal increase, particularly at low concentrations of the reacting species (Fig. 4b). This means that small fluctuations of the input concentrations near logic **0** point would result in large changes of the output signal, thus producing high analog noise. The convex response shape is typical for most of chemical processes, particularly in biocatalytic reactions, where the reaction rate is increasing with the elevated concentrations of the reacting species. On the other hand, the system does not produce much noise at higher concentrations of the reacting species (near logic value **1**), when the biocatalytic reaction reaches saturation, thus resulting in flattening of the response function. The noise at the logic value **0** can be decreased by decreasing the slope of the response function at low concentrations of the reacting species. In other words, the reaction should be



#### Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 4 (a)

A biocatalytic reaction mimicking an AND logic gate where the output 1 signal (oxidation of guaiacol observed as the increase of the optical absorbance) was obtained only in the presence of both input signals, guaiacol and  $\text{H}_2\text{O}_2$ . (b) Response-surface mapping of the output signal for different concentrations of the applied input signals. Note the convex shape of the response-surface pattern with the sharp signal increase at low concentrations of the reacting species. (c) The same biocatalytic cascade as

shown in (a) but with the addition of a “filter” system returning a fraction of the output signal back. (d) Response-surface mapping of the output signal for different concentrations of the applied input signals. Note that the system operation with the added “filter” changes the original convex response-surface pattern to the sigmoid shape, thus decreasing the output signal at low concentrations of the reacting species and reducing the analog noise at the logic input 0 (A part of this figure was adapted from Bakshi et al. 2013 with permission)

inhibited at the low concentrations. This can be achieved by modifying the process with an additional step consuming the product of the reaction at its low concentration. Addition of ascorbic acid (Asc) to the system resulted in the conversion of the reaction product (oxidized guaiacol) back to the initial reduced form (Fig. 4c), thus converting the response function to the sigmoid shape (Fig. 4d). Since Asc was added in the limited (optimized) amount, after full consumption of Asc, the process limiting the output signal formation was stopped, and the output signal was increased, finally reaching the saturated value. The additional reaction step inhibiting the output signal formation at low concentrations of the input

signals is considered as a “filter” function, similarly to electronic processes. The analog noise decrease with the use of different “filters” was achieved in the operation of various enzyme-based logic gates (AND, OR) converting the convex response function to the sigmoid shape (Halámek et al. 2012; Pita et al. 2011; Privman et al. 2010b, 2013b, 2014; Zavalov et al. 2012). This modified response was particularly important for decreasing noise in the multistep systems operating as circuits with concatenated logic gates (Privman et al. 2013c) and even more important for the realization of digital biosensing systems (Katz et al. 2012; Wang and Katz 2010, 2011) considered later in this chapter.

## Interfacing of the Enzyme Logic Gates with Various Signal-Transducing Systems

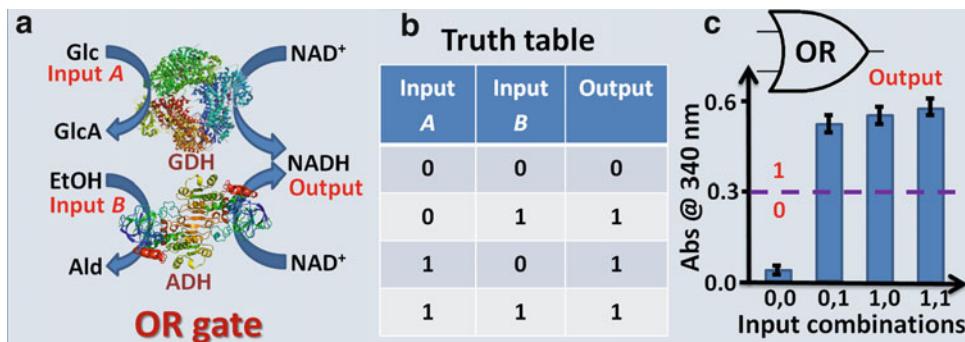
### Optical Analysis of the Output Signals Generated by Enzyme Logic Systems

Optical analysis of chemical products in enzyme-biocatalyzed reactions is frequently used in standard enzyme assay tests, particularly for the assay of enzymatic reactions biocatalyzed by NAD<sup>+</sup>/NADH-dependent dehydrogenases (e.g., lactate dehydrogenase (LDH)), oxidases (e.g., glucose oxidase (GOx)), or peroxidases (e.g., horseradish peroxidase (HRP)) (Bergmeyer 1974). The LDH-biocatalyzed production or oxidation of NADH (note that the reaction is reversible) can be easily followed by measuring optical absorbance at  $\lambda = 340$  nm characteristic of NADH (Passonneau and Lowry 1993). The same optical analysis can be performed for the reaction biocatalyzed by any other NAD<sup>+</sup>/NADH-dependent enzyme. The GOx-biocatalyzed glucose oxidation can be analyzed through the analysis of the concomitant product H<sub>2</sub>O<sub>2</sub> (Wilson and Turner 1992). This reaction is usually coupled to the HRP-biocatalyzed formation of a colored product in the presence of H<sub>2</sub>O<sub>2</sub> (Dunford 1991). The chromogenic substrates used in the HRP reaction are usually 2,2'-azino-bis(3-ethylbenzothiazoline-6-sulfonic acid) (ABTS) resulting in the oxidized colored product (ABTS<sub>ox</sub>)-absorbing light at  $\lambda = 420$  nm or 3,3',5,5'-tetramethylbenzidine (TMB) which is converted to the oxidized product absorbing light at  $\lambda = 650$  nm. The optical analysis of the ABTS or TMB oxidation products can be used to follow any biocatalytic reaction resulting in the formation of H<sub>2</sub>O<sub>2</sub> when it is coupled to the process biocatalyzed by HRP. Optical analysis of ATP can be performed using bioluminescence produced by luciferase/luciferin biocatalytic system (Fraga 2008). Light emission produced by this system is a simple way to analyze ATP, which can be coupled to any other biocatalytic reaction producing or consuming ATP utilized in enzyme logic gates. In the sections below, the optical absorbance and bioluminescence assay coupled to biocatalytic reactions mimicking

various Boolean logic gates are discussed. Other optical methods, for example, surface plasmon resonance, can be used to follow biocatalytic reactions and analyze the output signals produced by the enzyme logic gates.

### Optical Absorbance Measurements for Transduction of Output Signals Produced by Enzyme Logic Gates

Various enzyme-catalyzed reactions have been used to mimic basic Boolean logic operations with the output signals measured as optical absorbance changes due to formation or consumption of colored biochemicals, including AND (Bakshi et al. 2013; Baron et al. 2006; Halámek et al. 2012; Privman et al. 2013a), OR (Baron et al. 2006; Strack et al. 2008a; Zavalov et al. 2012), NAND (Zhou et al. 2009a), NOR (Baron et al. 2006; Zhou et al. 2009a), XOR (Baron et al. 2006; Halámek et al. 2011a; Privman et al. 2010a; Strack et al. 2008a), INHIBIT (Baron et al. 2006; Strack et al. 2008a), Identity (Baron et al. 2006), and Inverter (Baron et al. 2006) gates. The individual logic gates with the optical absorbance readout have been assembled to form logic networks of different structures and various complexities (Niazov et al. 2006; Privman et al. 2009a). Even more complex multienzyme/multi-logic gate branched network with switchable logic operation and optical readout has been applied for the analysis of injury biomarkers (Halámek et al. 2010b). This system represented the most sophisticated logic network still operating in a single solution. Since many enzymes operated together, performing different logic functions, the optimization of the reacting conditions was extremely complicated. This work was extended to reversible logic gates composed of several Boolean operations organized in the form of complex networks: Feynman (controlled NOT (CNOT)) gate (Moseley et al. 2014), Double Feynman gate (Fratto and Katz 2015), Toffoli gate (Fratto and Katz 2015), Peres gate (Fratto and Katz 2015), and Fredkin (Controlled-Swap) gate (Fratto and Katz 2016). These logic gates included many enzyme-catalyzed reactions organized in flow systems with modular architecture allowing for the optical analysis of multiple products in



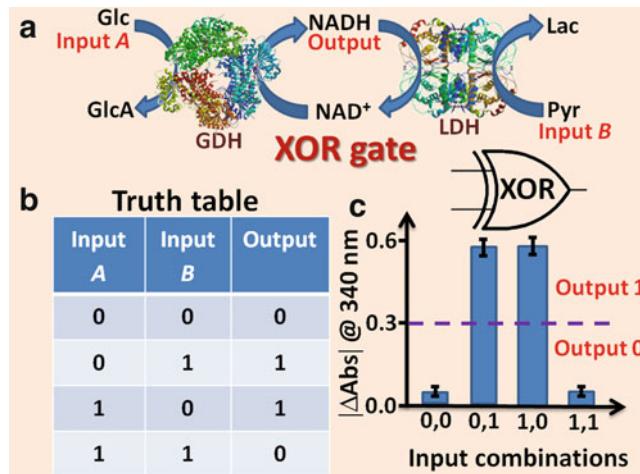
**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 5** Optical absorbance measurements used for the analysis of the output signals produced by the enzyme logic gate. (a) The OR logic gate based on enzyme catalytic reactions – schematics. (b) Truth table of Boolean OR gate. (c) Optical

absorbance corresponding to the biocatalytically produced NADH measured for different input combinations. The dash line shows the threshold value separating logic 1 and logic 0 output signals (Adopted with permission from Katz 2017)

different channels. It should be noted that systems of such high complexity cannot be assembled in a homogeneous solution because of “cross-talking” between different logic gates. Thus, spatial separation of the reacting processes and time difference between them (“clocking”) are mandatory for their realization in flow devices composed of separate reacting cells and channels. Also, the multichannel design of the logic systems allowed the use of the same chemical species (e.g., NADH) for different logic outputs analyzed separately in different channels. In this section we will discuss example systems of limited complexity based on the enzyme-catalyzed reactions mimicking OR, AND, and XOR logic gates with the output signals measured as optical absorbance changes. The interested readers can find more complex systems based on the same optical absorbance readout method in many published papers (Halámek et al. 2010b; Niazov et al. 2006; Privman et al. 2009a).

The OR gate was realized using two parallel biocatalytic reactions activated with two different input signals and producing the same chemical in both reactions considered as the output signal (Fig. 5a). Two enzymes, glucose dehydrogenase (GDH) and alcohol dehydrogenase (ADH), and the NAD<sup>+</sup> cofactor represented the non-variable part of the system (“machinery”), which was the same for all combinations of the applied inputs.

Glucose (Glc) and ethanol (EtOH) were defined as input signals, Input A and Input B, respectively, and they were applied to the gate “machinery” in four different combinations: 0,0; 0,1; 1,0, and 1,1, where logic value 0 corresponded to the absence of the input chemical (meaning its zero concentration), while the input concentration corresponding to logic value 1 was optimized experimentally to produce conveniently high output signals. Input A (Glc) was converted to the oxidized product, gluconic acid (GlcA), reducing in the concomitant reaction NAD<sup>+</sup> to yield NADH. This reaction was catalyzed by GDH. In another reaction catalyzed by ADH, EtOH was oxidized to acetaldehyde (Ald), and NAD<sup>+</sup> was reduced to NADH. Overall, NADH, considered as the final output product, was produced in the presence of either input or both of them (input combinations 0,1; 1,0; and 1,1). The only input combination resulting in no production of NADH was 0,0 (in other words the absence of both substrates for the biocatalytic reactions). Figure 5b shows the truth table corresponding to the OR gate, and Fig. 5c shows the experimental results for the enzyme-based system mimicking OR gate function. Note that the biocatalytically produced NADH was analyzed by following optical absorbance at  $\lambda = 340$  nm characteristic of NADH. The low and high absorbance separated by a threshold was considered as 0 and 1 output signals.



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 6** Optical absorbance measurements used for the analysis of the output signals produced by the enzyme logic gate. (a) The XOR logic gate based on enzyme catalytic reactions – schematics. (b) Truth table of Boolean XOR gate. (c)

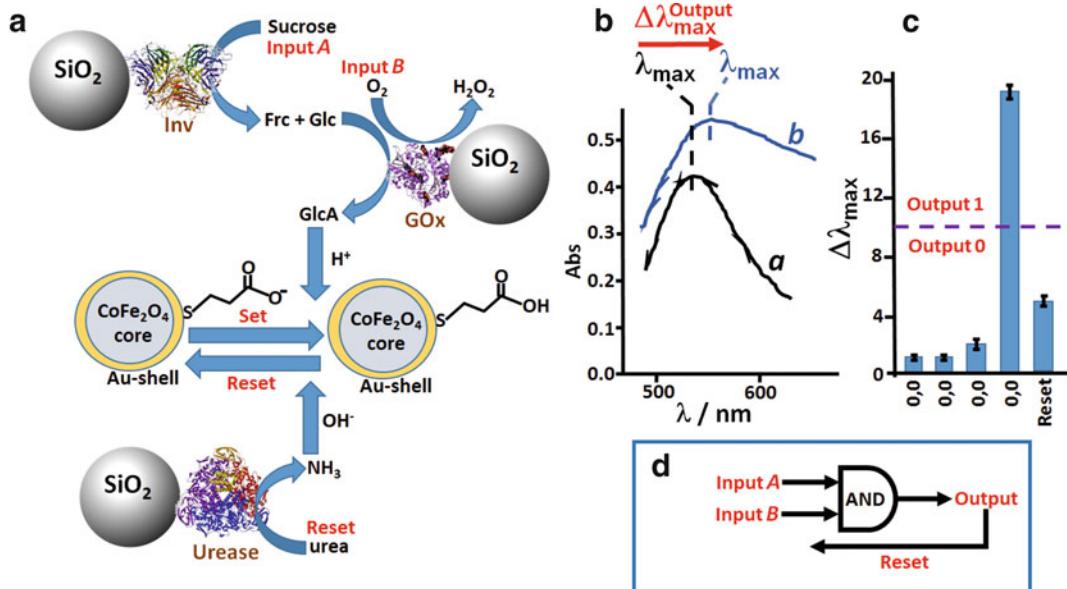
The XOR gate was realized using two biocatalytic reactions driven in the opposite directions by two-input signals represented by Glc and pyruvate (Pyr), Input *A* and Input *B*, respectively. The “machinery” part of the logic gate included GDH and lactate dehydrogenase (LDH) (Fig. 6a). NAD<sup>+</sup> and NADH cofactors (oxidized and reduced forms, respectively) were added to the system in equal concentrations as a part of the “machinery.” In the presence of Glc, the reaction catalyzed by GDH resulted in the reduction of NAD<sup>+</sup>, thus increasing the NADH concentration and the corresponding absorbance at  $\lambda = 340$  nm. On the other hand, in the presence of Pyr, the reaction catalyzed by LDH resulted in the oxidation of NADH, thus decreasing its absorbance. Thus, the optical absorbance was changed, increased or decreased, in the presence of Glc or Pyr (input signal combinations **0,1** and **1,0**), respectively. The biocatalytic reactions were optimized in such a way that both of them running simultaneously (inputs **1,1**) compensated each other, thus resulting in no absorbance changes. Obviously, applying no chemical inputs (inputs **0,0**), the biocatalytic reactions were not activated, and the absorbance was not changed. In order to

Absolute value of the optical absorbance changes corresponding to the biocatalytically produced/consumed NADH measured for different input combinations. The dash line shows the threshold value separating logic **1** and logic **0** output signals (Adopted with permission from Katz 2017)

fit the XOR gate logic function, the output signal was defined as the absolute value of the absorbance change corresponding to either increase or decrease of the NADH concentration. Figure 6b shows the truth table of the XOR gate, and Fig. 6c demonstrates results for its experimental realization. Note that the exemplified XOR gate produces the output signal **0** for the balanced input signals (**0,0** and **1,1**), while the unbalanced inputs (**0,1** and **1,0**) result in the output **1** measured as the absolute value of the absorbance change.

Two logic gates (OR, XOR) discussed above are the most typical examples of the enzyme-based systems used for mimicking Boolean logic operations. The AND gate and multi-gate cascades with the optical analysis of the generated output signals were discussed above in section “[Enzyme-based Logic Gates and Short Logic Circuits](#).” Many other gates with different logic (Baron et al. 2006) and more complex signal-processing functions (Fratto and Katz 2015, 2016; Moseley et al. 2014) have been experimentally realized using optical absorbance measurements as the transduction technique to follow the output signals.

A different approach to the optical analysis of output signals produced by the enzyme logic gates



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 7** Optical absorbance measurements used for the analysis of the output signals produced by the enzyme logic gate. (a) The AND–Reset logic system based on the biocatalyzed reactions and integrated with the magnetic core/Au-shell nanoparticles functionalized with a thiolated monolayer containing carboxylic groups. (b) Absorbance spectra corresponding to the LSPR of the NPs measured at

different pH values: (a) pH 7.0, (b) pH ca. 4. (c) Logic output defined as the  $\lambda_{\max}$  shift generated by the system upon application of various combinations of the input signals and the reset signal. The dash line shows the threshold value separating logic **1** and logic **0** output signals. (d) Logic scheme of the AND–Reset system (A fragment of this figure was adapted from Pita et al. 2008a with permission)

was realized using shift of the absorbance band rather than change of the absorbance intensity (Pita et al. 2008a). The optical analysis was based on the dependence of plasmon energy in Au nanostructures controlled by the electrical charge formed on their surfaces. Magnetic core (CoFe<sub>2</sub>O<sub>4</sub>)–Au-shell nanoparticles (NPs; 18 ± 3 nm) were functionalized with a thiolated monolayer bearing carboxylic groups (Fig. 7a). The charge produced on the surface of the NPs was controlled by the dissociation state of the surface-bound carboxylic groups. At the initial neutral pH, the carboxylic groups were dissociated, thus producing a negative charge on the surface. When pH was decreased below pK<sub>a</sub> of the carboxylic groups (ca. 5.2 ± 0.1) (Zhao et al. 1999), the carboxylic groups were protonated, and the surface charge became neutral. This pH-dependent change of the surface charge resulted in the change of the surface-localized plasmon energy, thus shifting  $\lambda_{\max}$  of the NPs absorbance band (the neutral

charge resulted in the shift of the  $\lambda_{\max}$  to longer wavelength – red shift of the absorbance band). The pH change was produced by the enzyme-catalyzed reactions mimicking OR and AND logic gates (Pita et al. 2008a). The enzymes were covalently bound to SiO<sub>2</sub> particles (ca. 74 μm diameter) used as a platform for the biocatalysts. Figure 7a shows schematically the biocatalytic process mimicking the AND logic gate (the OR gate was mimicked in a similar way but using different enzymes catalyzing different reactions). The biocatalytic process was activated with two inputs, sucrose, and O<sub>2</sub>, Inputs A and B, respectively. Sucrose was cut to fructose (Frc) and glucose (Glc) in the reaction catalyzed by invertase (Inv). Then glucose was oxidized by GOx, yielding gluconic acid (GlcA), thus, decreasing the pH value. The two-step biocatalytic reaction proceeded to the end only in the presence of both input signals (logic combination **1,1**), thus mimicking the

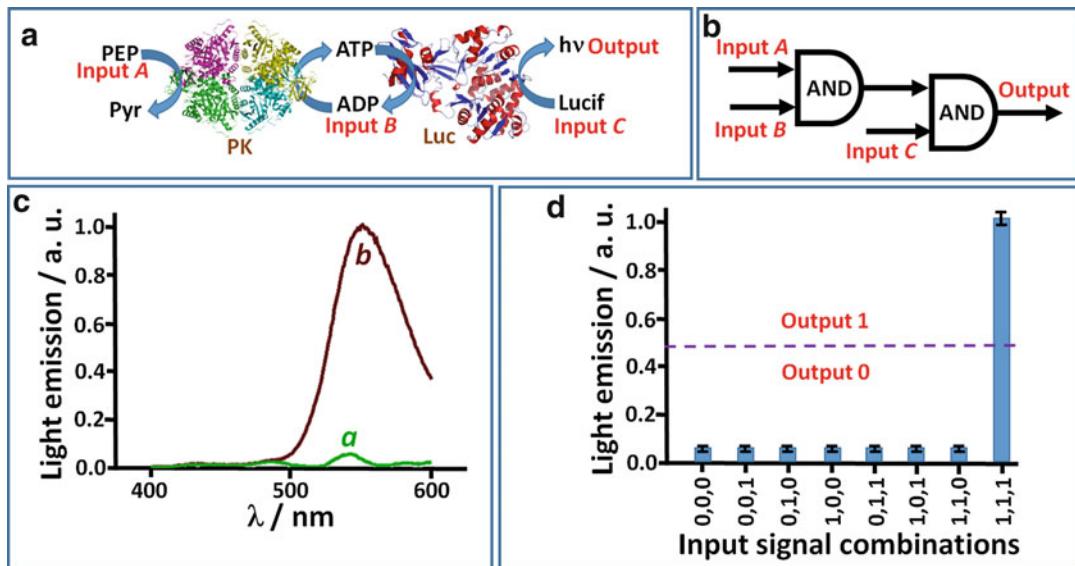
AND logic gate. The *in situ* produced low pH resulted in the protonation of the surface-confined carboxylic groups and resulted in the absorbance wavelength shift originating from the change of the localized surface plasmon resonance (LSPR) (Fig. 7b). The large shift of the  $\lambda_{\text{max}}$  (ca. 20 nm) was observed only for the **1,1** combination of the input signals, while  $\lambda_{\text{max}}$  was preserved almost unchanged for all other input combinations (**0,0; 0,1; 1,0**), as expected for the AND gate (Fig. 7c). The logic gate was reset to the initial pH and to almost initial  $\lambda_{\text{max}}$  by the reaction catalyzed by urease converting urea to NH<sub>3</sub> (Fig. 7a, c). The whole biocatalytic system represented the two-input AND gate with the Reset function (Fig. 7d). Different components of the logic system played different roles in the signal processing and output readout. The SiO<sub>2</sub> particles suspended in the solution operated as a convenient platform for the enzyme immobilization, and the superparamagnetic core (CoFe<sub>2</sub>O<sub>4</sub>) allowed easy manipulation of the signal-reading NPs in the presence of an external magnetic field, while the Au-shell was the reporting unit transducing pH changes to the optical signals. Other logic systems based on the enzyme-catalyzed pH changes are discussed below in sections “pH-Measurements as a Tool for Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems,” “Indirect Electrochemical Analysis of Output Signals Generated by Enzyme-Based Logic Systems Using Electrodes Functionalized with pH-Switchable Polymers,” “Conductivity Measurements as a Tool for Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems,” “Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems Using Semiconductor Devices,” and “Atomic Force Microscopy (AFM) Transduction of Chemical Output Signals Produced by the Enzyme-Based Logic Systems.”

#### Bioluminescence Measurements for Transduction of Output Signals Produced by Enzyme Logic Gates

While optical absorbance measurements are the most frequently used method for analyzing output

signals generated by the enzyme logic systems, the method is limited to the processes yielding or consuming chemicals absorbing light in the visible (including near UV) range of wavelength. The use of short UV or IR spectra are not convenient for multicomponent complex biochemical systems (note that proteins/enzymes absorb light around 280 nm, thus restricting optical measurements of the output signals in this UV area). Therefore, the analysis of biomolecules which are not colored requires different techniques. The most typical example is bioluminescent analysis of adenosine triphosphate (ATP) through a biocatalytic reaction in the presence of luciferase/luciferin system (Fraga 2008).

Figure 8a shows a two-step biocatalytic reaction in the presence of two enzymes, pyruvate kinase (PK) and luciferase (Luc), activated by three inputs, phosphoenolpyruvate (PEP), adenosine diphosphate (ADP), and luciferin (Lucif), Inputs *A*, *B*, and *C*, respectively. The first reaction catalyzed by PK results in the consumption of PEP and the concomitant conversion of ADP to ATP. Then, the second reaction catalyzed by Luc results in bioluminescence when ATP is produced through the first reaction, and luciferin is present in the system. The two-step reaction activated with three input signals resembles a logic network composed of two concatenated AND gates (Fig. 8b). Indeed, both biocatalytic reactions proceed till the end only in the presence of all reacting species (input combination **1,1,1**), while the absence of any of the input chemicals should inhibit the bioluminescence. Figure 8c, curve *b*, shows the light emission spectrum (bioluminescence) defined as the output signal **1** and observed for the input combination **1,1,1**. All other input signal combinations (**0,0; 0,1; 1,0**) resulted in the output signal **0** observed as low intensity background luminescence, Fig. 8c, curve *a*. The bar chart (Fig. 8d) shows all experimentally measured output signals for eight combinations of the inputs. The present system exemplifies the application of bioluminescence as the method for observing output signals generated in the presence of ATP, which is a very common component of various enzyme-based logic systems of various complexity (Guz et al. 2014). Another system mimicking a branched logic network was based



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 8** Bioluminescence measurements used for the analysis of the output signals produced by the enzyme logic gate. **(a)** Two concatenated AND logic gates based on enzyme catalytic reactions – schematics. **(b)** Logic scheme of the concatenated AND gates. **(c)** Bioluminescence spectra

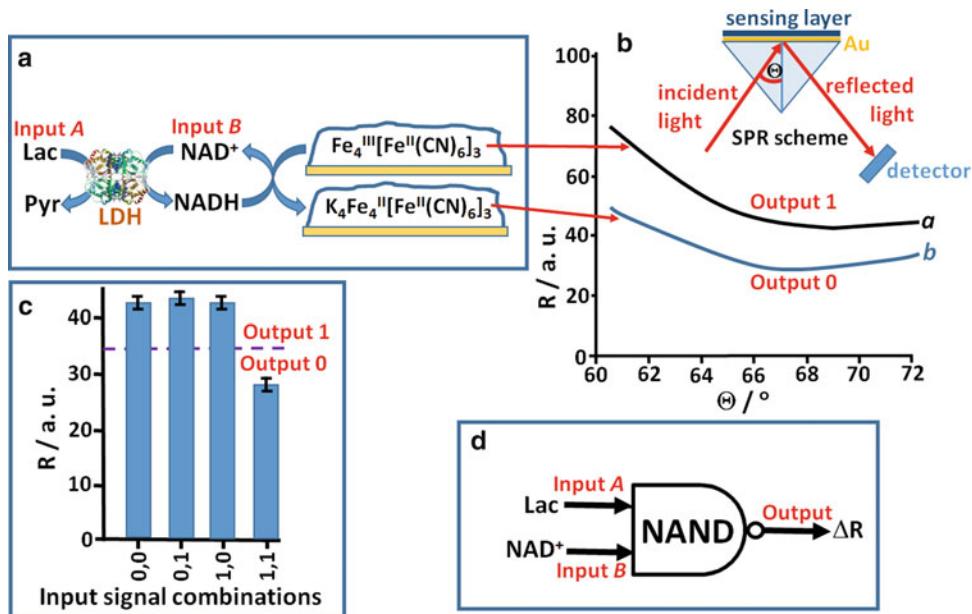
measured for the logic system when it produces output **0** (*a*) and **1** (*b*). **(d)** Light emission by the bioluminescent system measured for different combinations of the input signals. The *dash line* shows the threshold value separating logic **1** and logic **0** output signals (A fragment of this figure was adapted from Guz et al. 2014 with permission)

on bioluminescence produced upon the oxidation of luminol catalyzed by horseradish peroxidase (HRP) in the presence of enzymatically produced  $H_2O_2$  (Tam et al. 2009a). Since  $H_2O_2$  is a common product of many enzyme reactions (represented by different oxidases, e.g., glucose oxidase, lactate oxidase, etc.), the bioluminescence produced by the  $H_2O_2$ -HRP-luminol system is a very convenient method of the output signal transduction, being particularly useful when the absorbance measurements are difficult, for example, in heterogeneous system.

#### Surface Plasmon Resonance Measurements for Transduction of Output Signals Produced by Enzyme Logic Gates

Redox reactions taking place in polymer thin films deposited on a gold surface result in changes in surface reflectance and can be detected by surface plasmon resonance (SPR) measurements (Chegel et al. 2002; Wang and Knoll 2006). These surface redox processes can be coupled to biocatalytic

reactions (Raitman et al. 2001, 2002) and used to analyze output signals produced by the enzyme-based logic gates. Figure 9a shows schematically the reaction catalyzed by LDH and activated with two-input chemicals: lactate (Lac) and  $NAD^+$ , Input *A* and Input *B*, respectively. The reaction resulted in the reduction of  $NAD^+$  to NADH with the concomitant oxidation of Lac to pyruvate (Pyr). The biocatalytic system operated in a solution in contact with a chemically modified Au surface (SPR plate). Prussian blue (PB), an inorganic, insoluble, three-dimensional polymer, was deposited electrochemically on the Au surface and used to transduce the biocatalytic reaction output (NADH) to the surface reflectance changes measured by SPR (Raitman et al. 2001). The processes was started when PB was in the oxidized state,  $Fe_4^{III}[Fe^{II}(CN)_6]_3$ , produced by applying 0.3 V (vs. Ag/AgCl reference electrode) on the modified electrode. This state of PB demonstrated a relatively high reflectance measured by SPR method (Fig. 9b, curve *a*). In the presence of



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 9** SPR measurements used for the analysis of the output signals produced by the enzyme logic gate. (a) The NAND logic gate based on enzyme catalytic reactions coupled to redox transformations of Prussian blue thin film on an SPR sensing surface. (b) Reflectance spectra measured by

SPR and corresponding to the output **1** (a) and **0** (b). (c) Reflectance measured by SPR at  $\Theta = 68 {}^\circ$  for different combinations of the input signals. The dash line shows the threshold value separating logic **1** and logic **0** output signals. (d) NAND gate scheme (Adopted with permission from Katz 2017)

the biocatalytically produced NADH, PB was chemically reduced to  $\text{K}_4\text{Fe}_4^{\text{II}}[\text{Fe}^{\text{II}}(\text{CN})_6]_3$  state with a smaller reflectance (Fig. 9b, curve b), thus the reflectance value responded to the absence or presence of NADH in the solution. Before any reaction the modified surface demonstrated the high reflectance corresponding to the initial oxidized state of PB, but the reflectance was decreased after the biocatalytic production of NADH due to the formation of the reduced state of PB on the surface. Input **A** and Input **B** signals (Lac and  $\text{NAD}^+$ ) were applied in four different combinations, but the reaction resulted in the NADH formation only in the presence of both inputs (**1,1** combination). Therefore, the original high reflectance was preserved when the inputs were applied in **0,0**; **0,1** and **1,0** combinations, and it was decreased in the case of **1,1** combination (Fig. 9c). The high reflectance was defined as the output signal **1**, while the smaller reflectance was considered as the output **0**. The function

demonstrated by the system resembles NAND logic gate with the SPR transduction of the output signal (Fig. 9d). The SPR transduction of the output signal can be beneficial if the direct optical measurements of NADH are difficult, for example, in the presence of other colored components in the system absorbing light similar to the NADH spectrum.

### Electrochemical Analysis of the Output Signals Generated by Enzyme Logic Systems

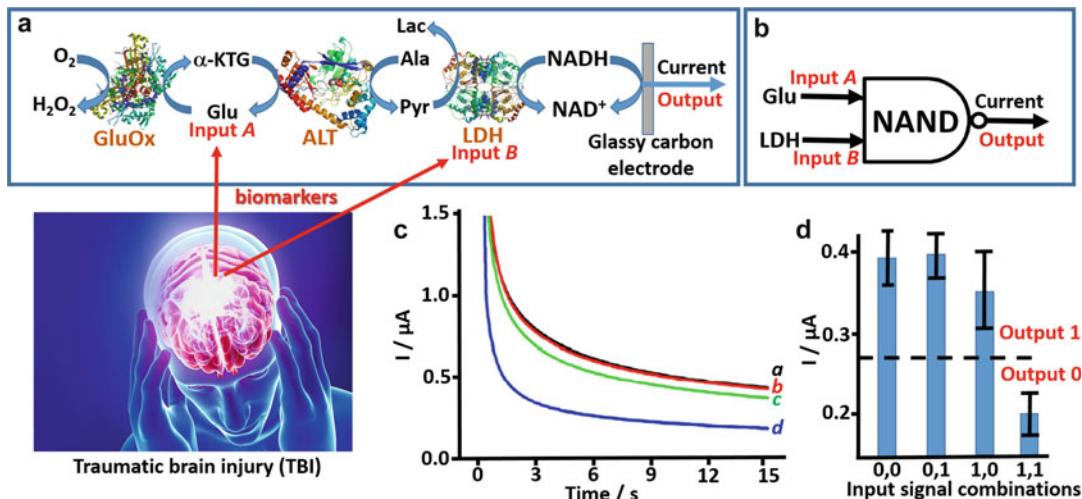
Electrochemical methods can be conveniently applied to the analysis of the output signals produced by the enzyme logic systems when redox-active species (e.g.,  $\text{H}_2\text{O}_2$  or NADH) are produced or consumed as the result of the logic operations (Chuang et al. 2011; Windmiller et al. 2011; Zhou et al. 2011). The electroanalytical methods used in such applications could be similar to those used in electrochemical biosensors. Various electroanalytical methods, including chronoamperometry

(Chuang et al. 2011; Zhou et al. 2011) and potentiometry (Guz et al. 2016; Mailloux et al. 2014a, 2015), can be employed for detecting the output signals. The potentiometry methods can be extended to the use of ion-selective and pH-sensing electrodes (Privman et al. 2009b, 2011; Zhou et al. 2009b), thus extending the output signal measurements to the analysis of various species which are not necessary redox active. Coupling enzyme-catalyzed reactions with switchable stimuli-responsive materials deposited on electrode surfaces allows for indirect electrochemical analysis of the output signals which are not redox active and not directly detectable by electrochemical means (Katz and Minko 2015; Katz et al. 2013; Privman et al. 2009b, 2011; Tam et al. 2008a; Yu et al. 2016; Zhou et al. 2009b). For example, pH variation in the course of enzyme-catalyzed reactions mimicking logic operations have been used to activate electrodes functionalized with pH-switchable polymer-based thin films. The pH changes produced by the biocatalytic system resulted in switching of a polymer brush thin film between a non-permeable shrunk state and a permeable swollen state, thus allowing the electrochemical analysis of the interfacial state using cyclic voltammetry or impedance spectroscopy (Privman et al. 2009b, 2011). Based on the same concept, but using a pH-switchable nano-porous membrane deposited on an electrode surface, enzyme logic operations were transduced to electrical outputs by impedance spectroscopy (Tokarev et al. 2009). Notably, in these systems, the electrochemical analysis was used to follow the interfacial properties of the modified electrode rather than direct analysis of the chemical species produced by the enzyme reactions. Signal-switchable materials can be used not only as thin films on sensing electrodes but also as switchable components of a bulk liquid phase, for example, in the form of a microemulsion switchable between oil-in-water and water-in-oil states (Motornov et al. 2009a). This system can be switched between the states with the low and high electrical resistance upon getting signals produced by an enzyme logic gate. The change in the system state can be followed using simple

DC-electrical conductivity measurements. In addition to electrochemical measurements on conducting electrodes, semiconducting devices (usually in the form of field-effect transistors (FET)) have been successfully used for electronic transduction of chemical output signals produced by the enzyme logic systems (Katz et al. 2017; Krämer et al. 2009; Molinnus et al. 2015; Poghossian et al. 2009, 2011, 2015a).

### Chronoamperometric Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems

Figure 10 exemplifies the use of chronoamperometric measurements for the analysis of output signals produced by enzyme logic system (Zhou et al. 2011). The biocatalytic cascade was initiated by glutamate (Glu) and lactate dehydrogenase (LDH), Input A and Input B, respectively. Note that in this hybrid system, one of the inputs was represented by a substrate for an enzymatic reaction (Glu), while the second input was an enzyme (LDH) (Fig. 10a). Two other enzymes, glutamate oxidase (GluOx) and alanine transaminase (ALT), as well as required substrates, alanine (Ala), NADH, and O<sub>2</sub>, were non-variable parts of the logic gate (“machinery”). Both inputs used in the logic system were represented with biomarkers of traumatic brain injury (TBI), thus aiming at the logic processing of the biomarker concentration changes and TBI diagnosis. In this biomedical/bioanalytical application, the logic values **0** and **1** of the input signals were defined as their physiologically normal and elevated pathophysiological concentrations, respectively. The multistep biocatalytic reaction (Fig. 10a) used in the logic analysis of the input signals (biomarkers) was proceeding at any combination of the inputs because the logic **0** was not physically zero concentration of the reacting species. However, the rate of the process was strongly dependent on the logic combination of the input signals reflecting difference in their concentrations. The highest rate of the biocatalytic reaction was achieved at **1,1** combination of the inputs, while all other combinations (**0,0**; **0,1**; and **1,0**) resulted in low rates of the biocatalytic reactions. The output signal was



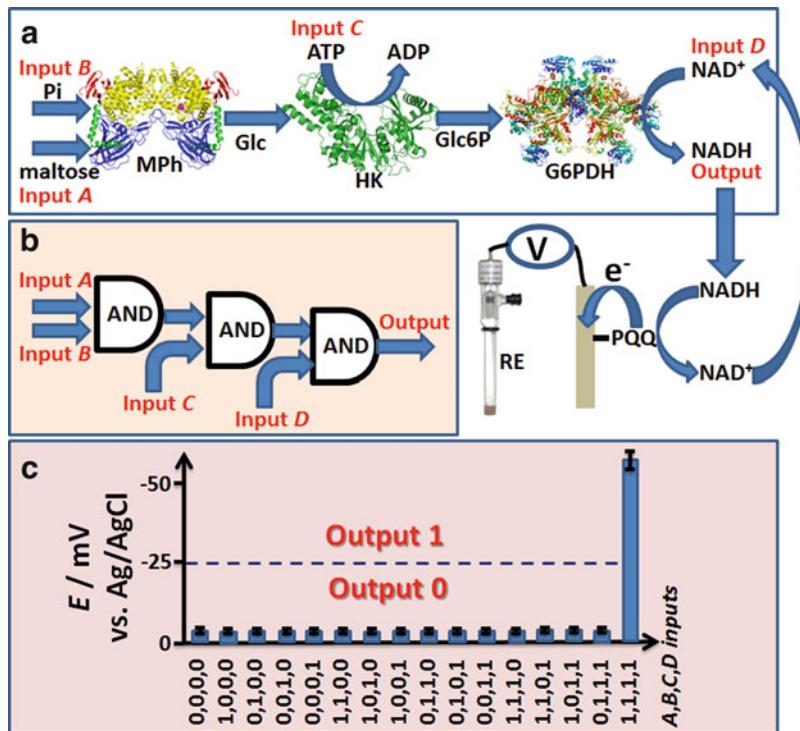
**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 10** Electrochemical transduction of the enzyme generated output signals – chronoamperometric measurements. (a) The NAND logic gate scheme based on enzyme catalytic reactions with the input signals corresponding to traumatic brain injury (TBI) biomarkers. NADH participating in the biocatalytic process is analyzed by chronoamperometry (at  $-0.1$  V vs. Ag/AgCl) on a glassy carbon electrode.

defined as the current on a sensing glassy carbon electrode corresponding to electrochemical oxidation of NADH. Since the analyzed species was consumed (not produced) by the biocatalytic reaction, the output signal was smaller for the higher concentrations of the input species, thus representing the NAND gate (Fig. 10b). The biocatalytic reaction was allowed for 6 min after adding the input signals to the gate “machinery,” and then the chronoamperometric analysis of the residual NADH was performed. Figure 10c shows the experimental chronoamperometric current changes obtained at different logic combinations of the inputs, and Fig. 10d shows the current values measured after 15 s of the chronoamperometric measurements. The low output signal (logic value 0) obtained at 1,1 combination of the inputs was signaling on the TBI conditions, thus allowing the biomedical diagnosis based on the combined consideration of both TBI biomarkers. Despite the fact that the logic analysis of the biomarkers was performed in a model system, the developed approach keeps promise for future

(b) NAND gate scheme. (c) Chronoamperometric curves measured for different combinations of the input signals: 0,0 (a), 0,1 (b), 1,0 (c), and 1,1 (d). (d) Current measured by chronoamperometry after 15 s from the beginning of the potential step for different combinations of the input signals. The dash line shows the threshold value separating logic 1 and logic 0 output signals (A fragment of this figure was adapted from Zhou et al. 2011 with permission)

rapid diagnostics of TBI in the binary “YES/NO” format, particularly useful for point-of-care applications.

**Potentiometric Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems**  
Electrochemical analysis of NADH participating in biocatalytic reactions can be performed by different electrochemical methods, including potentiometric measurements (Fig. 11). An example system (Mailloux et al. 2015) represented by a biocatalytic cascade activated with four input signals (Fig. 11a) was mimicking three concatenated AND logic gates (Fig. 11b). The biocatalytic process proceeded to the very end only if all input signals appeared at the logic values 1 (1,1,1,1 combination), thus resulting in the formation of NADH as the final product. The sensing electrode was modified with pyrroloquinoline quinone (PQQ) which is a known catalyst for the NADH oxidation (Katz et al. 1994). The electrocatalytic oxidation of NADH on the PQQ-modified electrode can proceed spontaneously (this was used



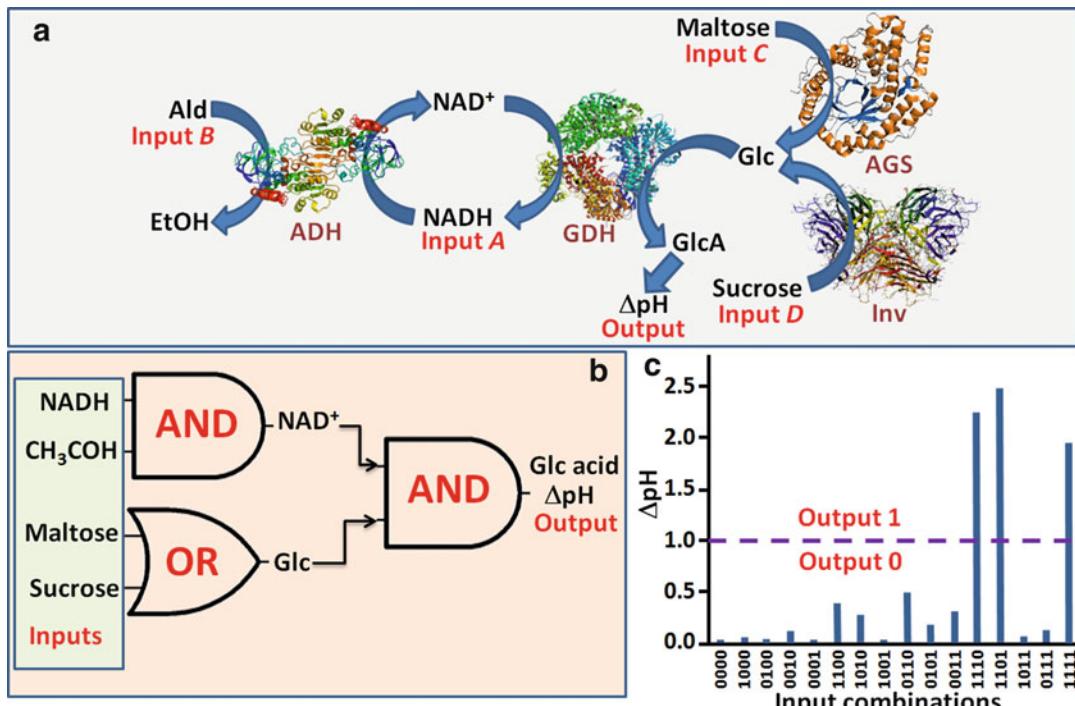
**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 11** Electrochemical transduction of the enzyme generated output signals – potentiometric measurements. (a) The biocatalytic cascade activated with four input signals and ended with the reduction of  $\text{NAD}^+$  to NADH. The produced NADH was analyzed potentiometrically on an electrode modified with PQQ catalyzing its oxidation. The negative potential produced on the PQQ-sensing electrode in the

presence of NADH was measured vs. a reference electrode. (b) The logic scheme composed of three concatenated AND logic gates corresponding to the biocatalytic cascade shown in (a). (c) Potentials produced on the PQQ-sensing electrode after applying four input signals in 16 different combinations. The dash line shows the threshold value separating logic **1** and logic **0** output signals (A fragment of this figure was adapted from ref. Mailloux et al. 2015 with permission)

previously in biofuel cells based on NADH oxidation (Willner et al. 1998) resulting in the formation of the negative potential measured versus a reference electrode (RE). Figure 11c shows the potential produced on the PQQ-modified sensing electrode at different combinations of the variable logic inputs (16 variants), where only one input combination (**1,1,1,1**) resulted in the high negative potential corresponding to the NADH production in the biocatalytic cascade, as expected for the system mimicking three concatenated AND gates. Similar potentiometric measurements have been used for the transduction of NADH formation to the electronic output in other logic systems of high complexity (Guz et al. 2016).

#### pH Measurements as a Tool for Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems

Many enzyme-catalyzed processes produce acids or bases upon redox or hydrolytic reactions. The most typical examples are the formation of gluconic acid (GlcA) upon oxidation of glucose catalyzed by GOx or GDH and production of organic acids from esters in reactions catalyzed by esterase. These reactions, if they proceed in a solution with low buffer capacity or without buffer at all, result in bulk pH changes which can be used to follow the enzymatic reaction and can serve as the final output signal. Enzyme-based logic gates (e.g., OR, AND) (Bychkova et al. 2010; Pita et al. 2008a, 2011; Privman et al. 2011; Tam



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 12** Electrochemical transduction of the enzyme generated output signals – pH measurements. (a) The biocatalytic cascade activated with four input signals and ended with oxidation of glucose to gluconic acid. The pH change was defined as the output signal measured with a standard pH-sensitive

electrode. (b) The equivalent logic network corresponding to the biocatalytic cascade shown in (a). (c) Bulk pH changes obtained with different combinations of the logic inputs. The dash line shows the threshold value separating logic 1 and logic 0 output signals (A fragment of this figure was adapted from Tam et al. 2009b with permission)

et al. 2009b; Tokarev et al. 2009; Wang et al. 2009; Zhou et al. 2009b) and logic networks of different complexity (Pita et al. 2012; Privman et al. 2009a; Tam et al. 2009b) have been realized using  $\Delta\text{pH}$  as the output signal measured with a regular pH-sensitive electrode and a standard pH meter. On the other hand, hydrolytic decomposition of urea is catalyzed by urease produces ammonia, thus increasing pH value. This reaction can be used to reset the initial neutral pH value after it was decreased by the biocatalytic reactions producing acids (Tokarev et al. 2009).

Figure 12a shows a reaction cascade catalyzed by four enzymes operating as a “machinery,” alcohol dehydrogenase (ADH), glucose dehydrogenase (GDH), amyloglucosidase (AGS), and invertase (Inv), activated with four input signals represented by NADH, acetaldehyde (Ald),

maltose, and sucrose, Inputs A, B, C, and D, respectively (Tam et al. 2009b). The final product of the biocatalytic reactions is gluconic acid (GlcA) which results in acidification of the reaction solution when the reactions proceed to the very end. The pH decrease ( $\Delta\text{pH}$ ) in the bulk solution was considered as the output signal measured with a standard pH-sensing electrode. The reaction cascade included two pathways. One was represented by two parallel reactions catalyzed by AGS and Inv and triggered with maltose and sucrose, respectively. This part of the reaction cascade mimics an OR logic gate and results in the production of glucose. The second pathway was composed of two consecutive reactions biocatalyzed by ADH and GDH and activated with Ald and NAD<sup>+</sup>, respectively. The first reaction step activated with Ald and NADH represents

an AND logic gate. The final reaction step converting glucose to GlcA was catalyzed by GDH only when both intermediate products, NAD<sup>+</sup> and glucose, were produced, thus also representing an AND gate connected to the AND-OR gates operating in parallel (Fig. 12b). The final output ( $\Delta\text{pH}$ ) was high (meaning logic value **1**) only when both intermediate products, glucose and NAD<sup>+</sup>, were produced through the OR and AND gates. This was achieved at the input logic combinations **1,1,1,0**; **1,1,0,1**; and **1,1,1,1**; otherwise the output signal was **0** (meaning small pH changes for all other input combinations) (Fig. 12c). While pH changes can be measured directly in the bulk solution, they can be further used to change interfacial properties of signal-responsive materials associated with electrode surfaces or emulsions and then analyzed using different electrochemical methods (not measuring pH changes directly). These systems are overviewed in the next section of this chapter.

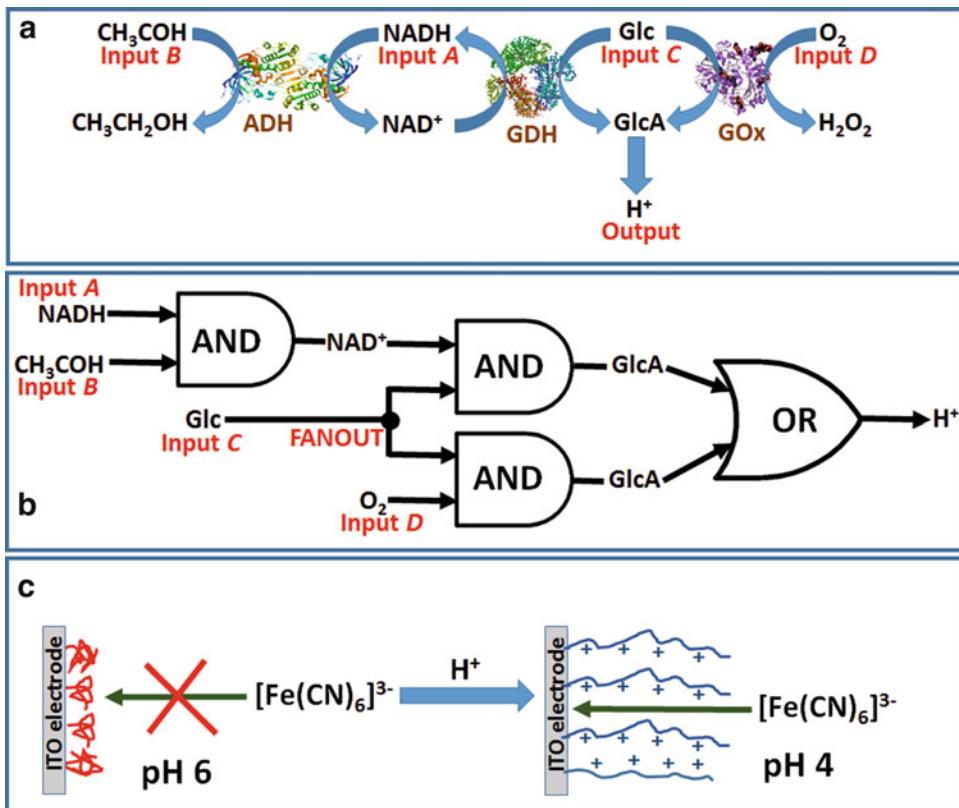
#### Indirect Electrochemical Analysis of Output Signals Generated by Enzyme-Based Logic Systems Using Electrodes Functionalized with pH-Switchable Polymers

Stimuli-responsive materials, exemplified by organic polymers (Luzinov et al. 2004; Minko 2006; Stuart et al. 2010) and more specifically by polyelectrolyte brushes attached to a modified electrode surface (Motornov et al. 2009b; Tam et al. 2008b, 2010a, b, c), can be used to analyze electrochemically changes in the environment, particularly the variation of the pH value. Indeed, protonation/deprotonation of the polyelectrolyte brushes results in their restructuring between swollen hydrophilic state permeable for ionic species of the opposite charge and collapsed hydrophobic state non-permeable for ionic species, thus inhibiting electrochemical processes at the electrode surface. Various polyelectrolyte brushes have been produced on electrodes and demonstrated pH-switchable electrode activity (Motornov et al. 2009b; Tam et al. 2008b, 2010a, b, c). These pH-switchable-modified electrodes have been functionally coupled to the enzyme-based logic systems producing pH changes *in situ* as the output signal of the logic

operations (Pita et al. 2012; Privman et al. 2009b, 2011). While in the majority of the reported systems, the enzyme reactions have been performed in solutions; in some structurally advanced assemblies, enzymes were physically bound to the modified electrode surfaces producing local pH changes at the interfaces, thus resulting in the switch of the electrode activity without bulk pH changes (Bocharova et al. 2010).

Two important factors should be taken into account when pH-switchable polyelectrolyte brushes are used as stimuli-responsive materials in connection with the pH changes produced by enzyme logic systems. (i) The range of the pH changes produced by the biocatalyzed reactions should include the pK<sub>a</sub> value of the polyelectrolyte. In other words, the pH change should be enough to change the protonation state of the polymer brushes. (ii) The direction of the pH change should correspond to the specific application. For example, decreasing pH upon the production of acids (e.g., gluconic acid in the biocatalytic oxidation of glucose) can result in protonation of polyelectrolyte brushes, thus converting them to the positively charged state permeable for anionic redox species and thus activating the modified electrode in the presence of the anionic redox probe. The opposite pH change (possibly in the reset process) can inhibit the electrochemical reactions at the modified electrode surface.

A biocatalytic cascade, outlined schematically in Fig. 13a, results in pH decrease due to the production of GlcA as the final product of a multi-step enzyme-catalyzed reaction (Privman et al. 2009). Three enzymes, alcohol dehydrogenase (ADH), glucose dehydrogenase (GDH), and glucose oxidase (GOx), activated with four input signals, NADH, acetaldehyde, glucose, and oxygen, Inputs *A*, *B*, *C*, and *D*, respectively, operate in concert performing logic operations on the inputs applied in different combinations. Figure 13b explains the biocatalytic reactions in terms of logic operations, in the form of a network composed of four concatenated-branched logic gates. The biocatalytic cascade finally resulting in the pH change was controlled by the pattern of the applied inputs. The pH decrease generated in

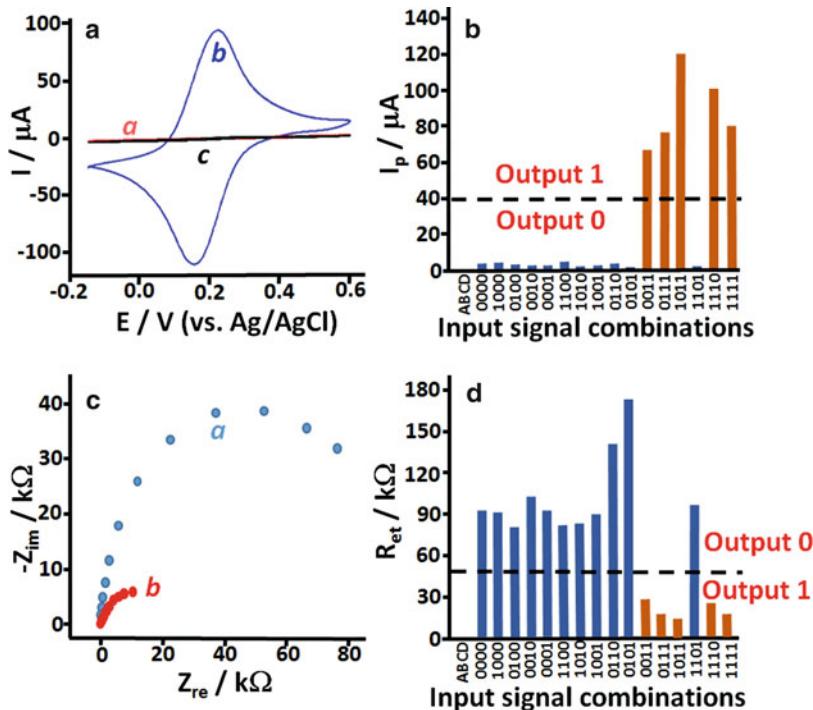


**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 13** (a) The biocatalytic cascade activated with four input signals and ended with oxidation of glucose to gluconic acid yielding acidic pH value. (b) The equivalent logic network corresponding to the biocatalytic cascade shown in (a). (c) pH-switchable electrode surface modified with a

polymeric brush. The hydrophobic interface (*left*) produced at pH 6 is not permeable for the solution redox species, while the protonated hydrophilic interface (*right*) produced at pH 4 is permeable and active for the redox process of the negatively charged redox probe (Adopted with permission from Katz 2017)

situ by the biochemical reactions was coupled with restructuring of a pH-responsive poly(4-vinylpyridine) (P4VP) brush-functionalized electrode. The reorganization of the pH-sensitive P4VP brush polymer from its neutral and hydrophobic state to the protonated and swollen state resulted in the switch of the electrode interface from the “OFF” state when electrochemical reactions are inhibited to the “ON” state when the electrode is electrochemically active (Fig. 13c). The ON-OFF switch of the electrode activity was analyzed following an electrochemical reaction of a soluble diffusional redox probe,  $[\text{Fe}(\text{CN})_6]^{3-/4-}$  (Fig. 14). This allowed to analyze the logic output generated by the enzyme-

catalyzed reactions by electrochemical means. Cyclic voltammograms (Fig. 14a) and Faradaic impedance spectra (Fig. 14c) measured on the P4VP-modified electrode demonstrated its switchable features controlled by the pH value, changed by the enzyme reactions depending on the combinations of the chemical inputs applied to the logic system. Since the number of the variable inputs was 4, the whole set of the input signal combinations included 16 variants. The input combinations resulting in the production of GlcA and thus decreasing the pH value were considered to generate the logic output 1. Otherwise, in case of no pH changes because of no production of GlcA, the system generated the



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 14** Experimental results corresponding to the system shown schematically in Fig. 13. (a) Cyclic voltammograms obtained for the indium tin oxide (ITO) electrode modified with the P4VP polymer brush in (a) the initial OFF state, pH ca. 6.5; (b) the ON state enabled by the input combination **1,1,1,0**, recorded at pH ca. 4.3; and (c) in situ reset to the OFF state, pH ca. 8.8. Potential scan rate was  $100 \text{ mV} \cdot \text{s}^{-1}$ . (b) Anodic peak currents,  $I_p$ , for the 16 possible input combinations. The data were obtained from

cyclic voltammograms recorded under the same conditions as in panel (a). (c) Faradaic impedance spectra (in the form of Nyquist plots) for (a) the initial OFF state, pH ca. 6.5; (b) the ON state enabled with the input combination **1,1,1,1**, recorded at pH ca. 4.0. (b) Electron transfer resistance,  $R_{et}$ , for the 16 possible input combinations. The data were derived from impedance spectra obtained under the same conditions as in panel (c). The *dash lines* in (b) and (d) show the threshold values separating logic **1** and logic **0** output signals (Fragments of this figure were adapted from Privman et al. 2009b with permission)

output signal **0**. The produced pattern of the output signals was determined by the Boolean logic encoded in the logic circuitry (Fig. 14b and d). The use of the switchable P4VP-modified electrode controlled by the pH changes allows the amplification of the output signals. Indeed, the concentrations of the chemical inputs applied for activating the biocatalytic reactions and concentrations of the biocatalytically produced outputs might be much smaller than the concentration of the redox probe (in the present example  $[\text{Fe}(\text{CN})_6]^{3-/-4-}$ ) operating with the switchable electrode. Therefore, the response corresponding to the redox probe measured by electrochemical means in the form of changes in the cyclic

voltammograms or Faradaic impedance spectra might be much higher compared with the electrochemical responses measured directly for the biochemical species participating in the enzyme-catalyzed reactions.

#### Conductivity Measurements as a Tool for Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems

Signal-responsive materials (specifically, poly-electrolyte brushes) can be associated not only with electrode surfaces discussed in the previous section but can also be integrated with various heterogeneous systems bringing them signal-switchable features. For example, silica

nanoparticles ( $\text{SiO}_2\text{-NPs}$ ; 200 nm diameter) were functionalized with polyelectrolyte brushes to allow their switchable hydrophobicity-hydrophilicity features being controlled by external pH changes generated by enzymatic reactions (Motornov et al. 2008, 2009a). Figure 15a shows reactions resulting in the pH decrease due to the formation of acetic acid or gluconic acid catalyzed by esterase and GOx, respectively (Motornov et al. 2009a). This part of the enzymatic process represents the OR logic gate activated by glucose (Glc) and ethyl acetate (Et-O-Ac), Inputs A and B, respectively. The acidic pH produced enzymatically in situ was reset to the initial neutral pH value by producing  $\text{NH}_3$  in the reaction catalyzed with urease and activated with the Reset input of urea. The pH-signal-responsive system included  $\text{SiO}_2\text{-NPs}$  functionalized with a block copolymer, PS-b-P4VP-b-PEO, which is composed of polystyrene (PS), poly(4-vinyl pyridine) (P4VP), and poly(ethylene oxide) (PEO) (Fig. 15b). The switchable amphiphilic  $\text{SiO}_2\text{-NPs}$  have been used as emulsifiers of Pickering emulsions (Binks et al. 2005; Duan et al. 2005; Fujii et al. 2005, 2006; Saleh et al. 2005).

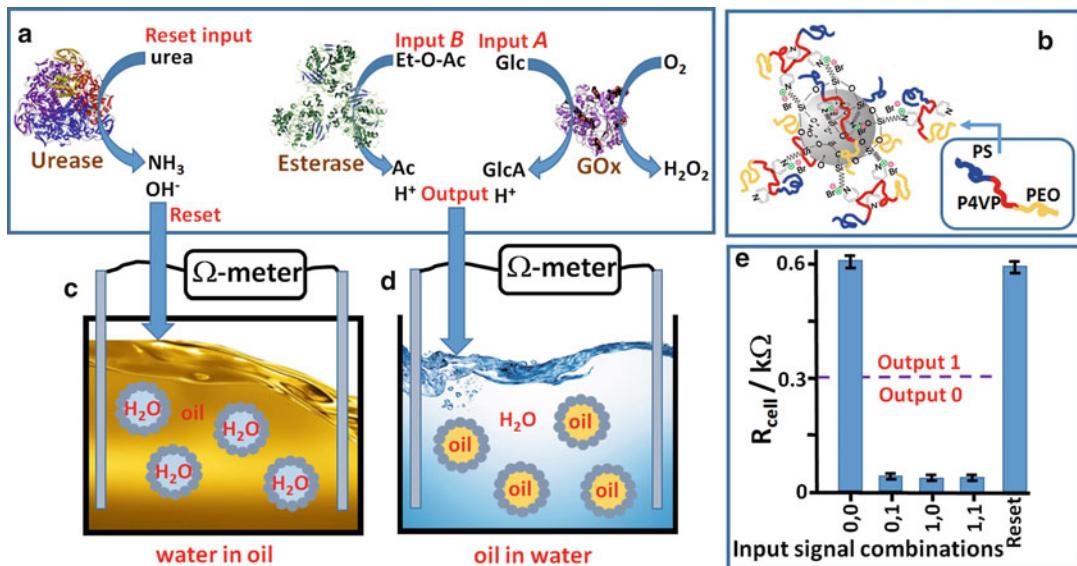
In Pickering emulsions, nanoparticles with amphiphilic properties are located at the water/oil interface stabilizing different kinds of the emulsions, water in oil (W/O) or oil in water (O/W), depending on the wetting properties of the nanoparticles. The nanoparticles with more hydrophilic properties stabilize O/W emulsions, while nanoparticles with more hydrophobic features preferably stabilize W/O emulsions. When the wetting properties of the nanoparticles are changed upon protonation/deprotonation of their acid/base groups, the emulsion can be converted from O/W to W/O or vice versa. As an example, the nanoparticles functionalized with the PS-b-P4VP-b-PEO grafted to the  $\text{SiO}_2\text{-NPs}$  surface stabilize W/O emulsions at pH ca. 6 and O/W emulsions at pH ca. 4 due to non-protonated (neutral) and protonated (positively charged) states of P4VP at the corresponding pH values (Fig. 16a).

As a result, the functionalized  $\text{SiO}_2\text{-NPs}$  perform a “command” function to switch between inverse (W/O) and direct (O/W) emulsions upon receiving signals from enzyme-catalyzed

reactions. The change in the emulsion structure was read out by DC conductivity measurements demonstrating dramatic changes in the conductivity upon the emulsion inversion (Fig. 15c, d). After producing the O/W emulsion with a low Ohmic resistance upon application of 0,1; 1,0; and 1,1 input signals (note that the enzyme system was mimicking the OR gate), the emulsion was converted back to the W/O state with a high resistance by applying the Reset input bringing pH back to pH ca. 6. Figure 15e shows the resistance measured between two electrodes immersed in the emulsion after applying different input combinations to the enzyme logic system mimicking OR gate. Note that the final output plotted in Fig. 15e as the cell resistance ( $R_{\text{cell}}$ ) corresponds to the NOR logic gate since the resistance is decreasing when the pH is increased. Importantly, the switchable features of the  $\text{SiO}_2\text{-NPs}$  interface were transduced to the changes of the bulk conductivity of the liquid system (Motornov et al. 2009a). In addition to the resistance measurements, the reversible transition between the W/O and O/W emulsions was visualized by adding an oil-soluble dye coloring the oil phase (Motornov et al. 2009a) (Fig. 16b, c).

#### Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems Using Semiconductor Devices

The integration of enzyme-based logic gates with field-effect devices based on an electrolyte-insulator-semiconductor (EIS) system (Konorov et al. 2006; Shinwari et al. 2007; Vlasov et al. 2003) is the most attractive and promising approach for the transduction of biomolecular logic signals into electrical output signals. Indeed, the EIS devices are electrochemical analogs of electronic elements used in conventional electronic logic gates and computing systems. Previous research, not always related to the logic gates, has demonstrated the use of EIS devices for the detection of pH changes (Poghossian 1992) and for the analysis of enzymatic reactions (Siqueira et al. 2009; Weng et al. 2016) and charged macromolecules (DNA, proteins, polyelectrolytes) (Abouzar et al. 2012; Poghossian et al. 2013, 2015b). Thus, the use of the EIS electronic devices is straightforward for the



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 15** The use of DC conductivity measurements for transducing output signals generated by the enzyme logic gate. (a) The biocatalytic cascade mimicking OR gate and the biocatalytic reset system – schematics. (b) SiO<sub>2</sub>-NPs functionalized with the block copolymer, PS-b-P4VP-b-PEO, pH-switchable brushes. (c) The W/O emulsion demonstrating the high Ohmic resistance for the DC current. This emulsion was in the beginning of the experiments when pH of the aqueous phase was ca. 6. Also this emulsion was produced upon

applying the Reset input. (d) The O/W emulsion demonstrating the low Ohmic resistance for the DC current. This emulsion was produced when the pH of the aqueous phase was decreased to pH ca. 4 by the production of acids in the enzymatic reactions. (e) The bar chart showing the output signals in the form of the Ohmic DC current resistance measured in the cell after applying the input signals in different combinations. The dash line shows the threshold value separating logic **1** and logic **0** output signals (A fragment of this figure was adapted from Motornov et al. 2009a with permission)

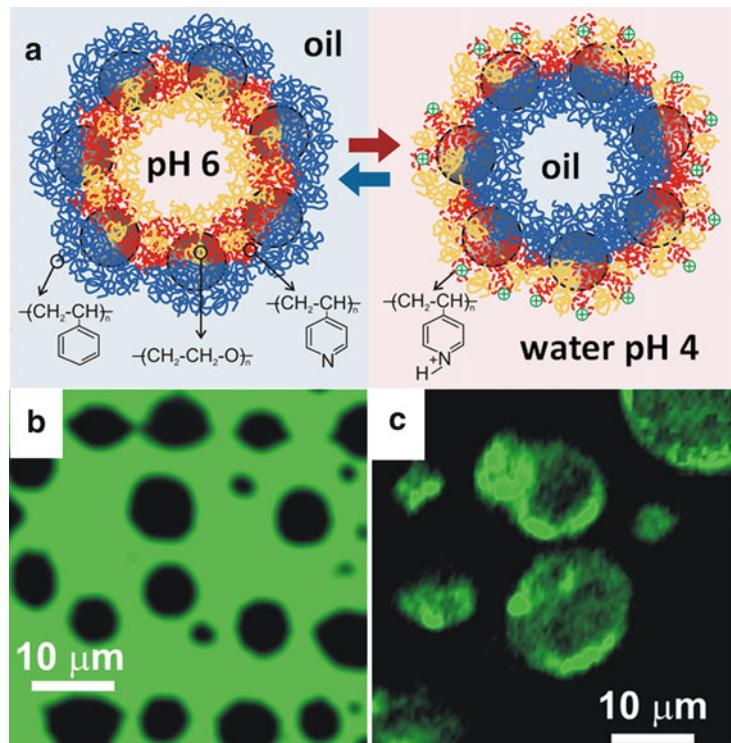
transduction of various output signals generated by the enzyme logic systems.

It should be noted that the experimental work on the enzyme logic systems associated with the EIS devices was mostly limited by relatively simple AND and OR Boolean logic gates producing pH changes readable by the EIS device (Krämer et al. 2009; Poghossian et al. 2009, 2011, 2015a). An example system mimicking OR logic gate realized by two parallel reactions catalyzed by esterase and GOx in a solution is shown schematically in Fig. 17a. Both reactions resulted in acidification of the background solution due to the formation of either butyric acid or gluconic acid or both of them if the reactions are activated by the corresponding input signals of glucose (Glc) and ethyl butyrate, Inputs *A* and *B*, respectively (Krämer et al. 2009). The application of the input signals in three combinations (**0,1**; **1,0**; and **1,1**) resulted in the pH decrease due to the

activation of either or both biocatalytic reactions. The obtained result was considered as the output signal **1**. Obviously, the absence of both inputs (meaning **0,0** logic combination) did not result in any reaction and did not produce any pH change, thus generating the output signal **0**. The reaction solution containing the enzymes and inputs applied in various combinations was analyzed by the pH-sensitive EIS device with the interface functionalized with Au nanoparticles coated with a thiolated monolayer containing carboxylic groups (Fig. 17b). When the EIS-sensing interface was exposed to the initial neutral background solution (pH 7), the carboxylic groups associated with the nanoparticles were dissociated and negatively charged. Their negative charge was preserved in case of the output signal **0** (meaning no pH change) generated by the enzyme logic system in response to **0,0** input combination. However, in the presence of the acids produced by the enzyme

### Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications,

**Fig. 16** (a) Water-in-oil (W/O) and oil-in-water (O/W) emulsions obtained at different pH values due to different protonation states of the pyridine units in the polymer brush bound to the SiO<sub>2</sub>-NPs. (b) and (c) Microscopic images of the W/O and O/W emulsions, respectively, with the oil (toluene) phase colored by adding the oil-soluble 1,1,4,4-tetraphenyl-1,3-butadien dye (note the green color associated with the oil phase) (The figure was adapted from Motornov et al. 2009a with permission)



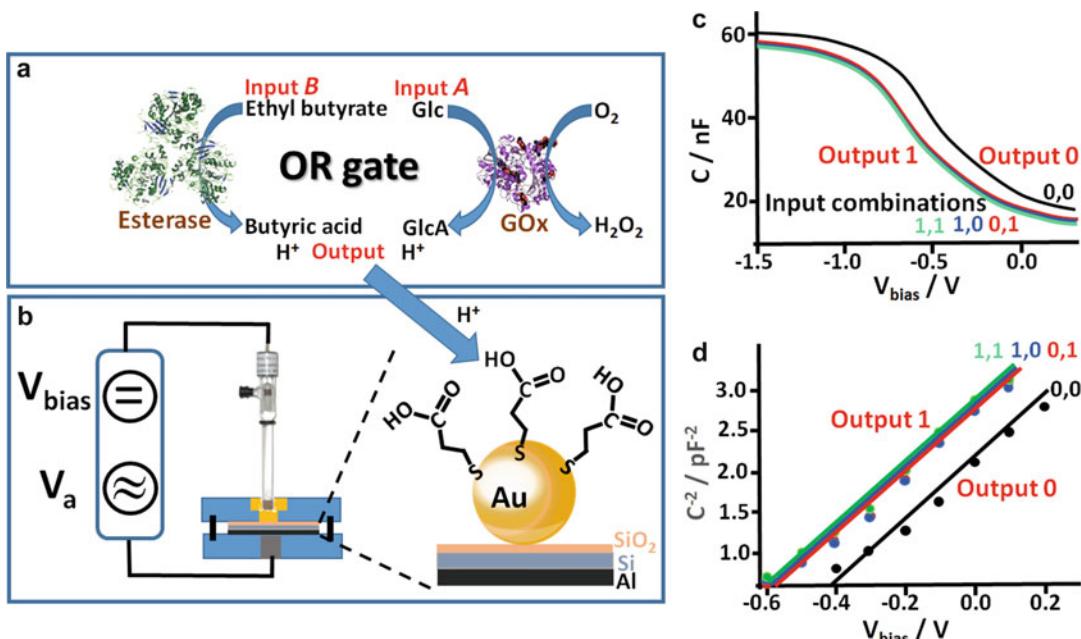
reactions (output signal 1), the carboxylic groups were protonated, and the surface charge became neutral (Fig. 17b).

Impedance spectroscopy, particularly analyzing the interfacial capacitance, was applied to follow the charge variation on the surface of the EIS device (Fig. 17c). Mott-Schottky plots (Albery et al. 1996; Lasia 2014) derived from the impedance spectra allowed the analysis of the flat band voltage ( $V_{FB}$ ) changed as the result of the biocatalytic reaction (Fig. 17d). The interfacial capacitance and  $V_{FB}$  derived from the impedance measurements were considered as the electronic output signals converted by the EIS device from the pH variation produced by the enzyme logic gate. The changes in the EIS interfacial capacitance (Fig. 17c) or shift of the Mott-Schottky function (Fig. 17d) were observed when the input signals (Glc and ethyl butyrate) were applied in three different combinations (0,1; 1,0; 1,1) but not in the 0,0 combination, thus demonstrating the Boolean OR gate features. The present example is really simple in its realization, but the EIS devices can be applied to much more

sophisticated logic systems. The transduction of the biochemical signals, e.g., pH changes, to the electronic format with the use of the semiconductor (EIS) device is conceptually important due to the demonstrated interfacing of the biomolecular logic and truly electronic systems. The full advantage of this approach can be achieved when solid-state electronics is integrated with a “soft matter” biological logic system without soluble species involved in the signal-processing steps.

### Macro/Micro/Nano-Mechanical Transduction of Chemical Output Signals Produced by Enzyme-Based Logic Systems

Enzyme-catalyzed reactions, particularly when the produced chemicals communicate with signal-responsive materials, can result in mechanical movements of objects at macro-, micro-, and nano-scales (Motornov et al. 2008; Munkhjargal et al. 2013; Strack et al. 2010; Tokarev et al. 2009). These processes could be considered as prototypes of futuristic “smart” nanomachines and nano-robotic systems (Morin 2017; Wang 2013), where the mechanical operations are



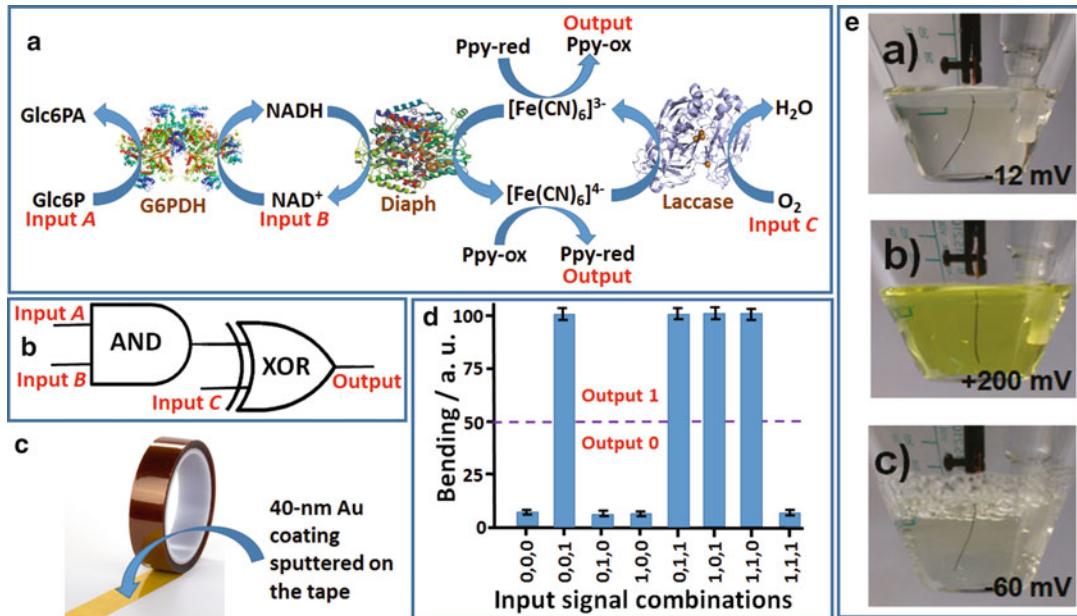
**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 17** Electronic transduction of the enzyme generated output signals – the use of a semiconductor device. (a) The enzyme-based OR logic gate producing pH changes as the output signal and the Reset function realized with a biocatalytic reaction. (b) Electronic scheme of the semiconductor signal-transducing device. The transducer surface is functionalized with Au NPs coated with a thiolated monolayer

containing carboxylic groups. (c) Capacitance versus potential applied to the gate of the EIS device measured after application of the input signals in various combinations. (d) Mott-Schottky plots derived from the impedance spectra obtained for the signal-transducing Si chip after application of the input signals in various combinations (A fragment of this figure was adapted from Krämer et al. 2009 with permission)

controlled by logically processed biomolecular/biological signals. In the example systems overviewed in the sections below, the mechanical transformations are considered as the final output signals produced after processing the input signal combinations through the enzyme biocatalytic cascades mimicking Boolean logic gates. Depending on the dimensions of the mechanically moving species the technical tools used to follow the transformations are ranging from simple photos for macro-objects (Strack et al. 2010) to atomic force microscopy for nano-objects (Motornov et al. 2008; Tokarev et al. 2009).

**Mechanical Bending of a Cantilever Used for Transduction of Chemical Output Signals Produced by the Enzyme-Based Logic Systems**  
Various materials, often represented by conducting electrodes, can be functionalized with

conductive polymers (Inzelt 2008; Skotheim et al. 1998), particularly using polypyrrole (Wang et al. 2001), polythiophene (Schopf and Koßmehl 1997), and polyaniline (Kang et al. 1998), deposited on surfaces to allow control over interfacial properties dependent on the redox state of the polymer film. Counter ion exchange between the polymer matrices and external solutions triggered by the redox reactions in the conducting polymers results in the shrinking–swelling of the polymer films allowing their application as artificial muscles (Baughman 1996). Deposition of the conducting polymers on flexible supports, e.g., polymeric strips (Otero and Sansiñena 1995, 1997, 1998; Pei and Inganläs 1992) or micro-cantilevers (Lahav et al. 2001; Smela et al. 1995), allowed for their mechanical bending upon reduction–oxidation of the polymer films, being usually induced by electrochemical



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 18** Mechanical transduction of the output signals produced by the enzyme logic gate. (a) The biocatalytic cascade activated with three input signals controlling the oxidation state of  $[Fe(CN)_6]^{3-}/[Fe(CN)_6]^{4-}$ -redox species. (b) The equivalent logic network corresponding to the biocatalytic cascade shown in (a). (c) Flexible Au-coated tape used as a conducting support for the Ppy film. (d) The

output signals in the form of tape bending after applying the input signals in different combinations. (e) The photos showing the bending tape when the Ppy film was in its reduced a), oxidized b), and again reduced c) states. The potentials measured (vs. Ag/AgCl electrode) on the conducting tape are shown in the photos (A fragment of this figure was adapted from Strack et al. 2010 with permission)

means. In this case the flexible support was represented by a conducting material chemically modified with the redox polymer. Unfortunately, little attention has been given to chemically induced mechanical actuation triggered by redox transformations of the conducting polymers upon their reactions with reducing/oxidizing species applied in a solution (Küttel et al. 2009; Strack et al. 2010). The chemical, or better biochemical, activation of mechanical processes can be important in the future implantable biomedical devices operating upon physiological commands, thus representing an important research direction, particularly when the mechanical operations are controlled by logically processed multiple biomolecular signals.

Fig. 18a shows a reaction cascade catalyzed by three enzymes, glucose-6-phosphate dehydrogenase (G6PDH), diaphorase (Diaph), and laccase, and activated with three signals, glucose-6-

phosphate (Glc6P), NAD<sup>+</sup>, and O<sub>2</sub>, Inputs A, B, and C, respectively (Strack et al. 2010). The first enzymatic reaction catalyzed by G6PDH resulted in the production of NADH-reduced species when both chemical Inputs A and B appeared, thus representing the AND logic gate. The produced NADH resulted in the formation of  $[Fe(CN)_6]^{4-}$ -reduced species through the Diaph-catalyzed process. On the other hand, in the presence of O<sub>2</sub> (Input C) the reaction catalyzed by laccase resulted in  $[Fe(CN)_6]^{3-}$ -oxidized species. The process started in a solution containing both  $[Fe(CN)_6]^{4-}$ - and  $[Fe(CN)_6]^{3-}$ -redox species. The deviation of their concentrations from the initial values to the dominated reduced or oxidized species was considered as the logic 1 output (meaning that the increase of the reduced or oxidized species is essentially the same logic output). Therefore, the process catalyzed by Diaph and laccase represents a XOR logic gate where the

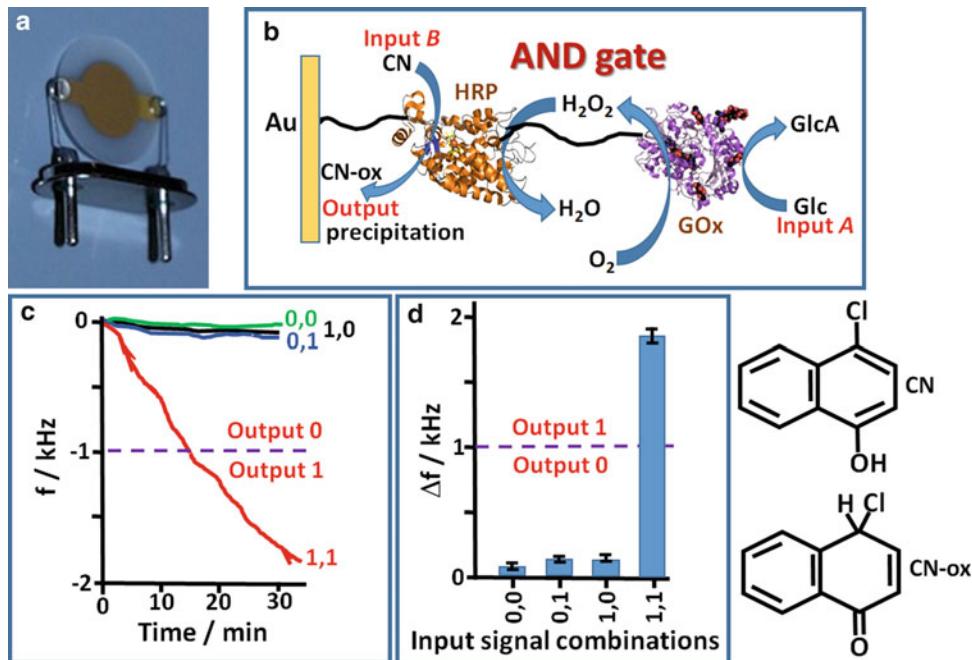
unbalanced operation (input signals **0,1** and **1,0**) produces output **1**, and the balanced operation (input signals **0,0** and **1,1**) results in the output **0**. Indeed, the balanced biocatalytic reactions resulted in no changes in the  $[Fe(CN)_6]^{4-}$  and  $[Fe(CN)_6]^{3-}$  concentrations, while the unbalanced reactions catalyzed by either Diaph or laccase produced more  $[Fe(CN)_6]^{4-}$  or more  $[Fe(CN)_6]^{3-}$ , respectively. Figure 18b shows the equivalent logic network composed of the concatenated AND–XOR logic gates corresponding to the biocatalytic reactions. A flexible polymeric support (polyimide tape) coated with a 40-nm Au conducting film (Fig. 18c) was modified with a polypyrrole (Ppy) film (ca. 8.6  $\mu$ m) deposited electrochemically. When the Ppy-modified strip was contacted with the solution containing the  $[Fe(CN)_6]^{4-}/[Fe(CN)_6]^{3-}$ -redox species and the enzyme logic system, the redox state of Ppy was controlled by the redox state of the species dominated in the solution. In other words, the presence of  $[Fe(CN)_6]^{4-}$  resulted in the reduced state of Ppy, while  $[Fe(CN)_6]^{3-}$  yielded the oxidized state of Ppy. The Ppy-modified strip demonstrated bending upon the variation of the polymer redox state, originating from the redox induced volume changes of the polymer film, thus representing a mechanical actuator controlled by the logically processed biochemical signals. The macroscopic bending of the Ppy strip was photographed using a digital camera (Fig. 18e). The photos shown in images (a), (b), and (c) correspond to the reduced-oxidized-reduced states of PPy, and the potentials measured on the Ppy-modified strip-electrodes are given in the images. The mechanical bending of the PPy-modified strip was defined as the final output signal produced by the enzyme logic system, and it is shown in Fig. 18d as the function of the input signal combinations processed by the enzymatic cascade mimicking the AND–XOR concatenated network.

#### Quartz Crystal Microbalance (QCM) Transduction of Chemical Output Signals Produced by the Enzyme-Based Logic Systems

Enzyme-catalyzed reactions can produce insoluble in water products precipitating on a solid

support modified with the working enzymes. These biocatalytic reactions have been used in various biosensors, mostly based on the formation of biomolecular complexes due to biorecognition processes (immune recognition or DNA complementarity), while the enzymes producing the precipitating products have been used as labels in the biomolecular complexes (Alfonta et al. 2000, 2001; Bardea et al. 2000; Patolsky et al. 1999a, b). The accumulation of the insoluble product on the sensing interface was analyzed using impedance spectroscopy or quartz crystal microbalance (QCM) measurements, thus providing the biosensing signals. Similar biocatalytic systems mimicking logic operations and producing insoluble precipitates on solid supports can be used for micro-gravimetric transduction of the chemical outputs. The QCM measurements are based on change in vibration frequency of the piezoelectric device (Fig. 19a) upon the deposition of the insoluble material on its surface (Johannsmann 2015).

Fig. 19b shows schematically the biocatalytic cascade operated on a QCM Au surface and mimicking AND logic gate. Since the insoluble product should be precipitated on the QCM surface for the micro-gravimetric analysis, the enzymes catalyzing the reactions have to be immobilized on the sensing surface. The Au surface of the QCM sensor was modified with a self-assembled thiolated monolayer bearing active ester groups, which were used for covalent binding of horseradish peroxidase (HRP) through the formation of amide bonds. In the next modification step, the second enzyme, glucose oxidase (GOx), was immobilized on the HRP layer by bridging the enzymes with glutaric dialdehyde. The bi-enzyme layers biocatalytically operated in the presence of two chemical inputs, glucose (Glc) and 4-chloro-1-naphthol (CN). Inputs *A* and *B*, respectively. The glucose oxidation catalyzed by GOx resulted in the concomitant production of  $H_2O_2$ , and the second reaction catalyzed by HRP resulted in oxidation of CN to insoluble oxidized product (CN-ox) precipitated on the QCM surface. The CN-ox formation was only possible in the presence of Glc and CN, in other words when Inputs *A* and *B* were applied at **1,1** combination. When one or both inputs



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 19** Microgravimetric transduction of the output signals produced by the enzymes logic gate. **(a)** A QCM device used for transducing the enzyme logic gate output signals. **(b)** The biocatalytic cascade mimicking AND gate and operating on the surface of the QCM device. The enzyme reactions result in the formation of an insoluble product precipitating

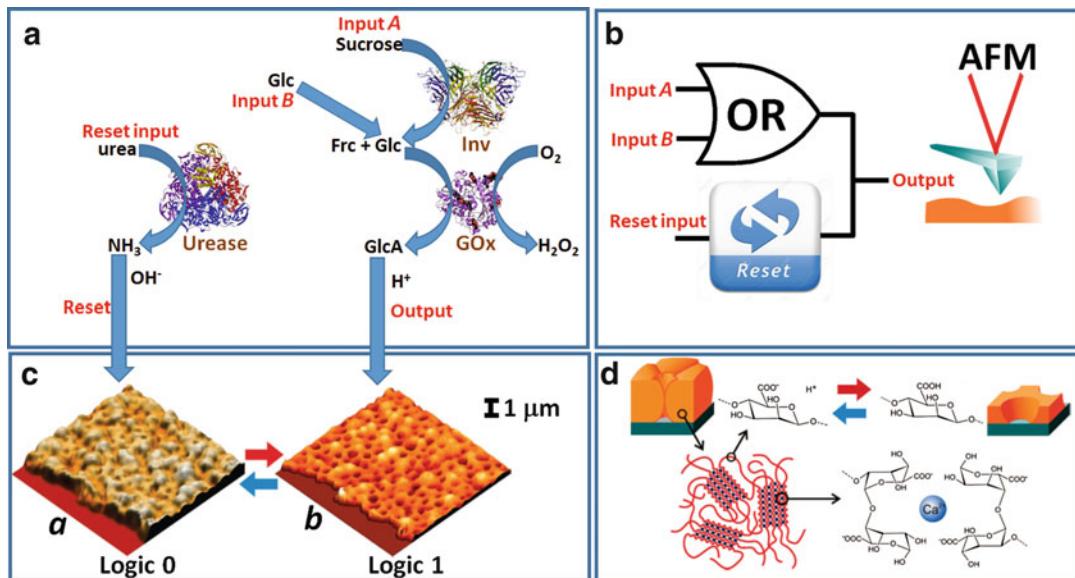
on the QCM surface. **(c)** Frequency variation measured with the QCM device upon application of the input signals in various combinations. **(d)** The bar chart showing the frequency changes measured with the QCM device after 30 min from application of the input signals in different combinations. The structures of 4-chloro-1-naphthol (CN; Input B) and its oxidized product (CN-ox) are shown (Adopted with permission from Katz 2017)

were missing (input combinations **0,0**; **0,1**; and **1,0**), the biocatalytic reactions did not proceed till the very end, and the insoluble product was not formed on the QCM surface. Therefore, the biocatalytic cascade mimicked the AND logic gate. Figure 19c shows time-dependent frequency changes measured on the QCM when the input signals were applied in different combinations. Only **1,1** input combination resulted in the frequency change (corresponding to the output signal **1**) due to the deposition of the insoluble reaction product, while all other input combinations were not affecting the QCM frequency (meaning output signal **0**). Figure 19d summarizes the results in the form of the bar chart showing frequency changes measured after 30 min from the time when the inputs were applied in different combinations. The obtained results demonstrated micro-gravimetric

transduction of the output signals generated by the enzyme logic gate, where the final signal was presented with the mechanical oscillation of the quartz crystal.

#### Atomic Force Microscopy (AFM) Transduction of Chemical Output Signals Produced by the Enzyme-Based Logic Systems

Nanostructured objects functionalized with signal-responsive materials can respond to the signals produced by the enzyme logic gate. The responding nano-objects can be represented by nano-porous membranes with variable nanoporosity (Tokarev et al. 2009) and by nanoparticles with aggregation/disaggregation features (Motornov et al. 2008) dependent on the chemical signals produced by enzymatic reactions. Nano-particles assembly-disassembly can be triggered by chemical signals processed through



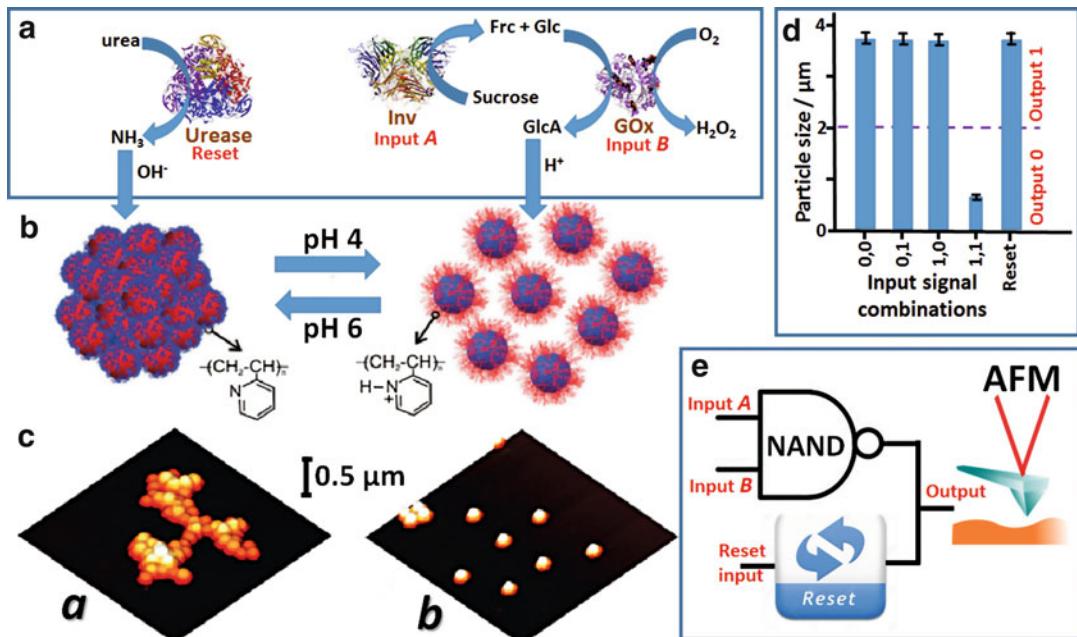
**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 20** AFM transduction of the output signal produced by the enzyme logic gate – switchable porosity in a membrane. (a) The biocatalytic cascade mimicking an OR logic gate and a Reset function. (b) The schematics of the OR-Reset logic device with the output transduced by AFM. (c) The nanoporous membrane visualized with AFM in the closed-pore

state at neutral pH (a) and open-pore state at the acidic pH (b). (d) Schematics showing the swollen alginate polymer resulting in the closed-pore state and shrunk polymer producing the open-pore state of the membrane. The structure of the  $Ca^{2+}$ -cross-linked alginate is shown (A fragment of this figure was adapted from Tokarev et al. 2009 with permission)

biomolecular logic gates representing a biocomputing nanoplatform for therapeutics and diagnostics (Evans et al. 2016). Biomolecular logic gates have been used to arrange dynamically gold nanoparticles on DNA origami (Yang et al. 2016). Assembling of gold nanoparticles has been demonstrated using a pH-responsive DNA nanomachine logically processing biomolecular input signals (Yao et al. 2016).

Fig. 20a shows the biocatalytic cascade mimicking an OR logic gate (Tokarev et al. 2009). The gate “machinery” was composed of two enzymes, invertase (Inv) and glucose oxidase (GOx), and oxygen in the solution. The biocatalytic reactions were initiated by two biomolecular substrates, sucrose, and glucose, Inputs A and B, respectively, applied in four different combinations. Sucrose was cut into fructose and glucose in the reaction catalyzed by Inv. Glucose produced in situ from sucrose or added as an independent input was oxidized in the reaction catalyzed by GOx

yielding gluconic acid, thus decreasing the solution pH value. The acidic solution was produced in the presence of either or both inputs, sucrose and/or glucose, as expected for the OR logic gate. The produced acidic pH was reset to the initial neutral value by a reaction converting urea (Reset input) to  $NH_3$  catalyzed by urease. Overall, the biocatalytic process represented the OR gate with the Reset function (Fig. 20b). The solution with the pH controlled by the external input signals was in the immediate contact with a pH-switchable nano-porous membrane (Fig. 20c) composed of alginate hydrogel (Rehm 2009) comprised of D-mannuronic acid and L-guluronic acid residues cross-linked with  $Ca^{2+}$  cations (Fig. 20d). At the initial neutral pH, the alginate polymer contains deprotonated (negatively charged) carboxylic groups, thus being hydrophilic and swollen. The swollen polymer matrix expends in its volume and closes completely the nanopores in the membrane body. When the solution pH is decreased below



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 21** AFM transduction of the output signal produced by the enzyme logic gate – switchable aggregation/disaggregation of nanoparticles (NPs). **(a)** The biocatalytic cascade mimicking a NAND logic gate and a Reset function. **(b)** Reversible aggregation/disaggregation of the NPs controlled by pH values. The NPs were functionalized with P2VP polymer brushes changing their charge at different pH values due to the pyridine group protonation (pH 4) and deprotonation (pH 6). **(c)** The AFM images of the

aggregated (*a*) and disaggregated (*b*) nanoparticles after their deposition on a surface. **(d)** The bar chart showing the size of the NPs in their aggregated and disaggregated states obtained upon application of the input signals in different combinations and after applying the reset signal. The *dash line* shows the threshold value separating logic **1** and logic **0** output signals. **(e)** Schematics of the NAND–Reset logic device with the output transduced by AFM (A fragment of this figure was adapted from Motornov et al. 2008 with permission)

pK<sub>a</sub> of the alginate polymer (pH < 4), the carboxylic groups are getting protonated resulting in their neutral charge, thus yielding the hydrophobic shrunk polymer. This results in the opening the nanopores ( $380 \pm 116$  nm measured by AFM at the half depth of the pores). The reset of the solution pH to the initial neutral value returns the nanopores to their closed state (Fig. 20c). The operation of the enzyme logic system was based on the pH variation due to the biocatalytic reactions. Obviously, the pH variation can be measured directly with a pH-sensitive electrode or indirectly following the membrane permeability controlled by the pH value. The latter was measured by observing molecular diffusion through the pores and by measuring the membrane impedance when the membrane was

deposited in an electrode surface. However, for the present section of the review, it is important to emphasize that the membrane state was analyzed with atomic force microscopy (AFM) (Fig. 20c). The membrane state with the open nanopores (output signal **1**) was achieved when the input signals were applied at **0,1**; **1,0**; and **1,1** combinations; otherwise the closed-pore state (output signal **0**) was obtained with the input combination **0,0** (Fig. 20c). The membrane was reset to the initial state by the Reset signal after the output signal **1** was produced.

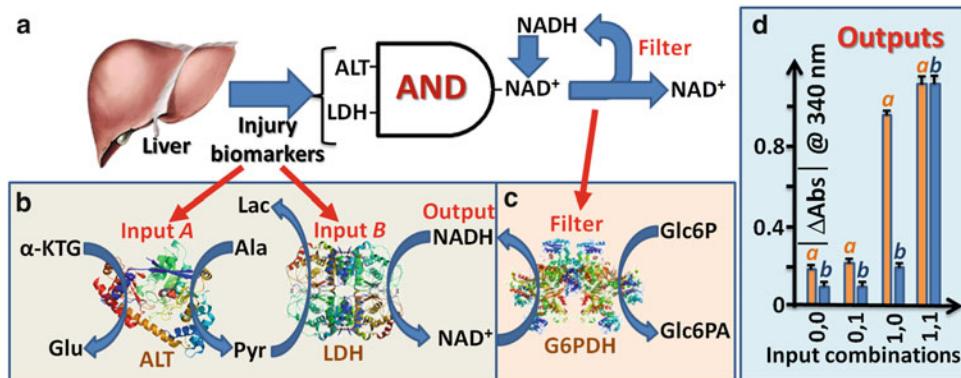
A similar biocatalytic cascade (Fig. 21a) but using enzymes as the input signals (Inv and GOx, Inputs *A* and *B*, respectively) was used to variate the solution pH and then control aggregation/dissociation of SiO<sub>2</sub>-NPs (200 nm) functionalized

with poly(2-vinylpyridine) (P2VP) polyelectrolyte (Motornov et al. 2008). Despite the fact that the biocatalytic cascade was the same as in the previous example, the new definition of the input signals resulted in the different logic function. Indeed, the final output signal (GlcA producing acidic pH value) was obtained only when both biocatalytic reactions proceeded in the presence of both input signals (both enzymes present). At the initial neutral pH value, the P2VP polymer was deprotonated and neutral. This resulted in the aggregation of the neutral NPs, thus yielding their aggregates with a large size. When the pH value was decreased ( $\text{pH} < 4$ ) due to the acidification of the solution with the *in situ* produced GlcA, the pyridine units of P2VP polymer were protonated bringing positive charge to the NPs. The positively charged NPs were dissociated from their aggregates revealing their small size (Fig. 21b). The size of the nano-species in their aggregated and dissociated states was measured *ex situ* observing the structures deposited from the suspensions on Si wafers using AFM (Fig. 21c). The large size aggregates were defined as the output signal **1**, while the small size disaggregated NPs were considered as the output signal **0**. The output **0** was obtained only when both enzyme inputs were present to complete the biocatalytic cascade (input signal combination **1,1**), while all other input combinations (**0,0**; **0,1**; **1,0**) resulted in no formation of acidic pH, thus keeping the NPs in their aggregated state (meaning the output signal **1**). This logic operation corresponds to the NAND gate (Fig. 21d). The reset to the initial neutral pH returning the dissociated NPs to the aggregated state was achieved with the biocatalytic reaction producing  $\text{NH}_3$  (Fig. 21a). Overall, the studied system resembles the NAND–Reset function, which was followed by AFM measurements (Fig. 21e).

The systems exemplified above are also interesting because they demonstrated the nanomechanical actuation logically controlled by the pH signals produced *in situ* through biocatalytic reactions. Also, it should be noted that simple reformulation of the input signal definitions can result in different logic operations (OR/NAND in the given examples).

## Digital Biosensors Based on Enzyme Logic Gates

Enzyme logic gates have been applied for the digital analysis of different biomarkers in biomedical (Katz et al. 2012; Wang and Katz 2010, 2011) and forensic applications (Bakshi et al. 2014; Katz and Halámek 2014; Kramer et al. 2013). Particularly, AND logic gates can be used to analyze biomarkers with limited specificity. While each analyzed biomarker separately cannot be specific enough for any biomedical conclusion, a combination of several biomarkers appearing with a specific concentration pattern can be already enough for the decision with a high level of confidence. The digital processing of biomarkers resulting in the YES/NO (**1,0**) answer is important for making fast conclusion about biomedical conditions, possibly followed by automatic action (e.g., drug release) in the frame of the *Sense-and-Act* concept, being particularly useful in point-of-care and end user applications (Katz et al. 2012; Wang and Katz 2010, 2011). Such bioanalytical systems have been already developed for the analysis of different injuries: soft tissue injury, traumatic brain injury, liver injury, abdominal trauma, hemorrhage shock, oxidative stress (Halámek et al. 2010a; Halámková et al. 2012; Manesh et al. 2009; Zhou et al. 2011), and radiation-caused tissue damage (Bocharova et al. 2011). It is important to note that the definition of the input signals in bioanalytical logic systems is more sophisticated than in the logic gates which do not pretend to be used in practical applications. Indeed, when logic gates are only used for demonstrating the concept of the enzyme-based logic operations, the logic inputs **0** and **1** are defined as the absence of the input (zero concentration of the reactive species) and arbitrary concentrations of the input chemicals, conveniently separated from the logic **0** value. In biomedical applications, the logic **0** value is defined as a normal physiological concentration of the analyzed biomarkers, while logic **1** value corresponds to their pathophysiological concentrations (might be higher or lower vs. normal concentration). In this definition, the gap separating **0** and **1** input concentrations might be narrow, thus resulting in difficulty for separating **0** and **1** output signals. In this situation, the “filter”



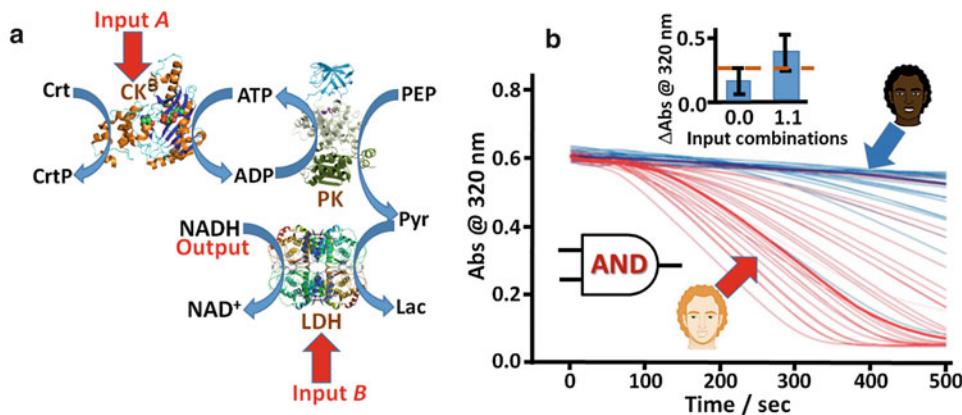
**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 22** (a) The AND gate applied for the logic analysis of liver injury biomarkers (ALT and LDH). (b) The biocatalytic cascade operated as the AND gate with the “filter” system (c) for the logic analysis of the liver injury biomarkers. (d) Optical

output signals obtained for different combinations of the input signals: (a) when the system operated without the “filter” and (b) when the “filter” was added to the biocatalytic cascade (A part of this figure was adapted from Halámek et al. 2011b with permission)

systems discussed above are particularly important (Halámek et al. 2011b, 2012; Mailloux et al. 2014b).

Fig. 22a shows an enzyme-based AND logic gate for the analysis of liver injury biomarkers (Halámek et al. 2011b). In this system the analyzed biomarkers were represented by enzymes: alanine transaminase (ALT) and lactate dehydrogenase (LDH) which have different concentrations under normal physiological conditions and in case of liver injury. Logic **0** and **1** levels of ALT (0.02 and 2 U mL<sup>-1</sup>) and LDH (0.15 and 1 U mL<sup>-1</sup>) input signals were selected in order to mimic circulating levels of these biomarkers under normal and pathophysiological conditions, respectively. The biocatalytic cascade operated in a solution containing  $\alpha$ -ketoglutaric acid ( $\alpha$ -KTG), alanine (Ala), and NADH in experimentally optimized concentrations (Fig. 22b). The first reaction step biocatalyzed by ALT resulted in the production of pyruvate (Pyr); while in the second step biocatalyzed by LDH, the produced Pyr was used to oxidize NADH, thus resulting in the optical absorbance decrease at 340 nm corresponding to the NADH consumption (Fig. 22d). Since **0** logic value of the input signals (represented in this system by enzymes) is not the enzyme zero concentration, the reaction cascade proceeds at any combination of the input signals,

not only at **1,1** combination as it was demonstrated above for other realizations of AND gates. Different combinations of the inputs (**0,0**; **0,1**; **1,0**; and **1,1**) result in different reaction rates and different output signals measured at a specific reaction time. However, the obtained signal pattern does not correspond to the expected AND logic, and the signal combinations **1,0** and **1,1** are yielding similar output signals (Fig. 22d) (see bars labeled with “a”). The system operation was improved by adding one more biocatalytic reaction performing the “filter” function (Fig. 22c). Indeed, the in situ produced NAD<sup>+</sup> was returned back to NADH in the presence of glucose-6-phosphate (Glc6P) and glucose 6-phosphate dehydrogenase (G6PDH) operating as a “filter” system. The optimized filter operation decreased the optical changes, particularly at the **1,0** input combination, and enabled the AND logic gate (Fig. 22d) (see bars labeled with “b”). Overall, the AND logic gate with the added “filter” step is allowed for high confidence analysis of the liver injury, not only in the model system but also in the animal study (Halámková et al. 2012). Multiplexing AND logic gates designed for the analysis of different biomarkers are allowed for rapid finding of various combinations of different injuries, thus giving a chance for fast medical intervention (Halámek et al. 2010a).



**Enzyme-Based Logic Systems: Composition, Operation, Interfacing, and Applications, Fig. 23** (a) The biocatalytic cascade mimicking AND logic gate activated with biomarkers (CK and LDH) characteristic of groups with different ethnicity. (b) Absorbance changes corresponding to the biomarker concentrations characteristic of Caucasian (bottom red lines) and African American

Another application of the enzyme logic gates was recently found in forensic analysis of biomarkers appearing in blood samples left at a crime scene by people of different ethnicity and gender (Bakshi et al. 2014; Katz and Halámková 2014; Kramer et al. 2013). Figure 23a shows a biocatalytic cascade mimicking an AND gate and activated by two enzymes, creatine kinase (CK) and lactate dehydrogenase (LDH), both having different concentrations in the blood of Caucasian and African American groups (Kramer et al. 2013). The system is operated in a solution containing creatine (Crt), ATP, NADH, phospho (enol)pyruvic acid (PEP), and pyruvate kinase (PK) in experimentally optimized concentrations. The reaction started from the process biocatalyzed by CK converting ATP to ADP and then the reaction biocatalyzed by PK resulted in the production of pyruvate, which was finally used to activate oxidation of NADH in the reaction biocatalyzed by LDH. Overall, the process resulted in the optical absorbance decrease at 340 nm corresponding to the NADH consumption. Despite the fact that the reactions proceeded for all combinations of the CK and LDH biomarkers (remember that logic **0** and **1** values corresponded to their concentrations in blood samples typical for people of different ethnicity;

(top blue lines) groups. The different lines correspond to the natural distribution of the biomarker concentrations. The inset shows the output signals for **0,0** and **1,1** input signal combinations corresponding to the African American and Caucasian groups, respectively (A part of this figure was adapted from Kramer et al. 2013 with permission)

logic **0** value is not the absence of the enzymes), the difference between response to **0,0** and **1,1** combinations was enough to distinguish the sample origin with a high confidence. It should be noted that in a model system, all four input combinations (**0,0**; **0,1**; **1,0**; and **1,1**) were possible; however, in the real blood samples, only **0,0** and **1,1** combinations were realized, corresponding to African American and Caucasian ethnic groups (Fig. 23b). The different absorbance changes originated from the natural distribution of the biomarker concentrations. Similar approach was applied for analyzing blood samples to distinguish the difference between male and female samples (Bakshi et al. 2014).

## Conclusions and Perspectives

The majority of research efforts in the area of biomolecular computing concentrate on composition and performance of logic gates, while giving little attention to the methods of the output signal transduction. This chapter aimed at highlighting the underrepresented aspects of biomolecular computing, specifically collecting examples of the enzyme logic systems connected to different physical methods of the output signal transduction.

While the most frequently used method is based on optical absorbance measurements after performing biocatalytic reactions in solutions, other methods, especially based on electrochemical techniques, are highly important. Electrochemical interfaces, particularly functionalized with signal-responsive materials, operating with the enzyme-based logic systems demonstrated significant advantages over the optical readout of the chemical output signals. It should be noted that the optical tools applied to the analysis of the output signals allow only the signal readout, while electrochemical interfaces can operate as actuators triggered by the biomolecular computing systems. Indeed, electrochemical systems can release molecular/biomolecular species in response to the signals processed by the enzyme logic gates, thus activating downstream reactions and processes (Guz et al. 2016; Mailloux et al. 2015), for example, releasing DNA species activating next logic steps in the DNA computing process. The integration of biomolecular logic systems with signal-responsive materials and electronic systems results in novel “smart” interfaces logically responding to various combinations of biochemical signals (Katz and Minko 2015). Interfacing of biocomputing systems, particularly based on enzyme-catalyzed reactions, with switchable electrodes and signal-responsive materials resulted in novel signal-controlled drug-releasing systems, “smart”-switchable membranes, electrodes, and biofuel cell controlled by complex patterns of biomolecular signals. This research direction does not claim to perform complex computations but is rather based on logic signal processing with limited complexity achievable at the present level of technology. Also, electrochemical signal transduces can switch ON/OFF various bioelectrochemical devices (e.g., biofuel cells) in response to logically processed biomolecular signals (Katz and Pita (2009).

Electrochemical interfaces (Jia et al. 2013) releasing DNA species (short artificial oligonucleotides) in response to the output signals generated by the enzyme logic systems resulted in the unique integration of enzyme-based and DNA-based logic elements in highly sophisticated hybrid computing systems (Guz et al. 2016; Mailloux et al. 2015), benefiting from the features

of the enzyme and DNA components. These systems allowed for the logically reversible processing of the initial input signals through the enzyme and DNA reactions, being particularly beneficial for biosensing applications when the restoration of the initial signal pattern from the final output signals is important.

Simple single-output-producing logic gates (e.g., AND, OR, NAND, XOR, etc.) can easily operate as a “one-pot” system, being assembled in a solution or on a transducer surface. However, when more sophisticated logic systems composed of concatenated-branched logic elements and producing several output signals are aimed, the use of a “one-pot” structure becomes difficult because of cross-talking of the logic elements and overlapping of the output signals. This makes the readout of the output signals particularly difficult when optical absorbance is used as the signal transduction method. One of the approaches used to solve the problem and to allow the increasing complexity of the logic systems is the spatial separation of the logic elements and organizing the output signals through physically different channels, for example, in multichannel flow devices (Fratto and Katz 2015, 2016). The application of different transduction methods to follow the output signals is highly important for increasing complexity of the biomolecular logic systems, particularly when the system generates several signals measured separately. An example system comprised of several concatenated logic gates based on a biocatalytic reaction cascade was used to illustrate the advantages of reading the output signals through different transduction mechanisms, including optical absorbance measurements and bioluminescence (Guz et al. 2014). The application of AFM for reading output signals generated by biomolecular logic systems allows extremely high sensitivity, up to the detection of single molecules, because the detection process can be localized on a very small surface area (Pita et al. 2008b; Strack et al. 2009).

Overall, the biomolecular computing systems, and particularly enzyme-based logic gates and networks, can contribute not only to the design of future “biocomputers,” which are not possible

at the present level of technology, but on a short term to the development of novel binary-operating biosensors with various transduction tools of the logically processed bio-signals. It should be noted that the biomolecular information processing systems demonstrated promising results while operating in a biological environment dealing with real biological samples, thus giving rise to the expectations of their real practical applications, particularly in therapeutics and diagnostics, and in forensic investigations. Biochemical computing and logic gate systems based on biomolecules have the potential to revolutionize the field of biosensors and bioactuators. Interfacing biocomputing elements with sensing processes would allow multi-signal analysis followed by biochemical processing of the data, giving a final digital (YES or NO) analytical answer. Such YES/NO information also allows direct coupling of signal processing with signal-responsive materials and chemical actuators to offer a closed-loop “Sense/Act” operation. Biochemical networks can offer robust error-free operation upon appropriate optimization of their components and interconnections. The whole research area may benefit from shifting the interests and motivations from pure computational goals, which are not easily realizable at the present level of technology, to designing an interface between biochemical/biological systems and electronics/materials for logic processing of signals within presently available systems of limited complexity.

## Bibliography

- Abouzar MH, Poghossian A, Cherstvy AG, Pedraza AM, Ingebrandt S, Schöning MJ (2012) Label-free electrical detection of DNA by means of field-effect nanoplate capacitors: experiments and modeling. *Phys Status Solidi A* 209:925–934
- Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266:1021–1024
- Albery WJ, O’Shea GJ, Smith AL (1996) Interpretation and use of Mott-Schottky plots at the semiconductor/electrolyte interface. *J Chem Soc Faraday Trans* 92:4083–4085
- Alfonta L, Katz E, Willner I (2000) Probing antigen-antibody interactions on electrode supports by the biocatalyzed precipitation of an insoluble product. *Anal Chem* 72:927–935
- Alfonta L, Bardea A, Khersonsky O, Katz E, Willner (2001) Chronopotentiometry and faradaic impedance spectroscopy as signal transduction methods for the biocatalytic precipitation of an insoluble product on electrode supports: routes for enzyme sensors, immunosensors and DNA-sensors. *Biosens Bioelectron* 16:675–687
- Alon U (2007) An introduction to systems biology: design principles of biological circuits. Chapman & Hall/CRC Press, London
- Ashkenasy G, Dadon Z, Alesebi S, Wagner N, Ashkenasy N (2011) Building logic into peptide networks: bottom-up and top-down. *Isr J Chem* 51:106–117
- Bakshi S, Zavalov O, Halámk J, Privman V, Katz E (2013) Modularity of biochemical filtering for inducing sigmoid response in both inputs in an enzymatic AND gate. *J Phys Chem B* 117:9857–9865
- Bakshi S, Halámková L, Halámk J, Katz E (2014) Biocatalytic analysis of biomarkers for forensic identification of gender. *Analyst* 139:559–563
- Bardea A, Katz E, Willner T (2000) Probing antigen-antibody interactions on electrode supports by the biocatalyzed precipitation of an insoluble product. *Electroanalysis* 12:1097–1106
- Baron R, Lioubashevski O, Katz E, Niazov T, Willner I (2006) Logic gates and elementary computing by enzymes. *J Phys Chem A* 110:8548–8553
- Baughman RH (1996) Conducting polymer artificial muscles. *Synth Met* 78:339–353
- Benenson Y (2009) Biocomputers: from test tubes to live cells. *Mol BioSyst* 5:675–685
- Benenson Y (2012) Biomolecular computing systems: principles, progress and potential. *Nat Rev Genet* 13:455–468
- Bergmeyer H-U (ed) (1974) Methods of enzymatic analysis. Verlag Chemie GmbH, Weinheim
- Binks BP, Murakami R, Armes SP, Fujii S (2005) Temperature-induced inversion of nanoparticle-stabilized emulsions. *Angew Chem Int Ed* 44:4795–4798
- Bocharova V, Tam TK, Halámk J, Pita M, Katz E (2010) Reversible gating controlled by enzymes at nanostructured interface. *Chem Commun* 46:2088–2090
- Bocharova V, Halámk J, Zhou J, Strack G, Wang J, Katz E (2011) Alert-type biological dosimeter based on enzyme logic system. *Talanta* 85:800–803
- Bychkova V, Shvarev A, Zhou J, Pita M, Katz E (2010) Enzyme logic gate associated with a single responsive microparticle: scaling biocomputing to microsize systems. *Chem Commun* 46:94–96
- Calude CS, Costa JF, Dershowitz N, Freire E, Rozenberg G (eds) (2009) Unconventional computation, Lecture notes in computer science, vol 5715. Springer, Berlin
- Chegel VI, Raitman OA, Lioubashevski O, Shirshov Y, Katz E, Willner I (2002) Redox-switching of electro-refractive, electrochromic and conducting functions of Cu<sup>2+</sup>/polyacrylic acid films associated with electrodes. *Adv Mater* 14:1549–1553
- Chuang M-C, Windmiller JR, Santhosh P, Valdés-Ramírez G, Katz E, Wang J (2011) High-fidelity

- simultaneous determination of explosives and nerve agent threats via a Boolean biocatalytic cascade. *Chem Commun* 47:3087–3089
- de Silva AP (2007) Molecular logic and computing. *Nat Nanotechnol* 2:399–410
- de Silva AP (2013) Molecular logic-based computation. Royal Society of Chemistry, Cambridge, UK
- Duan HW, Wang DY, Sobal NS, Giersig M, Kurth DG, Mohwald H (2005) Magnetic colloidosomes derived from nanoparticle interfacial self-assembly. *Nano Lett* 5:949–952
- Dunford HB (1991) Horseradish peroxidase: structure and kinetic properties, Chapter 1. In: Everse J, Everse KE, Grisham MB (eds) Peroxidases in chemistry and biology, vol 2. CRC Press, Boca Raton, pp 1–24
- Evans AC, Thadani NN, Suh J (2016) Biocomputing nano-platforms as therapeutics and diagnostics. *J Control Release* 240:387–393
- Ezziane Z (2006) DNA computing: applications and challenges. *Nanotechnology* 17:R27–R39
- Fraga H (2008) Firefly luminescence: a historical perspective and recent developments. *Photochem Photobiol Sci* 7:146–158
- Fratto BE, Katz E (2015) Reversible logic gates based on enzyme-biocatalyzed reactions and realized in flow cells – modular approach. *Chem Phys Chem* 16:1405–1415
- Fratto BE, Katz E (2016) Controlled logic gates – switch gate and Fredkin gate based on enzyme-biocatalyzed reactions realized in flow cells. *Chem Phys Chem* 17:1046–1053
- Fujii S, Read ES, Binks BP, Armes SP (2005) Stimulus-responsive emulsifiers based on nanocomposite microgel particles. *Adv Mater* 17:1014–1017
- Fujii S, Armes SP, Binks BP, Murakami R (2006) Stimulus-responsive particulate emulsifiers based on lightly cross-linked poly(4-vinylpyridine)-silica nanocomposite microgels. *Langmuir* 22:6818–6825
- Guz N, Halámk J, Rusling JF, Katz E (2014) A biocatalytic cascade with several output signals – towards biosensors with different levels of confidence. *Anal Bioanal Chem* 406:3365–3370
- Guz N, Fedotova TA, Fratto BE, Schlesinger O, Alfanta L, Kolpashchikov D, Katz E (2016) Bioelectronic interface connecting reversible logic gates based on enzyme and DNA reactions. *ChemPhysChem* 17:2247–2255
- Halámk J, Windmiller JR, Zhou J, Chuang MC, Santhosh P, Strack G, Arugula MA, Chinnapareddy S, Bocharova V, Wang J, Katz E (2010a) Multiplexing of injury codes for the parallel operation of enzyme logic gates. *Analyst* 135:2249–2259
- Halámk J, Bocharova V, Chinnapareddy S, Windmiller JR, Strack G, Chuang M-C, Zhou J, Santhosh P, Ramirez GV, Arugula MA, Wang J, Katz E (2010b) Multi-enzyme logic network architectures for assessing injuries: digital processing of biomarkers. *Mol BioSyst* 6:2554–2560
- Halámk J, Bocharova V, Arugula MA, Strack G, Privman V, Katz E (2011a) Realization and properties of biochemical-computing biocatalytic XOR gate based on enzyme inhibition by a substrate. *J Phys Chem B* 115:9838–9845
- Halámk J, Zhou J, Halámková L, Bocharova V, Privman V, Wang J, Katz E (2011b) Biomolecular filters for improved separation of output signals in enzyme logic systems applied to biomedical analysis. *Anal Chem* 83:8383–8386
- Halámk J, Zavalov O, Halámková L, Korkmaz S, Privman V, Katz E (2012) Enzyme-based logic analysis of biomarkers at physiological concentrations: AND gate with double-sigmoid “filter” response. *J Phys Chem B* 116:4457–4464
- Halámková L, Halámk J, Bocharova V, Wolf S, Mulier KE, Beilman G, Wang J, Katz E (2012) Analysis of biomarkers characteristic of porcine liver injury – from biomolecular logic gates to animal model. *Analyst* 137:1768–1770
- Inzelt G (2008) Conducting polymers. Springer, Berlin
- Jia YM, Duan RX, Hong F, Wang BY, Liu NN, Xia F (2013) Electrochemical biocomputing: a new class of molecular-electronic logic devices. *Soft Matter* 9:6571–6577
- Jin Z, Güven G, Bocharova V, Halámk J, Tokarev I, Minko S, Melman A, Mandler D, Katz E (2012) Electrochemically controlled drug-mimicking protein release from iron-alginate thin-films associated with an electrode. *ACS Appl Mater Interfaces* 4:466–475
- Johannsmann D (2015) The quartz crystal microbalance in soft matter research: fundamentals and modeling. Springer, Heidelberg
- Kahan M, Gil B, Adar R, Shapiro E (2008) Towards molecular computers that operate in a biological environment. *Physica D* 237:1165–1172
- Kang ET, Neoh KG, Tan KL (1998) Polyaniline: a polymer with many interesting intrinsic redox states. *Prog Polym Sci* 23:277–324
- Katz E (ed) (2012a) Biomolecular information processing – from logic systems to smart sensors and actuators. Wiley-VCH, Weinheim
- Katz E (ed) (2012b) Molecular and supramolecular information processing – from molecular switches to unconventional computing. Wiley-VCH, Weinheim
- Katz E (2015) Biocomputing – tools, aims, perspectives. *Curr Opin Biotechnol* 34:202–208
- Katz E (2017) Enzyme-based logic gates and networks with output signals analyzed by various methods. *ChemPhysChem* 18:1688–1713
- Katz E, Halámk J (2014) New approach in forensic analysis – biomolecular computing based analysis of significant forensic biomarkers. *Ann Forensic Res Analysis* 1: article #1002
- Katz E, Minko S (2015) Enzyme-based logic systems interfaced with signal-responsive materials and electrodes. *Chem Commun* 51:3493–3500
- Katz E, Pita M (2009) Biofuel cells controlled by logically processed biochemical signals: towards physiologically regulated bioelectronic devices. *Chem Eur J* 15:12554–12564
- Katz E, Privman V (2010) Enzyme-based logic systems for information processing. *Chem Soc Rev* 39:1835–1857

- Katz E, Lötzbeyer T, Schlereth DD, Schuhmann W, Schmidt H-L (1994) Electrocatalytic oxidation of reduced nicotinamide coenzymes at gold and platinum electrode surfaces modified with a monolayer of pyrroloquinoline quinone. Effect of  $\text{Ca}^{2+}$  cations. *J Electroanal Chem* 373:189–200
- Katz E, Wang J, Privman M, Halámk J (2012) Multi-analyte digital enzyme biosensors with built-in Boolean logic. *Anal Chem* 84:5463–5469
- Katz E, Minko S, Halámk J, MacVittie K, Yancey K (2013) Electrode interfaces switchable by physical and chemical signals for biosensing, biofuel and biocomputing applications. *Anal Bioanal Chem* 405:3659–3672
- Katz E, Poghossian A, Schöning MJ (2017) Enzyme-based logic gates and circuits – analytical applications and interfacing with electronics. *Anal Bioanal Chem* 409:81–94
- Konorov PP, Yafyasov AM, Bogevolnov VB (2006) Field effect in semiconductor-electrolyte interfaces. Princeton University Press, Princeton
- Krämer M, Pita M, Zhou J, Ornatska M, Poghossian A, Schöning MJ, Katz E (2009) Coupling of biocomputing systems with electronic chips: electronic interface for transduction of biochemical information. *J Phys Chem C* 113:2573–2579
- Kramer F, Halámková L, Poghossian A, Schöning MJ, Katz E, Halámk J (2013) Biocatalytic analysis of biomarkers for forensic identification of ethnicity between caucasian and african american groups. *Analyst* 138:6251–6257
- Küttel C, Stemmer A, Wei X (2009) Strain response of polypyrole actuators induced by redox agents in solution. *Sens Actuat B* 141:478–484
- Lahav M, Durkan C, Gabai R, Katz E, Willner I, Welland ME (2001) Redox activation of a polyaniline-coated cantilever: an electro-driven microdevice. *Angew Chem Int Ed* 40:4095–4097
- Lasia A (2014) Semiconductors and Mott-Schottky plots, Chapter 10. In: *Electrochemical impedance spectroscopy and its applications*. Springer, New York, pp 251–255
- Luzinov I, Minko S, Tsukruk VV (2004) Adaptive and responsive surfaces through controlled reorganization of interfacial polymer layers. *Prog Polym Sci* 29:635–698
- Mailloux S, Guz N, Zakharchenko A, Minko S, Katz E (2014a) Majority and minority gates realized in enzyme-biocatalyzed systems integrated with logic networks and interfaced with bioelectronic systems. *J Phys Chem B* 118:6775–6784
- Mailloux S, Zavalov O, Guz N, Katz E, Bocharova V (2014b) Enzymatic filter for improved separation of output signals in enzyme logic systems towards ‘sense and treat’ medicine. *Biomater Sci* 2:184–191
- Mailloux S, Gerasimova YV, Guz N, Kolpashchikov DM, Katz E (2015) Bridging the two worlds: a universal interface between enzymatic and DNA computing systems. *Angew Chem Int Ed* 54:6562–6566
- Manesh KM, Halámk J, Pita M, Zhou J, Tam TK, Santhosh P, Chuang MC, Windmiller JR, Abidin D, Katz E, Wang J (2009) Enzyme logic gates for the digital analysis of physiological level upon injury. *Biosens Bioelectron* 24:3569–3574
- Melnikov D, Strack G, Pita M, Privman V, Katz E (2009) Analog noise reduction in enzymatic logic gates. *J Phys Chem B* 113:10472–10479
- Mermin ND (2007) Quantum computer science: an introduction. Cambridge University Press, Cambridge, UK
- Minko S (2006) Responsive polymer brushes. *J Macromol Sci Polym Rev* 46:397–420
- Molinuss D, Bäcker M, Iken H, Poghossian A, Keusgen M, Schöning MJ (2015) Concept for a biomolecular logic chip with an integrated sensor and actuator function. *Phys Status Solidi A* 212:1382–1388
- Moore GE (1998) Cramming more components onto integrated circuits. *Proc IEEE* 86:82–85
- Morin J-F (2017) Surface and solid-state nanomachines: preparation, assembly and testing. Wiley-VCH, Weinheim. in press (ISBN-13: 978-0470592427).
- Moseley F, Halámk J, Kramer F, Poghossian A, Schöning MJ, Katz E (2014) Enzyme-based reversible CNOT logic gate realized in a flow system. *Analyst* 139:1839–1842
- Motornov M, Zhou J, Pita M, Gopishetty V, Tokarev I, Katz E, Minko S (2008) “Chemical transformers” from nanoparticle ensembles operated with logic. *Nano Lett* 8:2993–2997
- Motornov M, Zhou J, Pita M, Tokarev I, Gopishetty V, Katz E, Minko S (2009a) An integrated multifunctional nanosystem from command nanoparticles and enzymes. *Small* 5:817–820
- Motornov M, Tam TK, Pita M, Tokarev I, Katz E, Minko S (2009b) Switchable selectivity for gating ion transport with mixed polyelectrolyte brushes: Approaching “smart” drug delivery systems. *Nanotechnology* art. #434006
- Munkhjargal M, Matsuura Y, Hatayama K, Miyajima K, Arakawa T, Kudo Mitsubayashi HK (2013) Glucose-sensing and glucose-driven “organic engine” with co-immobilized enzyme membrane toward autonomous drug release systems for diabetes. *Sens Actuat B* 188:831–836
- Niazov T, Baron R, Katz E, Lioubashevski O, Willner I (2006) Concatenated logic gates using four coupled biocatalysts operating in series. *Proc Natl Acad Sci USA* 103:17160–17163
- Otero TF, Sansiñena JM (1995) Artificial muscles based on conducting polymers. *Bioelectrochem Bioenerg* 38:411–414
- Otero TF, Sansiñena JM (1997) Bilayer dimensions and movement in artificial muscles. *Bioelectrochem Bioenerg* 42:117–122
- Otero TF, Sansiñena JM (1998) Soft and wet conducting polymers for artificial muscles. *Adv Mater* 10:491–494
- Passonneau JV, Lowry OH (1993) Chapter 1. In: *Enzymatic analysis: a practical guide*. Humana Press, Totowa, pp 3–22
- Patolsky F, Zayats M, Katz E, Willner I (1999a) Precipitation of an insoluble product on enzyme-monolayer-

- electrodes for biosensor applications: characterization by faradaic impedance spectroscopy, cyclic voltammetry and microgravimetric quartz-crystal-microbalance analyses. *Anal Chem* 71:3171–3180
- Patolsky F, Katz E, Bardea A, Willner I (1999b) Enzyme-linked amplified electrochemical sensing of oligonucleotide-DNA interactions by means of the precipitation of an insoluble product and using impedance spectroscopy. *Langmuir* 15:3703–3706
- Pei Q, Inganlås O (1992) Conjugated polymers and the bending cantilever method: electrical muscles and smart devices. *Adv Mater* 4:277–278
- Pischel U (2010) Advanced molecular logic with memory function. *Angew Chem Int Ed* 49:1356–1358
- Pischel U, Andreasson J, Gust D, Pais VF (2013) Information processing with molecules – quo Vadis? *ChemPhysChem* 14:28–46
- Pita M, Krämer M, Zhou J, Poghossian A, Schöning MJ, Fernández VM, Katz E (2008a) Optoelectronic properties of nanostructured ensembles controlled by biomolecular logic systems. *ACS Nano* 2:2160–2166
- Pita M, Cui L, Gaikwad RM, Katz E, Sokolov I (2008b) Atomic force microscopy study of immunosensor surface to increase sensitivity and scale down size of ELISA-type sensor. *Nanotechnology* 19: art. # 375502
- Pita M, Privman V, Arugula MA, Melnikov D, Bocharova V, Katz E (2011) Towards biochemical filter with sigmoidal response to pH changes: buffered biocatalytic signal transduction. *Phys Chem Chem Phys* 13:4507–4513
- Pita M, Privman M, Katz E (2012) Biocatalytic enzyme networks designed for binary-logic control of smart electroactive nanobiointerfaces. *Top Catal* 55:1201–1216
- Poghossian AS (1992) The super-Nernstian pH sensitivity of  $Ta_2O_5$ -gare ISFETs. *Sens Actuat B* 7:367–370
- Poghossian A, Krämer M, Abouzar MH, Pita M, Katz E, Schöning MJ (2009) Interfacing of biocomputing systems with silicon chips: enzyme logic gates based on field-effect devices. *Procedia Chem* 1:682–685
- Poghossian A, Malzahn K, Abouzar MH, Mehndiratta P, Katz E, Schöning MJ (2011) Integration of biomolecular logic gates with field-effect transducers. *Electrochim Acta* 56:9661–9665
- Poghossian A, Weil M, Cherstvy AG, Schöning MJ (2013) Electrical monitoring of polyelectrolyte multi-layer formation by means of capacitive field-effect devices. *Anal Bioanal Chem* 405:6425–6436
- Poghossian A, Katz E, Schöning MJ (2015a) Enzyme logic AND-reset and OR-reset gates based on a field-effect electronic transducer modified with multi-enzyme membrane. *Chem Commun* 51:6564–6567
- Poghossian A, Bäcker M, Mayer D, Schöning MJ (2015b) Gating capacitive field-effect sensors by the charge of nanoparticle/molecule hybrids. *Nanoscale* 7: 1023–1031
- Privman V, Katz E (2015) Can bio-inspired information processing steps be realized as synthetic biochemical processes? *Phys Status Solidi A* 212:219–228
- Privman V, Strack G, Solenov D, Pita M, Katz E (2008) Optimization of enzymatic biochemical logic for noise reduction and scalability: how many biocomputing gates can be interconnected in a circuit? *J Phys Chem B* 112:11777–11784
- Privman V, Arugula MA, Halámk J, Pita M, Katz E (2009a) Network analysis of biochemical logic for noise reduction and stability: a system of three coupled enzymatic AND gates. *J Phys Chem B* 113:5301–5310
- Privman M, Tam TK, Pita M, Katz E (2009b) Switchable electrode controlled by enzyme logic network system: approaching physiologically regulated bioelectronics. *J Am Chem Soc* 131:1314–1321
- Privman V, Zhou J, Halámk J, Katz E (2010a) Realization and properties of biochemical-computing biocatalytic XOR gate based on signal change. *J Phys Chem B* 114:13601–13608
- Privman V, Halámk J, Arugula MA, Melnikov D, Bocharova V, Katz E (2010b) Biochemical filter with sigmoidal response: increasing the complexity of biomolecular logic. *J Phys Chem B* 114: 14103–14109
- Privman M, Tam TK, Bocharova V, Halámk J, Wang J, Katz E (2011) Responsive interface switchable by logically processed physiological signals – towards “smart” actuators for signal amplification and drug delivery. *ACS Appl Mater Interfaces* 3:1620–1623
- Privman V, Zavalov O, Halámková L, Moseley F, Halámk J, Katz E (2013a) Networked enzymatic logic gates with filtering: new theoretical modeling expressions and their experimental application. *J Phys Chem B* 117:14928–14939
- Privman V, Fratto BE, Zavalov O, Halámk J, Katz E (2013b) Enzymatic AND logic gate with sigmoid response induced by photochemically controlled oxidation of the output. *J Phys Chem B* 117:7559–7568
- Privman V, Zavalov O, Halámková L, Moseley F, Halámk J, Katz E (2013c) Networked enzymatic logic gates with filtering: new theoretical modeling expressions and their experimental application. *J Phys Chem B* 117:14928–14939
- Privman V, Domanskyi S, Mailloux S, Holade Y, Katz E (2014) Kinetic model for a threshold filter in an enzymatic system for bioanalytical and biocomputing applications. *J Phys Chem B* 118:12435–12443
- Raitman OA, Katz E, Willner I, Chegel VI, Popova GV (2001) Photonic transduction of a three-state electronic memory and of electrochemical sensing of NADH using surface plasmon resonance spectroscopy. *Angew Chem Int Ed* 40:3649–3652
- Raitman OA, Katz E, Bückmann AF, Willner I (2002) Integration of polyaniline/polyacrylic acid films and redox-enzymes on electrode supports: an in situ electrochemical/surface plasmon resonance study of the bioelectrocatalyzed oxidation of glucose or lactate in the integrated bioelectrocatalytic systems. *J Am Chem Soc* 124:6487–6496
- Rehm HA (ed) (2009) Alginates: biology and applications. Springer, Heidelberg

- Rinaudo K, Bleris L, Maddamsetti R, Subramanian S, Weiss R, Benenson Y (2007) A universal RNAi-based logic evaluator that operates in mammalian cells. *Nat Biotechnol* 25:795–801
- Saleh N, Phenrat T, Sirk K, Dufour B, Ok J, Sarbu T, Matyjaszewski K, Tilton RD, Lowry GV (2005) Adsorbed triblock copolymers deliver reactive iron nanoparticles to the oil/water interface. *Nano Lett* 5:2489–2494
- Schopf G, Koßmehl G (1997) Polythiophenes: electrically conductive polymers. Springer, Berlin
- Shinwari MW, Deen MJ, Landheer D (2007) Study of the electrolyte-insulator-semiconductor field-effect transistor (EISFET) with applications in biosensor design. *Microelectron Reliab* 47:2025–2057
- Siqueira JR, Abouzar MH, Bäcker M, Zucolotto V, Poghossian A, Oliveira ON, Schöning MJ (2009) Carbon nanotubes in nanostructured films: potential application as amperometric and potentiometric field-effect (bio-)chemical sensors. *Phys Status Solidi A* 206:462–467
- Skotheim TA, Elsenbaumer RL, Reynolds JR (eds) (1998) Handbook of conducting polymers. Marcel Dekker, New York
- Smela E, Inganäs O, Lundstrom I (1995) Controlled folding of micrometer-size structures. *Science* 268:1735–1738
- Stojanovic MN, Stefanovic D (2011) Chemistry at a higher level of abstraction. *J Comput Theor Nanosci* 8:434–440
- Stojanovic MN, Stefanovic D, Rudchenko S (2014) Exercises in molecular computing. *Acc Chem Res* 47:1845–1852
- Strack G, Pita M, Ornatska M, Katz E (2008a) Boolean logic gates using enzymes as input signals. *ChemBioChem* 9:1260–1266
- Strack G, Ornatska M, Pita M, Katz E (2008b) Biocomputing security system: concatenated enzyme-based logic gates operating as a biomolecular keypad lock. *J Am Chem Soc* 130:4234–4235
- Strack G, Chinnapareddy S, Volkov D, Halámk J, Pita M, Sokolov I, Katz E (2009) Logic networks based on immunorecognition processes. *J Phys Chem B* 113:12154–12159
- Strack G, Bocharova V, Arugula MA, Pita M, Halámk J, Katz E (2010) Artificial muscle reversibly controlled by enzyme reactions. *J Phys Chem Lett* 1:839–843
- Stuart MAC, Huck TS W, Genzer J, Müller M, Ober C, Stamm M, Sukhorukov GB, Szleifer I, Tsukruk VV, Urban M, Winnik F, Zauscher S, Luzinov I, Minko S (2010) Emerging applications of stimuli-responsive polymer materials. *Nat Mater* 9:101–113
- Szacilowski K (2008) Digital information processing in molecular systems. *Chem Rev* 108:3481–3548
- Szacilowski K (2012) Infochemistry. Wiley, Chichester
- Tam TK, Zhou J, Pita M, Ornatska M, Minko S, Katz E (2008a) Biochemically controlled bioelectrocatalytic interface. *J Am Chem Soc* 130:10888–10889
- Tam TK, Ornatska M, Pita M, Minko S, Katz E (2008b) Polymer brush-modified electrode with switchable and tunable redox activity for bioelectronic applications. *J Phys Chem C* 112:8438–8445
- Tam TK, Pita M, Katz E (2009a) Enzyme logic network analyzing combinations of biochemical inputs and producing fluorescent output signals: towards multi-signal digital biosensors. *Sensors Actuators B* 140:1–4
- Tam TA, Pita M, Ornatska M, Katz E (2009b) Biofuel cell controlled by enzyme logic network – approaching physiologically regulated devices. *Bioelectrochemistry* 76:4–9
- Tam TK, Pita M, Motornov M, Tokarev I, Minko S, Katz E (2010a) Modified electrodes with switchable selectivity for cationic and anionic redox species. *Electroanalysis* 22:35–40
- Tam TK, Pita M, Motornov M, Tokarev I, Minko S, Katz E (2010b) Electrochemical nanotransistor from mixed polymer brush. *Adv Mater* 22:1863–1866
- Tam TK, Pita M, Trotsenko O, Motornov M, Tokarev I, Halámk J, Minko S, Katz E (2010c) Reversible “closing” of an electrode interface functionalized with a polymer brush by an electrochemical signal. *Langmuir* 26:4506–4513
- Tokarev I, Gopishetty V, Zhou J, Pita M, Motornov M, Katz E, Minko S (2009) Stimuli-responsive hydrogel membranes coupled with biocatalytic processes. *ACS Appl Mater Interfaces* 1:532–536
- Unger R, Moult J (2006) Towards computing with proteins. *Proteins* 63:53–64
- Vlasov YG, Tarantov YA, Bobrov PV (2003) Analytical characteristics and sensitivity mechanisms of electrolyte-insulator-semiconductor system-based chemical sensors - a critical review. *Anal Bioanal Chem* 376:788–796
- Wang J (2013) Nanomachines: fundamentals and applications. Wiley-VCH, Weinheim
- Wang J, Katz E (2010) Digital biosensors with built-in logic for biomedical applications – biosensors based on biocomputing concept. *Anal Bioanal Chem* 398:1591–1603
- Wang J, Katz E (2011) Digital biosensors with built-in logic for biomedical applications. *Israel J Chem* 51:141–150
- Wang Y, Knoll W (2006) In situ electrochemical and surface plasmon resonance (SPR) studies of aniline-carboxylated aniline copolymers. *Anal Chim Acta* 558:150–157
- Wang L-X, Li X-G, Yang Y-L (2001) Preparation, properties and applications of polypyrroles. *React Funct Polym* 47:125–139
- Wang X, Zhou J, Tam TK, Katz E, Pita M (2009) Switchable electrode controlled by Boolean logic gates using enzymes as input signals. *Bioelectrochemistry* 77:69–73
- Weng WH, Wang CW, Pang ST, Pan TM (2016) Enzymatic glucose biosensor based on  $TbY_xO_y$  electrolyte-insulator-semiconductor. *J Electrochem Soc* 163: B445–B452
- Willner I, Arad G, Katz E (1998) A biofuel cell based on pyrroloquinoline quinone and microperoxidase-11

- monolayer-functionalized electrodes. *Bioelectrochem Bioenerg* 44:209–214
- Wilson R, Turner APF (1992) Glucose oxidase: an ideal enzyme. *Biosens Bioelectron* 7:165–185
- Windmiller JR, Santhosh P, Katz E, Wang J (2011) Bioelectronic system for the control and readout of enzyme logic gates. *Sens Actuat B* 155:206–213
- Yang J, Song ZC, Liu S, Zhang Q, Zhang C (2016) Dynamically arranging gold nanoparticles on DNA origami for molecular logic gates. *ACS Appl Mater Interfaces* 8:22451–22456
- Yao DB, Li H, Guo YJ, Zhou X, Xiao SY, Liang HJ (2016) A pH-responsive DNA nanomachine-controlled catalytic assembly of gold nanoparticles. *Chem Commun* 52:7556–7559
- Yu X, Lian WJ, Zhang JH, Liu HY (2016) Multi-input and -output logic circuits based on bioelectrocatalysis with horseradish peroxidase and glucose oxidase immobilized in multi-responsive copolymer films on electrodes. *Biosens Bioelectron* 80:631–639
- Zavalov O, Bocharova V, Privman V, Katz E (2012) Enzyme-based logic: OR gate with double-sigmoid filter response. *J Phys Chem B* 116:9683–9689
- Zhao J, Luo L, Yang X, Wang E, Dong S (1999) Determination of surface  $pK_a$  of  $\text{HS}(\text{CH}_2)_{10}\text{COOH}$  SAMs using an electrochemical titration method. *Electroanalysis* 11:1108–1111
- Zhou J, Arugula MA, Halámek J, Pita M, Katz E (2009a) Enzyme-based NAND and NOR logic gates with modular design. *J Phys Chem B* 113:16065–16070
- Zhou J, Tam TK, Pita M, Ornatska M, Minko S, Katz E (2009b) Bioelectrocatalytic system coupled with enzyme-based biocomputing ensembles performing Boolean logic operations: approaching “smart” physiologically controlled biointerfaces. *ACS Appl Mater Interfaces* 1:144–149
- Zhou N, Windmiller JR, Valdés Ramírez G, Zhou M, Halámek J, Katz E, Wang J (2011a) Enzyme-based NAND gate for rapid electrochemical screening of traumatic brain injury in serum. *Anal Chim Acta* 703:94–100
- Zhou J, Halámek J, Bocharova V, Wang J, Katz E (2011b) Bio-logic analysis of injury biomarker patterns in human serum samples. *Talanta* 83:955–959



## DNA Computing

Martyn Amos

Department of Computing and Mathematics,  
Manchester Metropolitan University, Manchester,  
UK

### Article Outline

Definition of the Subject

Introduction

The DNA Molecule

The First DNA Computation

Models of DNA Computation

Subsequent Work

Assessment

Future Directions

Bibliography

molecules as fundamental components of computing devices. It draws on concepts and expertise from fields as diverse as chemistry, computer science, molecular biology, physics, and mathematics. Although its theoretical history dates back to the late 1950s, the notion of computing with molecules was only physically realized in 1994, when Leonard Adleman demonstrated in the laboratory the solution of a small instance of a well-known problem in combinatorics using standard tools of molecular biology. Since this initial experiment, interest in DNA computing has increased dramatically, and it is now a well-established area of research. As we expand our understanding of how biological and chemical systems process information, opportunities arise for new applications of molecular devices in bioinformatics, nanotechnology, engineering, the life sciences, and medicine.

### Introduction

In the late 1950s, the physicist Richard Feynman first proposed the idea of using living cells and molecular complexes to construct “submicroscopic computers.” In his famous talk *There’s Plenty of Room at the Bottom* (Feynman 1961), Feynman discussed the problem of “manipulating and controlling things on a small scale,” thus founding the field of nanotechnology. Although he concentrated mainly on information storage and molecular manipulation, Feynman highlighted the potential for biological systems to act as small-scale information processors:

The biological example of writing information on a small scale has inspired me to think of something that should be possible. Biology is not simply writing information; it is doing something about it. A biological system can be exceedingly small. Many of the cells are very tiny, but they are very active; they manufacture various substances; they walk around; they wiggle; and they do all kinds of marvelous things - all on a very small scale. Also, they store information. Consider the possibility that we too can make a thing very small which does what

### Definition of the Subject

DNA computing (or, more generally, biomolecular computing) is a relatively new field of study that is concerned with the use of biological

we want - that we can manufacture an object that maneuvers at that level! (Feynman 1961)

## Early Work

Since the presentation of Feynman's vision, there has been a steady growth of interest in performing computations at a molecular level. In 1982, Charles Bennett (1982) proposed the concept of a "Brownian computer" based around the principle of reactant molecules touching, reacting, and effecting state transitions due to their random Brownian motion. Bennett developed this idea by suggesting that a Brownian Turing machine could be built from a macromolecule such as RNA. "Hypothetical enzymes," one for each transition rule, catalyze reactions between the RNA and chemicals in its environment, transforming the RNA into its logical successor.

In the same year, Conrad and Liberman developed this idea further in (Conrad and Liberman 1982), in which the authors describe parallels between physical and computational processes (e.g., biochemical reactions being employed to implement basic switching circuits). They introduce the concept of molecular level "word processing" by describing it in terms of transcription and translation of DNA, RNA processing, and genetic regulation. However, the paper lacks a detailed description of the biological mechanisms highlighted and their relationship with "traditional" computing. As the authors themselves acknowledge, "our aspiration is not to provide definitive answers ... but rather to show that a number of seemingly disparate questions must be connected to each other in a fundamental way" (Conrad and Liberman 1982).

In (Conrad 1985), Conrad expanded on this work, showing how the information processing capabilities of organic molecules may, in theory, be used in place of digital switching components. Particular enzymes may alter the three-dimensional structure (or *conformation*) of other *substrate* molecules. In doing so, the enzyme switches the *state* of the substrate from one to another. The notion of *conformational computing* (q.v.) suggests the possibility of a potentially rich and powerful computational architecture.

Following on from the work of Conrad et al., Arkin and Ross show how various logic gates may be constructed using the computational properties of enzymatic reaction mechanisms (Arkin and Ross 1994) (see Dennis Bray's article (Bray 1995) for a review of this work). In (Bray 1995), Bray also describes work (Hjelmfelt et al. 1991, 1993) showing how chemical "neurons" may be constructed to form the building blocks of logic gates.

## Motivation

We have made huge advances in machine miniaturization since the days of room-sized computers, and yet the underlying computational framework (the *von Neumann architecture*) has remained constant. Today's supercomputers still employ the kind of sequential logic used by the mechanical "dinosaurs" of the 1940s (Campbell-Kelly and Aspray 2004).

There exist two main barriers to the continued development of "traditional," silicon-based computers using the von Neumann architecture. One is inherent to the machine architecture, and the other is imposed by the nature of the underlying *computational substrate*. A computational substrate may be defined as *a physical substance acted upon by the implementation of a computational architecture*. Before the invention of silicon integrated circuits, the underlying substrates were bulky and unreliable. Of course, advances in miniaturization have led to incredible increases in processor speed and memory access time. However, there is a limit to how far this miniaturization can go. Eventually "chip" fabrication will hit a wall imposed by the *Heisenberg uncertainty principle* (HUP). When chips are so small that they are composed of components a few atoms across, quantum effects cause interference. The HUP states that the act of observing these components affects their behavior. As a consequence, it becomes impossible to know the exact state of a component without fundamentally changing its state.

The second limitation is known as the *von Neumann bottleneck*. This is imposed by the need for the central processing unit (CPU) to transfer instructions and data to and from the

main memory. The route between the CPU and memory may be visualized as a two-way road connecting two towns. When the number of cars moving between towns is relatively small, traffic moves quickly. However, when the number of cars grows, the traffic slows down and may even grind to a complete standstill. If we think of the cars as units of information passing between the CPU and memory, the analogy is complete. Most computation consists of the CPU fetching from memory and then executing one instruction after another (after also fetching any data required). Often, the execution of an instruction requires the storage of a result in memory. Thus, the speed at which data can be transferred between the CPU and memory is a limiting factor on the speed of the whole computer.

Some researchers are now looking beyond these boundaries and are investigating entirely new computational architectures and substrates. These developments include *quantum computing* (q.v.), *optical computing* (q.v.), *nanocomputers* (q.v.), and biomolecular computers. In 1994, interest in molecular computing intensified with the first report of a successful nontrivial molecular computation. Leonard Adleman of the University of Southern California effectively founded the field of DNA computing by describing his technique for performing a massively parallel random search using strands of DNA (Adleman 1994). In what follows we give an in-depth description of Adleman's seminal experiment, before describing how the field has evolved in the years that followed. First, though, we must examine more closely the structure of the DNA molecule in order to understand its suitability as a computational substrate.

## The DNA Molecule

Ever since ancient Greek times, man has suspected that the features of one generation are passed on to the next. It was not until Mendel's work on garden peas was recognized (Stubbe 1972) that scientists accepted that both parents contribute material that determines the characteristics of their offspring. In the early twentieth

century, it was discovered that *chromosomes* make up this material. Chemical analysis of chromosomes revealed that they are composed of both *protein* and *deoxyribonucleic acid* or *DNA*. The question was, which substance carries the genetic information? For many years, scientists favored protein because of its greater complexity relative to that of DNA. Nobody believed that a molecule as simple as DNA, composed of only four sub-units (compared to 20 for protein), could carry complex genetic information.

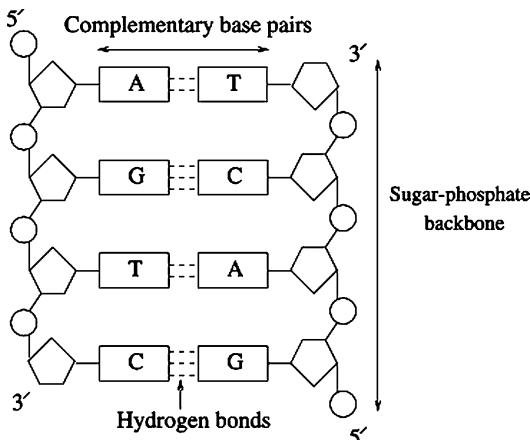
It was not until the early 1950s that most biologists accepted the evidence showing that it is in fact DNA that carries the genetic code. However, the physical structure of the molecule and the hereditary mechanism were still far from clear.

In 1951, the biologist James Watson moved to Cambridge to work with a physicist, Francis Crick. Using data collected by Rosalind Franklin and Maurice Wilkins at King's College, London, they began to decipher the structure of DNA. They worked with models made out of wire and sheet metal in an attempt to construct something that fitted the available data. Once satisfied with their double helix model, they published the paper (Watson and Crick 1953b; also see Watson and Crick 1953a) that would eventually earn them (and Wilkins) the Nobel Prize for Physiology or Medicine in 1962.

## DNA Structure

DNA (deoxyribonucleic acid) (Watson et al. 1987) encodes the genetic information of cellular organisms. It consists of *polymer chains*, commonly referred to as DNA *strands*. Each strand may be viewed as a chain of *nucleotides*, or *bases*, attached to a sugar phosphate "backbone." An  $n$ -letter sequence of consecutive bases is known as an  $n$ -mer or an *oligonucleotide* (commonly abbreviated to "oligo") of length  $n$ . Strand lengths are measured in *base pairs* (b.p.).

The four DNA nucleotides are adenine, guanine, cytosine, and thymine, commonly abbreviated to *A*, *G*, *C*, and *T*, respectively. Each strand, according to chemical convention, has a 5' and a 3' end; thus, any single strand has a natural orientation (Fig. 1). The classical double helix of DNA is formed when two separate strands bond.



**DNA Computing, Fig. 1** Structure of double-stranded DNA (From Amos 2005)

Bonding occurs by the pairwise attraction of bases; *A* bonds with *T* and *G* bonds with *C*. The pairs (*A*, *T*) and (*G*, *C*) are therefore known as *complementary base pairs*. The two pairs of bases form *hydrogen bonds* between each other, two bonds between *A* and *T*, and three between *G* and *C*.

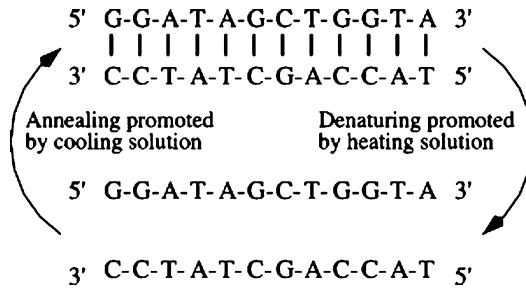
The bonding process, known as *annealing*, is fundamental to our implementation. A strand will only anneal to its complement if they have opposite polarities. Therefore, one strand of the double helix extends from 5' to 3' and the other from 3' to 5', as depicted in Fig. 1.

### Operations on DNA

Some (but not all) DNA-based computations apply a specific sequence of biological operations to a set of strands. These operations are all commonly used by molecular biologists, and we now describe them in more detail.

### Synthesis

Oligonucleotides may be synthesized to order by a machine the size of a microwave oven. The synthesizer is supplied with the four nucleotide bases in solution, which are combined according to a sequence entered by the user. The instrument makes millions of copies of the required oligo and places them in solution in a small vial.



**DNA Computing, Fig. 2** DNA melting and annealing (From Amos 2005)

### Denaturing, Annealing, and Ligation

Double-stranded DNA may be dissolved into single strands (or *denatured*) by heating the solution to a temperature determined by the composition of the strand (Breslauer et al. 1986). Heating breaks the hydrogen bonds between complementary strands (Fig. 2). *Annealing* is the reverse of melting, whereby a solution of single strands is cooled, allowing complementary strands to bind together (Fig. 2).

In double-stranded DNA, if one of the single strands contains a discontinuity (i.e., one nucleotide is not bonded to its neighbor), then this may be repaired by DNA *ligase* (Brown 1993). This particular enzyme is useful for DNA computing, as it allows us to create a unified strand from several strands bound together by their respective complements. Ligase therefore acts as a molecular “cement” or “mortar,” binding together several strands into a single strand.

### Separation of Strands

Separation is a fundamental operation and involves the extraction from a test tube of any *single* strands containing a specific short sequence (e.g., extract all strands containing the sequence *GCTA*). For this, we may use a “molecular sieving” process known as *affinity purification*. If we want to extract from a solution single strands containing the sequence *x*, we may first create many copies of its complement, *x̄*. We attach to these oligos biotin molecules (a process known as “biotinylation”) which in turn bind to a fixed matrix. If we pour the contents of the test tube over this matrix, strands containing *x* will anneal

to the anchored complementary strands. Washing the matrix removes all strands that *did not* anneal, leaving only strands containing  $x$ . These may then be removed from the matrix.

### Gel Electrophoresis

*Gel electrophoresis* is an important technique for sorting DNA strands by size (Brown 1993). Electrophoresis is the movement of charged molecules in an electric field. Since DNA molecules carry a negative charge, when placed in an electric field, they tend to migrate toward the positive pole. The rate of migration of a molecule in an *aqueous* solution depends on its shape and electric charge. Since DNA molecules have the same charge per unit length, they all migrate at the same speed in an aqueous solution. However, if electrophoresis is carried out in a *gel* (usually made of agarose, polyacrylamide, or a combination of the two), the migration rate of a molecule is also affected by its *size*. This is due to the fact that the gel is a dense network of pores through which the molecules must travel. Smaller molecules therefore migrate faster through the gel, thus sorting them according to size. In order to sort strands, the DNA is placed in a well cut out of the gel and a charge applied. After running the gel, the results are visualized by staining the DNA with dye and then viewing the gel under ultraviolet light. Bands of NDA of a specific length may then be cut out of the gel and soaked to free the strands (which may then be used again in subsequent processing steps).

### PCR

The DNA *polymerases* perform several functions, including the repair and duplication of DNA. Given a short *primer* (or “tag”) oligo, in the presence of nucleotide triphosphates (i.e., “spare” nucleotides), the polymerase extends the primer if and only if the primer is bound to a longer *template* strand.

Primer extension is fundamental to the *polymerase chain reaction* or PCR (Mullis et al. 1994). PCR is a process that quickly amplifies the amount of DNA in a given solution. PCR employs polymerase to make copies of a specific region (or *target sequence*) of DNA that lies between two *known* sequences. Note that this

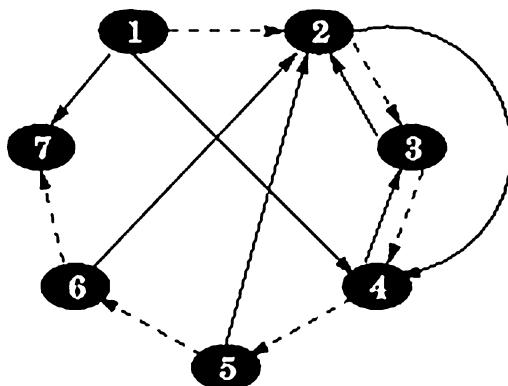
target sequence (which may be up to around 3,000 b.p. long) can be unknown ahead of time. In order to amplify template DNA with known regions (perhaps at either end of the strands), we first design forward and backward primers (i.e., primers that go from 5' to 3' on each strand). We then add a large excess (relative to the amount of DNA being replicated) of primer to the solution and heat it to denature the double-stranded template. Cooling the solution then allows the primers to anneal to their target sequences. We then add the polymerase, which extends the primers, forming an identical copy of the template DNA.

If we start with a single template, then of course we now have two copies. If we then repeat the cycle of heating, annealing, and polymerizing, it is clear that this approach yields an exponential number of copies of the template (since the number of strands doubles after each cycle). A typical number of cycles would be perhaps 35, yielding (assuming a single template) around 68 billion copies of the *target sequence* (e.g., a gene).

### The First DNA Computation

We now describe in detail the first ever successful computation performed using strands of DNA. This experiment was carried out in 1994 by Leonard Adleman of the University of Southern California. Adleman was already a highly distinguished computer scientist prior to this work, having been awarded, along with Rivest and Shamir, the 2002 ACM Turing Award for the development of the RSA public-key encryption scheme (Rivest et al. 1978).

Adleman utilized the incredible storage capacity of DNA to implement a brute-force algorithm for the directed Hamiltonian path problem (HPP) (Adleman 1994). The HPP involves finding a path through a graph (or “network”) that visits each vertex (“node”) exactly once. For an example application, consider a salesperson who wishes to visit several cities connected by rail links. In order to save time and money, she would like to know if there exists an itinerary that visits every city precisely once. We model this situation by constructing a graph where vertices represent



**DNA Computing, Fig. 3** Instance of the HPP solved by Adleman (From Amos 2005)

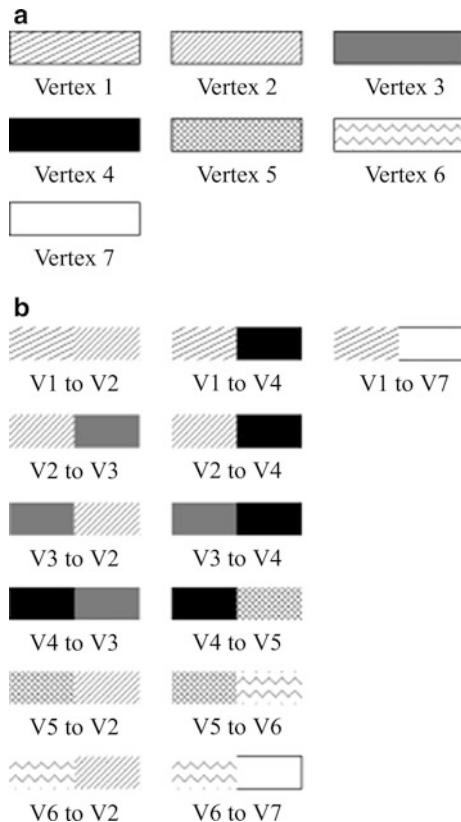
cities and edges the rail links. The HPP is a classic problem in the study of *complex networks and graph theory* (q.v.) (Gibbons 1985) and belongs to the class of problems referred to as *NP-complete* (Garey and Johnson 1979). To practitioners in the field of *computational complexity* (q.v.), the NP-complete problems are the most interesting, since they are among the hardest known problems, and include many problems of great theoretical and practical significance, such as network design, scheduling, and data storage.

The instance of the HPP that Adleman solved is depicted in Fig. 3, with the unique Hamiltonian path (HP) highlighted by a dashed line. Adleman's approach was simple:

1. Generate strands encoding random paths such that the Hamiltonian path (HP) is represented with high probability. The quantities of DNA used far exceeded those necessary for the small graph under consideration, so it is likely that *many* strands encoding the HP were present.
2. Remove all strands that do not encode the HP.
3. Check that the remaining strands encode a solution to the HPP.

The individual steps were implemented as follows:

**Stage 1:** Each vertex and edge was assigned a distinct 20-base sequence of DNA (Fig. 4a). This implies that strands encoding an HP were



**DNA Computing, Fig. 4** Adleman's scheme for encoding path-schematic representation of oligos (From Amos 2005)



**DNA Computing, Fig. 5** Example path created in Adleman's scheme (From Amos 2005)

of length 140 b.p., since there are seven vertices in the problem instance. Sequences representing edges act as "splints" between strands representing their endpoints (Fig. 4b).

In formal terms, the sequence associated with an edge  $i \rightarrow j$  is the 3' 10"-mer of the sequence representing  $v_i$  followed by the 5' 10"-mer of the sequence representing  $v_j$ . These oligos were then combined to form strands encoding random paths through the graph. An (illegal) example path ( $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$ ) is depicted in Fig. 5.



**DNA Computing, Fig. 6** Unique Hamiltonian path (From Amos 2005)

Fixed amounts (50 pmol) of each oligo were synthesized and mixed together and then ligase was added to seal any backbone nicks and form completely unified strands from the shorter “building blocks.” At the end of this reaction, it is assumed that a complete strand representing the HP is present with high probability. This approach solves the problem of generating an exponential number of different paths using a polynomial number of initial oligos Fig. 6.

**Stage 2:** PCR was first used to massively amplify the population of strands encoding paths starting at  $v_1$  and ending at  $v_7$ . These strands were amplified to such a massive extent that they effectively “overwhelmed” any strands that did not start and end at the correct points.

Next, strands that do not encode paths containing exactly  $n$  visits were removed. The product of the PCR amplification was run on an agarose gel to isolate strands of length 140 b.p. A series of affinity purification steps was then used to isolate strands encoding paths that visited each vertex exactly once.

**Stage 3:** PCR was used to identify the strand encoding the unique HP that this problem instance provides. For an  $n$ -vertex graph, we run  $n - 1$  PCR reactions, with the strand representing  $v_1$  as the left primer and the complement of the strand representing  $v_i$  as the right primer in the  $i$ th lane. The presence of molecules encoding the unique HP depicted in Fig. 3 should produce bands of length 40, 60, 80, 100, 120, and 140 b.p. in lanes 1 through 6, respectively. This is exactly what Adleman observed.

## Models of DNA Computation

Although Adleman provided the impetus for subsequent work on DNA computing, his algorithm was not expressed within a formal model

of computation. Richard Lipton realized that Adleman’s approach, though seminal, was limited in that it was specific to solving the HPP. Lipton proposed extending Adleman’s algorithm “in a way that allows biological computers to potentially radically change the way we do all computations, not just HPPs” (Lipton 1995).

### Satisfiability Model

Lipton’s article described a methodology for solving the *satisfiability problem* (SAT) (Cook 1971) using DNA. In terms of *computational complexity* (q.v.), SAT is the “benchmark” NP-complete problem and may be phrased as follows: given a finite set  $V = \{v_1, v_2, \dots, v_n\}$  of logical variables, we define a *literal* to be a variable,  $v_i$ , or its complement,  $\bar{v}_i$ . If  $v_i$  is *true*, then  $\bar{v}_i$  is *false* and vice versa. We define a *clause*,  $C_j$ , to be a set of literals  $\{v_1^j, v_2^j, \dots, v_l^j\}$ . An instance,  $I$ , of SAT consists of a set of clauses. The problem is to assign a Boolean value to each variable in  $V$  such that at least one variable in each clause has the value *true*. If this is the case, we may say that  $I$  has been *satisfied*.

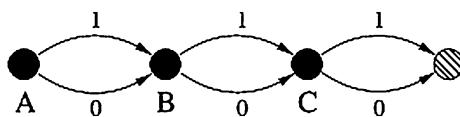
By showing how molecular computers could be applied to this archetypal NP-complete problem, Lipton hoped to show how *any* difficult problem may be solved using this approach. The underlying principle was the same as Adleman’s approach: generate all possible solutions to the problem and then gradually *filter out* strands until any remaining strands *must* be a solution to the problem. Lipton proposed solving an instance of SAT by starting with a tube containing strands representing all possible assignments to its variables.

Lipton’s main contribution lay in his method for encoding *arbitrary* binary strings as strands of DNA. If we only need a small number different strands as the foundation of our computer, then it would be fairly cheap and quick to specify and order them to be individually synthesized.

However, Lipton realized that this approach would quickly become infeasible as the number of variables grew. One of the characteristics of the NP-complete problems is that for even a small increase in the problem size (in this case, the number of variables), the number of possible solutions rises exponentially. For any nontrivial problem, Lipton would require a prohibitively expensive number of “one-off” strands. For example, if Lipton wanted to solve a problem with ten variables, he would need to order  $2^{10} = 1,024$  individual strands.

Lipton needed a way of encoding an *exponential*-sized pool of starting strands, using only a polynomial number of “building block” strands. Just as Adleman encoded a large number of possible paths through a graph, using a small number of node and edge strands, Lipton wanted to build a large number of assignment strands, using a small number of “variable” strands. His key insight was that an arbitrary binary string (in this case encoding the values of a set of variables) could be represented as a path through a graph, where each node represented one particular bit (or variable).

Figure 7 shows an example of a graph to encode a string of three bits (variables), each represented by a node labeled A, B, or C. Each node has two edges emanating from it, labeled either 1 or 0 (the edges leading out of node C lead to a “dummy” node, which just acts like a railway buffer to end the paths). Any three-bit string can be represented by a path built by taking one of the two possible paths at each node, the value of each bit being defined by the label of the edge that is taken at each step. For example, if we only ever take the “top” path at each branch, we encode the string 111, and if we only ever take the “bottom” path, we end up with 000. A path that goes “top, bottom, top” encodes the string 101; “top, top, bottom” encodes 110; and so on.



**DNA Computing, Fig. 7** Lipton’s graph representation for a three-bit binary string

The power of this encoding is that, just like Adleman’s approach, we only have to generate one sequence for each node (including the dummy node) and one for each edge. With the correct Watson-Crick encoding, random paths through the graph form spontaneously, just like in Adleman’s Hamiltonian path experiment. Using this approach, Lipton believed he could be confident of starting off with a tube containing all possible binary strings of a given length. Each strand in the tube would encode a possible assignment of values for a set of variables. The next stage was to remove strands encoding those assignments that did *not* result in satisfaction of the formula to be solved.

Because the general form of a SAT formula is a set of clauses combined with the AND operation, it follows that each clause *must* be satisfied: if evaluating only a single clause results in a value of 0, then the *whole* formula evaluates to 0, and the formula is not satisfied. Because the variables in each clause are separated by the OR operation, it only takes a single variable to take the value 1 for the whole clause to be satisfied. Lipton’s algorithm was very straightforward: he would proceed one clause at a time, looking at each component of it in turn and keeping only the strands encoding a sequence of bits that satisfied that particular clause. The “winning” strands would then go forward to the *next* clause, where the process would repeat, until there were no more clauses to examine. At the end of this procedure, if Lipton had *any* strands left in his tube, then he would know with certainty that the formula was satisfiable, since only strings satisfying every clause would have survived through the successive filters.

We now show how Lipton’s method can be *formally* expressed. In all *filtering* models of DNA computing, a computation consists of a sequence of operations on finite *multi-sets* of strings. Multi-sets are sets that may contain more than one copy of the same element. It is normally the case that a computation begins and terminates with a single multi-set. Within the computation, by applying legal operations of a model, several multi-sets may exist at the same time. We define operations on multi-sets shortly, but first consider the nature of an *initial set*.

An initial multi-set consists of strings which are typically of length  $O(n)$  where  $n$  is the problem size. As a subset, the initial multi-set should include all possible solutions (each encoded by a string) to the problem to be solved. The point here is that the superset, in any implementation of the model, is supposed to be relatively easy to generate as a starting point for a computation. The computation then proceeds by *filtering out* strings which cannot be a solution.

Within one possible model (Adleman 1995), the following operations are available on sets of strings over some alphabet  $\alpha$ :

- *Separate*( $T, S$ ). Given a set  $T$  and a substring  $S$ , create two new sets  $+(T, S)$  and  $-(T, S)$ , where  $+(T, S)$  is all strings in  $T$  containing  $S$  and  $-(T, S)$  is all strings in  $T$  not containing  $S$ .
- *Merge*( $T_1, T_2, \dots, T_n$ ). Given set  $T_1, T_2, \dots, T_n$ , create  $\cup(T_1, T_2, \dots, T_n) = T_1 \cup T_2 \cup \dots \cup T_n$ .
- *Detect*( $T$ ). Given a set  $T$ , return *true* if  $T$  is nonempty, otherwise return *false*.

For example, given  $\alpha = \{A, B, C\}$ , the following algorithm returns *true* only if the initial multi-set contains a string composed entirely of “A”s:

```
Input(T)
T ← −(T, B)
T ← −(T, C)
Output(detect(T))
```

Although Lipton does not explicitly define his operation set in (Lipton 1995), his solution to SAT may be phrased in terms of the operations above, described by Adleman in (Adleman 1995). Lipton employs the *merge*, *separate*, and *detect* operations described above. The initial set  $T$  contains many strings, each encoding a single  $n$ -bit sequence. All possible  $n$ -bit sequences are represented in  $T$ . The algorithm proceeds as follows:

1. Create initial set,  $T$
2. For each clause do begin
3. For each literal  $v_i$  do begin
4. if  $v_i = x_j$  extract from  $T$  strings encoding  $v_i = 1$  else extract from  $T$  strings encoding

- $v_i = 0$
5. End for
6. Create new set  $T$  by merging extracted strings
7. End for
8. If  $T$  nonempty then  $I$  is satisfiable

The pseudo-code algorithm may be expressed more formally thus:

```
1. Input(T)
2. for  $a = 1$  to  $|I|$  do begin
3.   for  $b = 1$  to  $|C_a|$  do begin
4.     if  $v_b^a = x_j$  then  $T_b \leftarrow + (T, v_b^a = 1)$ 
        else  $T_b \leftarrow + (T, v_b^a = 0)$ 
5.   end for
6.    $T \leftarrow \text{merge}(T_1, T_2, \dots, T_b)$ 
7. end for
8. Output(detect(T))
```

Step 1 generates all possible  $n$ -bit strings. Then, for each clause  $C_a = \{v_1^a, v_2^a, \dots, v_l^a\}$  (Step 2), we perform the following steps. For each literal  $v_b^a$  (Step 3), we operate as follows: if  $v_b^a$  computes the positive form, then we extract from  $T$  all strings encoding 1 at position  $v_b^a$ , placing these strings in  $T_b$ ; if  $v_b^a$  computes the negative form, we extract from  $T$  all strings encoding 0 at position  $v_b^a$ , placing these strings in  $T_b$  (Step 4); after  $l$  iterations, we have satisfied every variable in clause  $C_a$ ; and we then create a new set  $T$  from the union of sets  $T_1, T_2, \dots, T_b$  (Step 6) and repeat these steps for clause  $C_a + 1$  (Step 7). If any strings remain in  $T$  after all clauses have been operated upon, then  $I$  is satisfiable (Step 8). Although Lipton did not report an experimental verification of his algorithm, the biological implementation would be straightforward, using affinity purification to implement extraction, pouring of tubes for *merge*, and running the solution through a gel to *detect*.

Soon after Lipton’s paper, the Proceedings of the Second Annual Workshop on DNA-Based Computers contained several papers describing significant developments in molecular computing. These included the parallel filtering model, the sticker model, DNA self-assembly, RNA computing, and surface-based computing.

We describe the last four developments (or variants thereof) in the next section. Here, we

briefly introduce the parallel filtering model, as it motivates some of the later discussion.

### Parallel Filtering Model

The first description of the parallel filtering model appeared in (Amos et al. 1996), with further analysis in (Amos 2005). This model was the first to provide a formal framework for the easy description of DNA algorithms for *any* problem in the complexity class *NP*. The main difference between the parallel filtering model and those previously proposed lies in the *implementation* of the removal of strings. All other models propose *separation* steps, where strings are *conserved* and may be used later in the computation. Within the parallel filtering model, however, strings that are removed are *discarded* and play no further part in the computation. This model is the first exemplar of the so-called “mark and destroy” paradigm of molecular computing and was motivated in part by discussion of the error-prone nature of biotin-based separation techniques. The new operation introduced within this model is:

- *Remove* ( $U, \{S_i\}$ ). This operation removes from the set  $U$ , in parallel, any string which contains at least one occurrence of any of the substrings  $S_i$ .

The proposed implementation of the *remove* operation is as follows: for a given substring  $S_i$ , add “tag” strands composed of its Watson-Crick *complement* to the tube  $U$ , so that the tags anneal only to strands containing the target subsequence. We then add polymerase to make tagged strands double stranded from the point of annealing (“mark”). We assume that working strands contain a specific restriction site embedded along them at fixed intervals, so adding the appropriate restriction enzyme removes only marked strands by repeated digestion (“destroy”).

### Subsequent Work

In this section we describe several other successful laboratory implementations of molecular-based computing, each illustrating a novel

approach or new technique. Our objective is not to give an exhaustive description of each experiment, but to give a high-level treatment of the general methodology, so that the reader may approach with confidence the fuller description in the literature.

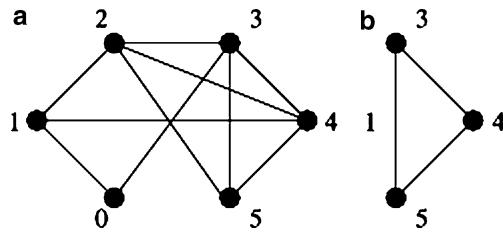
### DNA Addition

One of the first successful experiments reported after Adleman’s result was due to Guarnieri et al. in 1996 (Guarnieri et al. 1996), in which they describe a DNA-based algorithm for binary addition. The method uses single-stranded DNA reactions to add together two nonnegative binary numbers. This application, as the authors note, is very different from previous proposals, which use DNA as the substrate for a massively parallel random search.

Adding binary numbers requires keeping track of the position of each digit and of any “carries” that arise from adding 1 to 1 (remembering that  $1 + 1 = 0$  plus carry 1 in binary). The DNA sequences used represent not only binary strings but also allow for carries and the extension of DNA strands to represent answers. Guarnieri et al. use sequences that encode a digit in a given position and its significance or position from the right. For example, the first digit in the first position is represented by two DNA strands, each consisting of a short sequence representing a “position transfer operator,” a short sequence representing the digit’s value, and a short sequence representing a “position operator.”

DNA representations of all possible two-bit binary integers are constructed, which can then be added in pairs. Adding a pair involves adding appropriate complementary strands, which then link up and provide the basis for strand extension to make new, longer strands. This is termed a “horizontal chain reaction,” where input sequences serve as templates for constructing an extended result strand. The final strand serves as a record of successive operations, which is then read out to yield the answer digits in the correct order.

The results obtained confirmed the correct addition of  $0 + 0$ ,  $0 + 1$ ,  $1 + 0$ , and  $1 + 1$ , each calculation taking between 1 and 2 days of bench



**DNA Computing, Fig. 8** (a) Ouyang graph. (b) Example subgraph (From Amos 2005)

work. Although limited in scope, this experiment was (at the time) one of the few experimental implementations to support theoretical results.

### Maximal Clique Computation

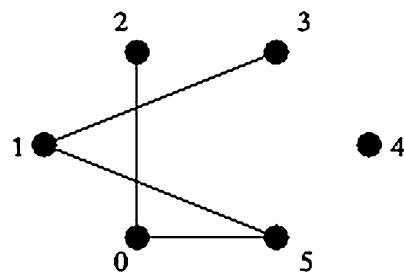
The problem of finding a maximal clique using DNA was addressed by Ouyang et al. in 1997 (Ouyang et al. 1997). A *clique* is a fully connected subgraph of a given graph (Fig. 8). The *maximal clique* problem asks, given a graph, how many vertices are there in the largest clique? Finding the size of the largest clique is an NP-complete problem.

The algorithm proceeds as follows: for a graph  $G$  with  $n$  vertices, all possible vertex subsets (subgraphs) are represented by an  $n$ -bit binary string  $b_{n-1}, b_{n-2}, \dots, b_0$ . For example, given the six-vertex graph used in (Ouyang et al. 1997) and depicted in Fig. 8a, the string 111000 corresponds to the subgraph depicted in Fig. 8b, containing  $v_5, v_4$ , and  $v_3$ .

Clearly, the largest clique in this graph contains  $v_5, v_4, v_3$ , and  $v_2$ , represented by the string 111100.

The next stage is to find pairs of vertices that are not connected by an edge (and, therefore, by definition, cannot appear together in a clique). We begin by taking the *complement* of  $G$ ,  $\bar{G}$ , which contains the same vertex set as  $G$ , but which only contains an edge  $\{v, w\}$  if  $\{v, w\}$  is *not* present in the edge set of  $G$ . The complement of the graph depicted in Fig. 8 is shown in Fig. 9.

If two vertices in  $\bar{G}$  are connected by an edge, then their corresponding bits cannot both be set to 1 in any given string. For the given problem, we must therefore remove strings encoding  $***1^*1$  ( $v_2$  and  $v_0$ ),  $1^{***}1$  ( $v_5$  and  $v_0$ ),  $1^{***}1^*$  ( $v_5$  and  $v_1$ ), and  $**1^*1^*$  ( $v_3$  and  $v_1$ ), where  $*$  means either 1 or 0. All other strings encode a (not necessarily maximal) clique.



**DNA Computing, Fig. 9** Complement of Ouyang graph (From Amos 2005)

We must then sort the remaining strings to find the largest clique. This is simply a case of finding the string containing the largest number of 1 s, as each one corresponds to a vertex in the clique. The string containing the largest number of 1 s encodes the largest clique in the graph.

The DNA implementation of the algorithm goes as follows. The first task is to construct a set of DNA strands to represent all possible subgraphs. There are two strands per bit, to represent either 1 or 0. Each strand associated with a bit  $i$  is represented by two sections, its position  $P_i$  and its value  $V_i$ . All  $P_i$  sections are of length 20 bases. If  $V_i = 1$ , then the sequence representing  $V_i$  is a restriction site unique to that strand. If  $V_i = 0$ , then the sequence representing  $V_i$  is a 10-base “filler” sequence. Therefore, the longest possible sequence is 200 bases, corresponding to the string 000000, and the shortest sequence is 140 bases, corresponding to 111111.

The computation then proceeds by digesting strands in the library, guided by the complementary graph  $\bar{G}$ . To remove strands encoding a connection  $i, j$  in  $\bar{G}$ , the current tube is divided into two,  $t_0$  and  $t_1$ . In  $t_0$  we cut strings encoding  $V_i = 1$  by adding the restriction enzyme associated with  $V_i$ . In  $t_1$  we cut strings encoding  $V_j = 1$  by adding the restriction enzyme associated with  $V_j$ . For example, to remove strands encoding the connection between  $V_0$  and  $V_2$ , we cut strings containing  $V_0 = 1$  in  $t_0$  with the enzyme Afl II and we cut strings containing  $V_2 = 1$  in  $t_1$  with Spe I. The two tubes are then combined into a new working tube, and the next edge in  $\bar{G}$  is dealt with.

In order to read the size of the largest clique, the final tube was simply run on a gel. The authors performed this operation and found the shortest

band to be 160 bp, corresponding to a 4-vertex clique. This DNA was then sequenced and found to represent the correct solution, 111100.

Although this is another good illustration of a DNA-based computation, the authors acknowledge the lack of scalability of their approach. One major factor is the requirement that each vertex be associated with an individual restriction enzyme. This, of course, limits the number of vertices that can be handled by the number of restriction enzymes available. However, a more fundamental issue is the exponential growth in the problem size (and thus the initial library), which we shall encounter again.

### Chess Games

In (Faulhammer et al. 2000), Faulhammer et al. describe a solution to a variant of the satisfiability problem that uses RNA rather than DNA as the computational substrate. They consider a variant of SAT, the so-called Knight problem, which seeks configurations of knights on an  $n \times n$  chess board, such that no knight is attacking any other (Watkins 2004).

The authors prefer the “mark and destroy” strategy rather than the repeated use of extraction to remove illegal solutions. However, the use of an RNA library and ribonuclease (RNase) H digestion gives greater flexibility, as one is not constrained by the set of restriction enzymes available. In this way, the RNase H acts as a “universal restriction enzyme,” allowing selective marking of virtually any RNA strands for parallel destruction by digestion.

The particular instance solved in (Faulhammer et al. 2000) used a  $3 \times 3$  board, with the variables  $a - i$  representing the squares. If a variable is set to 1, then a knight is present at that variable’s square, and 0 represents the absence of a knight. The  $3 \times 3$  knight problem may therefore be represented as the following instance of SAT:

$$\begin{aligned} & ((\neg h \wedge \neg f) \vee \neg a) \wedge ((\neg g \wedge \neg i) \vee \neg b) \\ & \wedge ((\neg d \wedge \neg h) \vee \neg c) \wedge ((\neg c \wedge \neg i) \vee \neg d) \\ & \wedge ((\neg a \wedge \neg g) \vee \neg f) \wedge ((\neg b \vee \neg f) \vee \neg g) \\ & \wedge ((\neg a \wedge \neg c) \vee \neg h) \wedge ((\neg d \wedge \neg b) \vee \neg i) \end{aligned}$$

which, in this case, simplifies to

$$\begin{aligned} & ((\neg h \wedge \neg f) \wedge \neg a) \wedge ((\neg g \wedge \neg i) \vee \neg b) \\ & \wedge ((\neg d \wedge \neg h) \vee \neg c) \wedge ((\neg c \wedge \neg i) \vee \neg d) \\ & \wedge ((\neg a \wedge \neg g) \vee \neg f). \end{aligned}$$

This simplification greatly reduces the number of laboratory steps required. The experiment proceeds by using a series of RNase H digestions of “illegal” board representations, along the lines of the parallel filtering model (Amos et al. 1996).

Board representations are encoded as follows: the experiment starts with all strings of the form  $x_1, \dots, x_n$ , where each variable  $x_i$  takes the value 1 or 0. Then, the following operations may be performed on the population of strings:

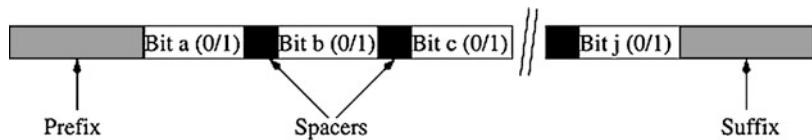
- Cut all strings containing any pattern of specified variables  $p_1, \dots, p_k$ .
- Separate the “test tube” into several collections of strings (molecules) by length.
- Equally divide (i.e., split) the contents of a tube into two tubes.
- Pour (mix) two test tubes together.
- Sample a random string from the test tube.

The first stage of the algorithm is the construction of the initial library of strands. Each strand sequence follows the template depicted in Fig. 10.

The prefix and suffix regions are included to facilitate PCR. Each variable is represented by one of two unique sequences of length 15 nucleotides, one representing the fact that the variable is set to one and the other the fact that it is set to 0. Variable regions are separated by short (five nucleotide) spacer regions. In order to avoid having to individually generate each individual sequence, a “mix and split” strategy (described in more detail in Faulhammer et al. 2000) is used. The RNA version of the library is then generated by *in vitro* transcription.

The algorithm proceeds as follows:

1. For each square, sequentially, split the RNA library into two tubes, labeled 1 and 2. After digestions have taken place, tube 1 will contain strands that contain a knight at that square, and tube 2 will contain strands that do *not* have knights at that square.

**DNA Computing,****Fig. 10** Template for RNA strands (From Amos 2005)

2. In tube 1, digest with RNase H strands that have no knight at position **a**, as well as strands that describe a knight at attacking positions **h** and **f**. This implements the logical statement  $((\neg h \wedge \neg f) \vee \neg a)$ .
3. In tube 2, digest strands that have a knight present at position **a**.
4. Remove the DNA oligos used to perform the above digestions.
5. Go to step 1, repeating with square **b**.

Steps 1 through 4 implement the following: “There may or may not be a knight in square **a** – if there *is*, then it is attacking squares **h** and **f**, so disallow this.” The algorithm only needs to be performed for squares **a**, **b**, **c**, **d**, and **f**, as square **e**, by the rules of chess, cannot threaten or be threatened on a board this size, and any illegal interactions that squares **g**, **h**, and **i** may have are with **a**, **b**, **c**, **d**, and **f** and have already been dealt with. At the conclusion of this stage, any remaining full-length strands are recovered, as they should encode legal boards.

The “mark and destroy” digestion operation is implemented as follows. If we wish to retain (i.e., select) strands encoding variable *a* to have value 1, DNA oligonucleotides corresponding to the complement of the *a* = 0 sequence are added to the tube and anneal to all strands encoding *a* = 0. RNase H is then added to the solution. Ribonuclease H (RNase H) is an endoribonuclease which specifically hydrolyzes the phosphodiester bonds of RNA hybridized to DNA. RNase H does not digest single- or double-stranded DNA, so his operation therefore leaves intact only those strands encoding *a* = 1, in a fashion similar to the removal operation of the parallel filtering model (Amos et al. 1996).

The results obtained (described in Faulhammer et al. 2000) were extremely encouraging: out of 43 output strands sampled, only one contained an illegal board. Given that the population sampled

encoded 127 knights, this gave an overall knight placement success rate of 97.7 %.

**Computing on Surfaces**

Another experiment that makes use of the “mark and destroy” paradigm is described in Liu et al. (2000) (although early work was performed from 1996). The key difference between this and previous experiments is that the DNA strands used are tethered to a support rather than being allowed to float freely in solution. The authors argue that this approach greatly simplifies the automation of the (potentially very many) repetitive chemical processes required during the performance of an experiment.

The authors report a DNA-based solution to a small instance of SAT. The specific problem solved is

$$(w \vee x \vee y) \wedge (w \vee \neg y \vee z) \wedge (\neg x \vee y) \\ \wedge (\neg w \vee \neg y).$$

Sixteen unique DNA strands were synthesized, each one corresponding to one of the  $2^4 = 16$  combinations of variable values. The actual encodings are given in Table 1 (taken from Liu et al. 2000).

Each of the 16 sets of strands was then affixed to a specific region of a gold-coated surface, so that each solution to the SAT problem was represented as an individual cluster of strands.

The algorithm then proceeds as follows. For each clause of the problem, a cycle of “mark,” “destroy,” and “unmark” operations is carried out. The goal of each cycle is to destroy the strands that do *not* satisfy the appropriate clause. Thus, in the first cycle, the objective is to destroy strands that do not satisfy the clause  $(w \vee x \vee y)$ . Destruction is achieved by “protecting” or *marking* strands that *do* satisfy the clause by annealing to them their complementary strands. *Escherichia*

**DNA Computing, Table 1** Strands used to represent SAT variable values

Strand	Sequence	wxyz
$S_0$	CAACCCAA	0000
$S_1$	TCTCAGAG	0001
$S_2$	GAAGGCAT	0010
$S_3$	AGGAATGC	0011
$S_4$	ATCGAGCT	0100
$S_5$	TTGGACCA	0101
$S_6$	ACCATTGG	0110
$S_7$	GTTGGGTT	0111
$S_8$	CCAAGTTG	1000
$S_9$	CAGTTGAC	1001
$S_{10}$	TGGTTTGG	1010
$S_{11}$	GATCCGAT	1011
$S_{12}$	ATATCGCG	1100
$S_{13}$	GGTTCAAC	1101
$S_{14}$	AACCTGGT	1110
$S_{15}$	ACTGGTCA	1111

*coli* exonuclease I is then used to digest unmarked strands (i.e., any single-stranded DNA).

By inspection of Table 1, we see that this applies to only two strands,  $S_0$  (0000) and  $S_1$  (0001). Thus, in cycle 1 the complements of the 14 other strands

$$\begin{aligned} w &= 1(S_8, S_9, S_{10}, S_{11}, S_{12}, S_{13}, S_{14}, S_{15}); \\ x &= 1(S_4, S_5, S_6, S_7, S_{12}, S_{13}, S_{14}, S_{15}); \\ y &= 1(S_2, S_3, S_6, S_7, S_{10}, S_{11}, S_{14}, S_{15}) \end{aligned}$$

were combined and hybridized to the surface before the exonuclease I was added. The surface was then regenerated (the unmark operation) to return the remaining surface-bound oligos ( $S_2 - S_{15}$ ) to single-stranded form. This process was repeated three more times for the remaining three clauses, leaving a surface containing only strands encoding a legal solution to the SAT problem.

The remaining molecules were amplified using PCR and then hybridized to an addressed array. The results of fluorescence imaging clearly showed four spots of relatively high intensity, corresponding to the four regions occupied by legal solutions to the problem ( $S_3$ ,  $S_7$ ,  $S_8$ , and  $S_9$ ).

Although these results are encouraging as a first move toward more error-resistant DNA computing, as the authors themselves acknowledge, there remain serious concerns about the scalability of this approach (1,536 individual oligos would be required for a 36-bit, as opposed to 4-bit, implementation).

### Computing with Hairpins

The tendency of DNA molecules to self-anneal was exploited by Sakamoto et al. in (Sakamoto et al. 2000) for the purposes of solving a small instance of SAT. The authors encode the given formula in “literal strings” which are conjunctions of the literals selected from each SAT clause (one literal per clause). A formula is satisfiable if there exists a literal string that does not contain any variable together with its negation. If each variable is encoded as a DNA subsequence that is the Watson-Crick complement of its negation, then any strands containing a variable and its negation self-anneal to form “hairpin” structures. These can be distinguished from non-hairpin structure-forming strands and removed. The benefit of this approach is that it does not require physical manipulation of the DNA, only temperature cycling. The drawback is that it requires  $3^m$  literal strings for  $m$  clauses, thus invoking once again the scalability argument.

### Gel-Based Computing

A much larger (20 variable) instance of 3-SAT was successfully solved by Adleman’s group in an experiment described in 2002 (Braich et al. 2002). This is, to date, the largest known problem instance successfully solved by a DNA-based computer; indeed, as the authors state, “this computational problem may yet be the largest yet solved by nonelectronic means” (Braich et al. 2002).

The architecture underlying the experiment is related to the sticker model described by Roweis et al. (1996). The difference here is that only separation steps are used – the application of stickers is not used. Separations are achieved by using oligo probes immobilized in polyacrylamide gel-filled glass modules, and strands are pulled through them by electrophoresis. Strands

are removed (i.e., retained in the module) by virtue of their hybridizing to the immobilized probes, with other strands free to pass through the module and be subject to further processing. Captured strands may be released and transported (again via electrophoresis) to other modules for further processing.

The potential benefits of such an approach are clear; the use of electrophoresis minimizes the number of laboratory operations performed on strands, which, in turn, increases the chance of success of an experiment. Since strands are not deliberately damaged in any way, they, together with the glass modules, are potentially reusable for multiple computations. Finally, the whole process is potentially automatable.

The problem solves a 20-variable, 24-clause 3-SAT formula  $\Phi$ , with a unique satisfying truth assignment. These are

$$\begin{aligned}\Phi = & (\neg x_{13} \vee x_{16} \vee x_{18}) \wedge (x_5 \vee x_{12} \vee \neg x_9) \\ & \wedge (\neg x_{13} \vee \neg x_2 \vee x_{20}) \wedge (x_{12} \vee x_9 \vee \neg x_5) \\ & \wedge (\neg x_{19} \vee \neg x_4 \vee x_6) \wedge (x_9 \vee x_{12} \vee \neg x_5) \\ & \wedge (\neg x_1 \vee x_4 \vee \neg x_{11}) \wedge (x_{13} \vee \neg x_2 \vee \neg x_{19}) \\ & \wedge (x_5 \vee x_{17} \vee x_9) \wedge (x_{15} \vee x_9 \vee \neg x_{17}) \\ & \wedge (\neg x_5 \vee \neg x_9 \vee \neg x_{12}) \wedge (x_6 \vee x_{11} \vee x_4) \\ & \wedge (\neg x_{15} \vee \neg x_{17} \vee x_7) \wedge (\neg x_6 \vee x_{19} \vee x_{13}) \\ & \wedge (\neg x_{12} \vee \neg x_9 \vee x_5) \wedge (x_{12} \vee x_1 \vee x_{14}) \\ & \wedge (x_{20} \vee x_3 \vee x_2) \wedge (x_{10} \vee \neg x_7 \vee \neg x_8) \\ & \wedge (\neg x_5 \vee x_9 \vee \neg x_{12}) \wedge (x_{18} \vee \neg x_{20} \vee x_3) \\ & \wedge (\neg x_{10} \vee \neg x_{18} \vee \neg x_{16}) \wedge (x_1 \vee \neg x_{11} \vee \neg x_{14}) \\ & \wedge (x_8 \vee \neg x_7 \vee \neg x_{15}) \wedge (\neg x_8 \vee x_{16} \vee \neg x_{10})\end{aligned}$$

with a unique satisfying assignment of

$$\begin{aligned}x_1 &= F, \quad x_2 = T, \quad x_3 = F, \quad x_4 = F, \\ x_5 &= F, x_6 = F, \quad x_7 = T, \quad x_8 = T, \\ x_9 &= F, \quad x_{10} = T, x_{11} = T, \\ x_{12} &= T, \quad x_{13} = F, \quad x_{14} = F, \\ x_{15} &= T, x_{16} = T, \quad x_{17} = T, \\ x_{18} &= F, \quad x_{19} = F, \quad x_{20} = F.\end{aligned}$$

As there are 20 variables, there are  $2^{20} = 1,048,576$  possible truth assignments. To represent all possible assignments, two distinct 15-base *value sequences* were assigned to each variable  $x_k$  ( $k = 1, \dots, 20$ ), one representing true ( $T$ ),  $X_k^T$ , and one representing false ( $F$ ),  $X_k^F$ . A mix

and split generation technique similar to that of Faulhammer et al. (2000) was used to generate a 300-base *library sequence* for each of the unique truth assignments. Each library sequence was made up of 20 value sequences joined together, representing the 20 different variables. These library sequences were then amplified with PCR.

The computation proceeds as follows: for each clause, a glass clause module is constructed which is filled with gel and contains covalently bound probes designed to capture only those library strands that *do satisfy* that clause; strands that do not satisfy the clause are discarded.

In the first clause module ( $\neg x_{13} \vee x_{16} \vee x_{18}$ ), strands encoding  $X_3^F$ ,  $X_{16}^F$ , and  $X_{18}^T$  are retained, while strands encoding  $X_3^T$ ,  $X_{16}^T$ , and  $X_{18}^F$  are discarded. Retained strands are then used as input to the next clause module, for each of the remaining clauses. The final (24th) clause module should contain only those strands that have been retained in all 24 clause modules and hence encode truth assignments satisfying  $\Phi$ .

The experimental results confirmed that a unique satisfying truth assignment for  $\Phi$  was indeed found using this method. Impressive though it is, the authors still regard with skepticism claims made for the potential superiority of DNA-based computers over their traditional silicon counterparts. In a separate interview, Len Adleman stated that “DNA computers are unlikely to become stand-alone competitors for electronic computers. We simply cannot, at this time, control molecules with the deftness that electrical engineers and physicists control electrons” (Regalado 2002). However, “they [DNA computers] enlighten us about alternatives to electronic computers and studying them may ultimately lead us to the true ‘computer of the future’” (Braich et al. 2002). In the next section, we consider how the focus of DNA computing has since shifted away from the solution of “traditional” problems.

## Assessment

In February 1995, one of the founding fathers of the theory of *computational complexity* (q.v.)

published a short paper on DNA computing. In his article, titled “On the Weight of Computations” (Hartmanis 1995), Juris Hartmanis sounded a cautionary note amidst the growing interest surrounding DNA computing. By calculating the smallest amount of DNA required to encode *all possible paths* through a graph, he calculated that Adleman’s experiment, if scaled up from 7 cities to 200 cities, would require an initial set of DNA strands that would weigh more than the Earth. Hartmanis was quick to point out the value of the initial work: “Adleman’s molecular solution of the Hamiltonian path problem is indeed a magnificent achievement and may initiate a more intensive exploration of molecular computing and computing in biological systems.” However, he also emphasized the long-term futility of hoping that DNA-based computers could ever beat silicon machines in this particular domain. Soon after Adleman’s experiment, hopes were expressed that a lot of the promise of molecular computers could be derived from their massive inherent parallelism – each operation is performed at the same time on trillions of DNA strands. However, as Turing showed, computers are not limited to any particular physical construction, and a DNA computer will suffer just as much as its silicon counterpart from the “exponential curse.” If we require a molecular algorithm for a difficult problem to run in reasonable *time*, then there is a fundamental requirement (unless P is shown to be equivalent to NP) for an exponential amount of *space* (in this case, the amount of DNA required). As Hartmanis observed, “... the exponential function grows too fast and the atoms are a bit too heavy to hope that the molecular computer can break the exponential barrier, this time the weight barrier.”

This view would now seem to largely reflect the community consensus. As Ogihara and Ray argued in 2000, “It is foolish to attempt to predict the future of technology, but it may be that the ideal application for DNA computation does not lie in computing large NP problems” (Ogihara and Ray 2000). In an interview in 2002, nanotechnologist Ned Seeman stated that “I am not a computer scientist, but I suspect it

[molecular computing] is not well suited to traditional problems. I certainly feel that it is better suited to algorithmic self-assembly and biologically-oriented applications” (Smalley 2005). That is not to say that molecular-based computations do not have a future, and in the final Section we briefly discuss some possible future directions that molecular computing may take (and offer pointers to other articles in the current volume).

## Future Directions

If molecular computations are to be scalable and/or sustainable, then it is clear that they should be performed with a minimum of human intervention. The type of experiment originally performed by Adleman was feasible because it required a relatively small number of biological steps (although it still took a week of bench time to carry out). However, the number of manipulations (and the amount of material) required for a non-trivial computation would quickly render this approach infeasible. Although attempts have been made to automate molecular computations using, for example, microfluidics (van Noort et al. 2002), this does not allow us to avoid the fundamental issue of scalability.

One promising subfield concerns molecular computing using machines (or “automata”) constructed from DNA strands and other biomaterials. The construction of *molecular automata* (q.v.) was demonstrated by Benenson et al. in (Benenson et al. 2003). This experiment builds on the authors’ earlier work (Benenson et al. 2001) on the construction of biomolecular machines. In (Benenson et al. 2003), the authors describe the construction of a molecular automaton that uses the process of DNA backbone hydrolysis and strand hybridization, fuelled by the potential free energy stored in the DNA itself.

One aim of researchers in this field is to build automata that may operate within living cells. Such devices may offer possibilities in terms of novel therapeutics, drug synthesis, bio-nanotechnology, or *amorphous computing* (q.v.)

Theoretical studies in DNA computing based on abstract models of the cell are already well established (see the article on *membrane computing*), but recent work on *bacterial computing* (q.v.) has illustrated the feasibility of engineering living cells for the purposes of human-defined computation.

Finally, the notion of *biomolecular self-assembly* (q.v.) suggests ways in which the tendency of molecules to form spontaneously ordered structures may be harnessed, either for the purposes of computation or for engineering-based applications (e.g., *DNA-templated self-assembly of proteins and nanowires* (q.v.)). *Algorithmic* self-assembly has been demonstrated in the laboratory by Mao et al. (2000). This builds on work done on the self-assembly of periodic two-dimensional arrays (or “sheets”) of DNA tiles connected by “sticky” pads (Winfree et al. 1998; Winfree 1998). The authors of (Mao et al. 2000) report a one-dimensional algorithmic self-assembly of DNA triple-crossover molecules (tiles) to execute four steps of a logical XOR operation on a string of binary bits.

Triple-crossover molecules contain four strands that self-assemble through Watson-Crick complementarity to produce three double helices in roughly a planar formation. Each double helix is connected to adjacent double helices at points where their strands cross over between them. The ends of the core helix are closed by hairpin loops, but the other helices may end in sticky ends which direct the assembly of the macrostructure. The tiles then self-assemble to perform a computation. The authors of (Mao et al. 2000) report successful XOR computations on pairs of bits, but note that the scalability of the approach relies on proper hairpin formation in very long single-stranded molecules, which cannot be assumed.

In 2006, Paul Rothemund produced a ground-breaking piece of work, which he called “DNA origami” (Rothemund 2006). His breakthrough was to describe a completely novel method for constructing *arbitrary* two-dimensional DNA-based structures. “When it comes to making shapes out of DNA,” explained Lloyd Smith in a commentary article (Smith 2006), “the material is there, and its properties are understood.

What was missing was a convincing, universal design scheme to allow our capabilities to unfold to the full.” Rothemund demonstrated the generality of his approach by showing how it could build structures as diverse as a triangle, a five-pointed star, and even a “smiley” face and a map of the Americas. The shapes that he managed to construct were ten times more complex than anything that had been constructed from DNA before. Rothemund’s scheme was different to previous tile-based methods in that it used only a *single* long DNA strand at its foundation. By repeatedly folding a long strand of DNA around in a maze-like pattern, a scaffold was formed that traced the outline of the desired object. This structure was then “clipped” into place by short DNA strands, which Rothemund referred to as “staples.” He used as the main building block a 7,000-base sequence of DNA taken from the M13 virus. He developed a computer program that would take a human-designed folding path (the “shape”), map it onto the sequence of the DNA strand, and then generate the sequences of the staples that would be required to hold it all together. In order to demonstrate the power of his method, he built a DNA map of the Americas, where a single nanometer (one billionth of a meter) represented 200 km in real life (the map was 75 nm across). “The results that emerge are stunning,” praised Lloyd Smith, after seeing the work. The possible impact of this work may well be huge, with one possible application being a “nanoworkbench.” In 2003, John Reif and his team had shown how a simple DNA scaffold could cause proteins to self-assemble into regular, periodic, two-dimensional grids (Yan et al. 2003). By using his approach to build much more complex holding structures, Rothemund argued that it could now be used to carry out biological experiments at the nanoscale, studying the behavior of complex protein assemblies, such as drugs, in a spatially controlled environment. As Lloyd Smith observes, “Thus equipped not only with DNA building materials and an understanding of their structural and chemical properties, but also with a versatile general approach to weaving them together, we are arriving at a new frontier in our pursuit of ever-smaller structures.

The barrier we have to surmount next is to deploy our knowledge to develop structures and devices that are really useful. Happily, in that endeavor we are now perhaps limited more by our imagination than our ability" (Smith 2006).

## Bibliography

### Primary Literature

- Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266:1021–1024
- Adleman LM (1995) On constructing a molecular computer. University of Southern California, Los Angeles, Draft
- Amos M (2005) Theoretical and experimental DNA computation. Springer, Berlin
- Amos M, Gibbons A, Hodgson D (1996) Error-resistant implementation of DNA computations. In: Landweber LF, Baum EB (eds) 2nd annual workshop on DNA based computers. Princeton University, NJ, 10-12 June 1996. American Mathematical Society, Providence
- Arkin A, Ross J (1994) Computational functions in biochemical reaction networks. *Biophys J* 67:560–578
- Benenson Y, Paz-Elizur T, Adar R, Keinan E, Livneh Z, Shapiro E (2001) Programmable and autonomous computing machine made of biomolecules. *Nature* 414:430–434
- Benenson Y, Adam R, Paz-Livneh T, Shapiro E (2003) DNA molecule provides a computing machine with both data and fuel. *Proc Natl Acad Sci USA* 100:2191–2196
- Bennett CH (1982) The thermodynamics of computation – a review. *Int J Theor Phys* 21:905–940
- Braich RS, Chelyapov N, Johnson C, Rothemund PWK, Adleman L (2002) Solution of a 20-variable 3-SAT problem on a DNA computer. *Science* 296:499–502
- Bray D (1995) Protein molecules as computational elements in living cells. *Nature* 376:307–312
- Breslauer KJ, Frank R, Blocker H, Marky LA (1986) Predicting DNA duplex stability from the base sequence. *Proc Natl Acad Sci USA* 83(11):3746–3750
- Brown TA (1993) Genetics: a molecular approach. Chapman and Hall, New York
- Campbell-Kelly M, Aspray W (2004) Computer: a history of the information machine, 2nd edn. Westview Press, Colorado
- Conrad M (1985) On design principles for a molecular computer. *Commun ACM* 28:464–480
- Conrad M, Liberman EA (1982) Molecular computing as a link between biological and physical theory. *J Theor Biol* 98:239–252
- Cook S (1971) The complexity of theorem proving procedures. Proceedings of the 3rd annual ACM symposium on theory of computing, pp 151–158
- Faulhammer D, Cukras AR, Lipton RJ, Landweber LF (2000) Molecular computation: RNA solutions to chess problems. *Proc Natl Acad Sci USA* 97:1385–1389
- Feynman RP (1961) There's plenty of room at the bottom. In: Gilbert D (ed) Miniaturization. Reinhold, New York, pp 282–296
- Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. WH Freeman and Company, New York
- Gibbons AM (1985) Algorithmic graph theory. Cambridge University Press, Cambridge
- Guarnieri F, Fliss M, Bancroft C (1996) Making DNA add. *Science* 273:220–223
- Hartmanis J (1995) On the weight of computations. *Bull Eur Assoc Theor Comput Sci* 55:136–138
- Hjelmfelt A, Weinberger ED, Ross J (1991) Chemical implementation of neural networks and turing machines. *Proc Natl Acad Sci USA* 88:10983–10987
- Hjelmfelt A, Schneider FW, Ross J (1993) Pattern recognition in coupled chemical kinetic systems. *Science* 260:335–337
- Lipton RJ (1995) DNA solution of hard computational problems. *Science* 268:542–545
- Liu Q, Wang L, Frutos AG, Condon AE, Corn RM, Smith LM (2000) DNA computing on surfaces. *Nature* 403:175–179
- Mao C, LaBean TH, Reif JH, Seeman NC (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature* 407:493–496
- Mullis KB, Ferré F, Gibbs RA (eds) (1994) The polymerase chain reaction. Birkhäuser, Boston
- Ogihara M, Ray A (2000) DNA computing on a chip. *Nature* 403:143–144
- Ouyang Q, Kaplan PD, Liu S, Libchaber A (1997) DNA solution of the maximal clique problem. *Science* 278:446–449
- Regalado A (2002) DNA computing. MIT Technology Review. <http://www.technologyreview.com/articles/00/05/regalado0500.asp>. Accessed 26 May 2008
- Rivest R, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public key cryptosystems. *Comm ACM* 21:120–126
- Rothemund PWK (2006) Folding DNA to create nanoscale patterns. *Nature* 440:297–302
- Roweis S, Winfree E, Burgoyne R, Chelyapov NV, Goodman MF, Rothemund PWK, Adleman LM (1996) A sticker based architecture for DNA computation. In: Landweber LF, Baum EB (eds) 2nd annual workshop on DNA based computers. Princeton University, NJ, 10-12 June 1996. American Mathematical Society, Providence
- Sakamoto K, Gouzu H, Komiya K, Kiga D, Yokoyama S, Yokomori T, Hagiya M (2000) Molecular computation by DNA hairpin formation. *Science* 288:1223–1226
- Smalley E (2005) Interview with Ned Seeman. *Technology Research News*, May 4
- Smith LM (2006) Nanostructures: the manifold faces of DNA. *Nature* 440:283–284
- Stubbe H (1972) History of genetics – from prehistoric times to the rediscovery of Mendel's laws. MIT Press, Cambridge

- van Noort D, Gast F-U, McCaskill JS (2002) DNA computing in microreactors. In: Jonoska N, Seeman NC (eds) DNA computing: 7th international workshop on DNA-based computers, vol 2340, LNCS. Springer, Berlin, pp 33–45
- Watkins JJ (2004) Across the board: the mathematics of chess problems. Princeton University Press, Princeton
- Watson JD, Crick FHC (1953a) Genetical implications of the structure of deoxyribose nucleic acid. *Nature* 171:964
- Watson JD, Crick FHC (1953b) Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature* 171:737–738
- Watson JD, Hopkins NH, Roberts JW, Steitz JA, Weiner AM (1987) Molecular biology of the gene, 4th edn. Benjamin/Cummings, Menlo Park
- Winfree E (1998) Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology
- Winfree E, Liu F, Wenzler L, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. *Nature* 394:539–544
- Yan H, Park SH, Finkelstein G, Reif JH, LaBean TH (2003) DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science* 301:1882–1884
- ### Books and Reviews
- Adleman L (1998) Computing with DNA. *Sci Am* 279:54–61
- Amos M (2006) Genesis machines: the new science of biocomputing. Atlantic Books, London
- Forbes N (2004) Imitation of life: how biology is inspiring computing. MIT Press, Cambridge
- Gonick L, Wheelis M (1983) The cartoon guide to genetics. Harper Perennial, New York
- Jones R (2004) Soft machines: nanotechnology and life. Oxford University Press, Oxford
- Păun G, Rozenberg G, Salomaa A (1998) DNA computing: new computing paradigms. Springer, Berlin
- Pool R (1995) A boom in plans for DNA computing. *Science* 268:498–499
- Watson J (2004) DNA: the secret of life. Arrow Books, London



## Molecular Automata

Joanne Macdonald<sup>1,3</sup>, Darko Stefanovic<sup>2</sup> and Milan Stojanovic<sup>1</sup>

<sup>1</sup>Division of Experimental Therapeutics,  
Department of Medicine, Columbia University,  
NY, USA

<sup>2</sup>Department of Computer Science and Center for  
Biomedical Engineering, University of New  
Mexico, Albuquerque, NM, USA

<sup>3</sup>Genecology Research Centre, Inflammation and  
Healing Research Cluster, School of Science and  
Engineering, University of the Sunshine Coast,  
Queensland, Australia

### Article Outline

Glossary  
Definition  
Introduction  
Molecular Automata as Language Recognizers  
Molecular Automata as Transducers and  
Controllers  
Future Directions  
Bibliography

### Keywords

Algorithmic self-assembly of DNA tiles;  
Deoxyribozyme; Deoxyribozyme-based automaton;  
Deoxyribozyme-based logic gate;  
DNA binary counter; DNA circuit; DNA computing;  
DNA logic gate; DNA sierpinski triangle;  
Ligase; Molecular automaton; Modular design of nucleic acid catalysts; Molecular finite-state automaton; Molecular logic gate;  
Molecular mealy automaton; Oligonucleotide;  
Phosphodiesterase

### Glossary

**Algorithmic self-assembly of DNA tiles** Spontaneous assembly of structures

consisting of interlocking semirigid DNA molecules (the tiles). Different tiles, containing different exposed ends, can be synthesized such that their affinity to interlock can be predefined. Particular sets of tiles may be constructed that result in the self-assembly of unique structures; each such set represents an algorithm for the assembly of the structure.

**DNA Sierpinski triangle** An algorithmically self-assembled DNA structure with a pattern approximating the Sierpinski gasket, a fractal.

**DNA computing** Generally, any use of the information-carrying capacity of DNA to achieve some computational or decision-making goal. The term is also used to refer specifically to the use of DNA in the manner pioneered by Adleman as a massively parallel computation substrate to solve certain combinatorial optimization problems.

**DNA circuit** A system consisting of multiple DNA logic gates in which DNA is used as the signal carrier between the gates; the system performs a Boolean logic function that may be more complex than the functions achievable using single logic gates.

**DNA binary counter** A device that uses DNA as a computational substrate to maintain its state, an integer binary numeral, and advances through successive states which represent successive integers. An algorithmic self-assembly implementation exists.

**DNA logic gate** A molecular device using DNA as a computational substrate to perform a Boolean logic function.

**Deoxyribozyme** An oligonucleotide synthesized such that its structure gives rise to enzymatic activity that can affect other oligonucleotides.

**Deoxyribozyme-based automaton** A molecular automaton which uses deoxyribozyme-based logic gates to achieve its function.

**Deoxyribozyme-based logic gate** An implementation of a DNA logic gate in which enzymatic activity of a deoxyribozyme is controlled by the presence or absence of activating or inhibiting oligonucleotides.

**Ligase** An enzyme, in particular a deoxyribozyme, that promotes the linking of two oligonucleotides into a single longer oligonucleotide.

**Modular design of nucleic acid catalysts** A method for the design of nucleic acid catalysts, in particular deoxyribozymes, wherein their nucleotide sequences are chosen by stringing together motifs (such as recognition regions, stems, and loops) which have been established to perform a desired function. With care, motifs will not interfere and will achieve a combined complex function.

**Molecular automaton** A device at the molecular scale which performs a predefined function; this function is seen at the macroscopic level as sampling the environment for molecular stimuli and providing

**Molecular finite-state automaton** A device that performs a language recognition function using molecules as a computational substrate; the language symbols, the states, and the transitions between states are realized as molecules.

**Molecular Mealy automaton** A device using molecules as a computational substrate to achieve a general-purpose sequential logic function, viz., a finite-state transducer.

**Molecular logic gate** A device using molecules as a computational substrate to perform a Boolean logic function.

**Oligonucleotide** A single-stranded DNA molecule consisting of a relatively small number of nucleotides, usually no more than a few dozen.

**Phosphodiesterase** An enzyme, in particular a deoxyribozyme, that promotes the cleavage of an oligonucleotide into two shorter oligonucleotides by catalyzing the hydrolysis a phosphodiester bond.

## Definition

Automata are devices that autonomously perform predetermined, often complex, functions. Historically, the term referred to mechanical devices that imitated human behaviors, especially those that exhibited intelligence. In the twentieth century, electronics made possible the development of various complex automata, the prime example being the computer. For its part, computer software was

organized as a hierarchy of automata operating within the computer automaton. A rich formal theory of automata arose in the fields of electronics and computer science to aid in the design and use of these now ubiquitous devices.

Molecular automata employ molecular-scale phenomena of binding, dissociation, and catalytic action to achieve predetermined functions. Different chemical implementations are possible, including proteins and inorganic reactions, but the principal prototypes today use nucleic acid chemistry, and this entry will focus on this modality. Nucleic acid automata may be able to interact directly with signaling molecular processes in living tissues. Thus, they provide a path toward autonomous diagnostic and therapeutic devices and even toward engineered control of cell behavior.

## Introduction

Humans appear to be naturally fascinated with the construction of devices with parts that move independently, contraptions that operate autonomously, and in particular, machines with an appearance of intelligence. Each technological advance throughout the ages has coincided with the production of more and more sophisticated devices; indeed, some argue that our strong innate urge toward a mechanistic philosophy has led to the development of automata (de Solla Price 1964).

Using tools at hand, ancient cultures created a variety of automata, from the animated figures of Rhodes to the singing figures of China. As with many human endeavors, the eighteenth century was the era of splendor and variety in automata, particularly among the French, and sophistication continued through the nineteenth: from courtly displays (mechanical) to digesting ducks (Riskin 2003; biomechanical) to painting devices (mechanical but emulating creative intelligence; Bailly 2003; Peppé 2002).

Modern automata are almost uniquely defined by the advent of computers. Just as electronics permitted building more complex systems than mechanics, software on programmable computers permitted vastly more complex systems. An entire discipline of automata theory developed to characterize the computational abilities and resource

requirements of different devices; all of theoretical computer science is an outgrowth of this effort.

The formal notion of an automaton centers on a description of the configuration, or state, of the device and of the transitions it makes from state to state. The states are typically discrete. The transitions are discrete as well and they occur in response to external stimuli formalized as the consumption of discrete input symbols. Given a current state and a current input symbol, the transition rules of an automaton define what state the automaton may enter next. A history of external stimuli is captured by an input symbol string. The automaton may produce output symbols. This has been formalized in two different but equally expressive ways: either an output symbol is associated with a specific state or it is associated with a specific transition between two states. In either case, the successive output symbols form an output string.

Automata defined in this fashion are closely linked with formal language theory. The input and the output symbols are drawn from certain alphabets, and the input and the output strings belong to languages over these alphabets. Of particular interest are automata with a restricted form of output: a *yes* or a *no* output is associated with each state. Such automata can be viewed as language recognizers under the following interpretation: if the state in which the automaton finds itself upon consuming an input string is a *yes* output state, that string is said to belong to the language, and otherwise not. More generally, an automaton defines a correspondence between strings in the input alphabet and strings in the output alphabet; thus, it can be understood as a language-translating device, or transducer. Language recognizers and transducers play a central role in traditional accounts of computational complexity, as well as in practical daily use, such as in web query processing, under the name of parsers. If the output alphabet consists of signals for mechanical actions, we have the essence of robotic control.

The number of symbols in an alphabet is generally taken to be finite. On the other hand, the number of states of an automaton might be finite or infinite. Finite-state automata are readily implemented in electronics using a register (to hold an encoding of the state) and a combinational circuit (to compute the transitions). Indeed,

only finite-state automata can be physically implemented. Nevertheless, a beautiful theory exists for classes of automata with infinite state; this theory has led to useful programming paradigms, and such automata can be simulated using finite resources as long as we are willing to accept that they may occasionally run out of these resources and fail. Examples include pushdown automata, in which the state includes the contents of a semi-infinite pushdown stack of symbols, and the Turing machine, in which the state includes the contents of an infinite tape of symbols.

Even as electronic embodiments of automata permeate the modern society, from mobile telephones to automated bank tellers to navigational satellites, they are not readily integrated into life processes: the environment of living tissues presents an obstacle for the deployment, powering, and operation of electronics. Biocompatible alternatives are needed.

With recent advances in molecular biology, research is turning toward the taming of molecules for the development of automata on the molecular scale. Within this body of research, several strands can be recognized. Firstly, researchers have adapted molecular processes for explicit emulation of mathematical theory. Secondly, researchers have sought to enhance known cellular systems for alternative control of biological events. Additionally, however, researchers are also beginning to engineer molecules for completely novel purposes unrelated to their original design.

Regardless of the purpose, all of these automata fundamentally require similar mechanisms to operate autonomously within their respective environments. Molecular automata must have some ability to sense inputs (molecular or other) within their external environment, make a pre-determined decision based on the sensed input, and then autonomously produce an output that affects their environment.

Here, we review recent advances in the engineering of molecular automata and outline fundamental principles for engineering of molecular automata. Section “[Molecular Automata as Language Recognizers](#)” treats molecular automata created as embodiments of the mathematical concept of a language recognizer. Minimal formal background is given, and recent research exemplars are introduced, along with sufficient

background in nucleic acid chemistry. We only treat exemplars that have been demonstrated in the laboratory. Section “[Molecular Automata as Language Recognizers](#)” treats molecular automata that work as transducers and is similarly organized. In section “[Future Directions](#),” we speculate on the future uses and developments of the molecular automata technology in science and medicine.

## Molecular Automata as Language Recognizers

### Preliminaries

#### Finite-State Automata

A finite-state automaton (sometimes just “finite automaton”) is a notional device that reads strings of symbols drawn from a finite alphabet and either accepts them or rejects them. Languages are sets of strings, and the language recognized by the automaton is defined to be the set of all strings which the automaton accepts.

We now consider how the automaton makes its decision. The automaton operates in a series of discrete steps, and each step consumes one symbol of the input string. The automaton exists in one of finitely many abstract states, and each step of operation causes a transition between states. A transition function describes which next states the automaton may enter upon consuming a particular symbol in a particular state. One state is designated as the start state; this is the state of the automaton before any input is read. Upon starting, the automaton reads the symbols from the input string and makes the indicated transitions until the string has been exhausted. If the final state in which it finds itself belongs to the designated subset of accepting states, we say that it has accepted the input string.

Formally, an alphabet  $\Sigma$  is a finite set of uninterpreted symbols. In examples, the sets  $\{0,1\}$  and  $\{a,b,c,\dots\}$  are common; in computing, ASCII and Unicode are used. While it may seem plausible that the nucleotides  $\{A,T,G,C\}$  should be used directly as the alphabet in molecular implementations of automata, device design, such as the use of restriction enzymes with

multi-nucleotide recognition sites, often dictates a less-dense representation; for instance, each of a few unique short oligonucleotides may stand for a symbol of the alphabet.

A string, or word, is a finite ordered sequence of symbols. A string of no symbols is written  $\varepsilon$ . Two strings  $s_1$  and  $s_2$  can be concatenated to form  $s_1s_2$ . Any set of strings is called a language. The notion of concatenation is extended to languages,  $L_1L_2 = \{s_1s_2 \mid s_1 \in L_1 \wedge s_2 \in L_2\}$ . A special operator, the Kleene star, finitely iterates concatenation:  $L^* = \varepsilon \cup L \cup LL \cup LLL \cup \dots$ . If we identify single-symbol strings with symbols, the set of all strings over an alphabet  $\Sigma$  is  $\Sigma^*$ .

Given a finite set of states  $Q$ , the transition function of an automaton is  $\delta: Q \times \Sigma \rightarrow Q$ . The start state is  $S \in Q$ , and the accepting (sometimes called final) states are  $F \subseteq Q$ . To formalize the operation of the automaton, we define the configuration of the automaton to be its current state paired with the unread input. The initial configuration for input  $w$  is  $(S, w)$ . The relation  $\vdash$  describes one step of operation:  $(s, x) \vdash (t, y)$  if  $(\exists a \in \Sigma)x = ay \wedge \delta(s, a) = t$ . The relation  $\vdash^*$  is the reflexive, transitive closure of  $\vdash$  and describes the complete operation. If  $(S, w) \vdash^* (s, \varepsilon)$ , then the automaton accepts  $w$  if and only if  $s \in F$ .

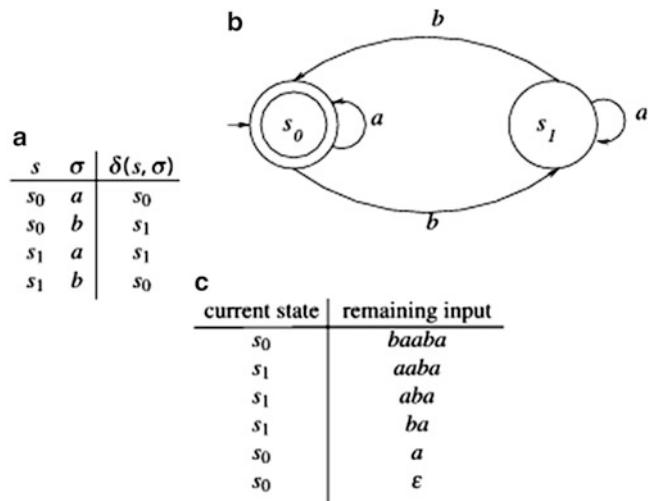
Finite-state automata as described are properly called deterministic. Nondeterministic finite-state automata are an apparent generalization that permits transitions that read strings rather than single symbols from the input at each step of operation, which is described by a transition relation  $\delta \subseteq Q \times \Sigma^* \times Q$ . With a nondeterministic automaton, we say it has accepted the input string if there is an accepting state among all states it may have reached while consuming the entire input string.

In the conventional visual representation of a finite-state automaton, the transition diagram, (example shown in Fig. 1), the states are shown as vertices in a graph and the transitions as directed edges labeled with symbols (or with strings for nondeterministic automata).

The class of languages that can be recognized by a finite-state automaton is known as regular languages; the term itself derives from an alternative description of the same class using regular grammars (see below). Nondeterministic automata

**Molecular Automata,**

**Fig. 1** Example finite-state automaton. The alphabet  $\{a,b\}$ , states  $\{s_0,s_1\}$ , start state  $s_0$ , accepting states  $\{s_0\}$ , and transition function given by the table in (a) define a deterministic finite-state automaton, graphically shown in (b). This automaton accepts precisely the strings that contain an even number of  $b$ 's. In (c), the actions of the automaton in accepting the string  $baaba$  are shown



recognize the same class of languages but may require fewer states for the same language.

The fact that nondeterministic automata are no more powerful than deterministic ones means that a number of variations that seem to be “in between” can be freely used. For instance, a deterministic automaton with a partial transition function  $\delta$ , i.e., in which the transition out of some particular state on some particular input symbol is not defined is really a nondeterministic automaton. Rather than having an explicit “stuck” state with a self-loop for all input symbols, as it would be required in a deterministic description, the automaton is stuck by virtue of not having any way out of a state. This is used in Benenson’s automata description (section “[Benenson’s Finite Automaton](#)”).

Finding functioning encodings for the symbols, states, and transitions in a molecular implementation is a challenging task, and therefore prototypes have dealt with small numbers of symbols and states, and it is of interest to know how many different automata can be encoded in a particular prototype. Considering deterministic automata alone, given that the transition functions  $\delta$  map  $Q \times \Sigma$  to  $Q$ , the number of possible such functions is  $|Q|^{|Q||\Sigma|}$ ; there are  $|Q|$  choices for the initial state, and there are  $2^{|Q|}$  choices for accepting states (any subset of the set of states). Thus, there are  $|Q|2^{|Q|}|Q|^{|Q||\Sigma|}$  different automata descriptions. Of course, many of these describe automata that are identical up to a relabeling of the

states; moreover, automata may be different and yet recognize the same language, so these calculations should not be taken too seriously.

**More Powerful Automata**

Pushdown automata are a natural extension of finite-state automata; these devices include a notionally infinite pushdown stack. This structure allows the device to write unbounded number symbols to its scratch memory, but at any time, only the most recently written symbol can be read back (which also erases it). These devices are capable of recognizing a larger class of languages, called the context-free languages.

Turing machines add two pushdown stacks to a finite-state automaton or, equivalently, an infinite read/write tape. These devices are capable of recognizing any language. All plausible models of what can be computed that have been put forward have been shown equivalent to Turing machines; we say that they describe universal computation.

**Wang Tiles and Self-Assembly**

One form of universal computation has been designed from a specific set of *Wang tiles* (Wang 1963). These are a mathematical model wherein square unit tiles are labeled with specific “glue” symbols on each edge. Each tile is only allowed to associate with tiles that have matching symbols, and the tiles cannot be rotated or reflected. The computational interest of Wang tiles is the

question of proving whether they can be used to tile a plane. Any Turing machine can be translated into a set of Wang tiles; the Wang tiles can tile the plane if and only if the Turing machine will never halt.

## Prototypes

### Molecular Automata as Language Recognizers

The encoding of finite automata states and transitions using DNA and restriction enzymes was first proposed by Rothemund (1996). A detailed design was also offered by Garzon et al. (1998). Shapiro's group extended this idea to an autonomously operating cascade of cleavages of a double-stranded DNA directed by oligonucleotides (Adar et al. 2004; Benenson et al. 2003, 2001). In effect, this group demonstrated the first molecular equivalent of a finite automaton. What is now sometimes called the Shapiro-Benenson-Rothemund automaton consists of a mixture of two groups of DNA molecules (input and "software," i.e., transition rules) and the *FokI* restriction enzyme ("hardware"). The automaton from (Benenson et al. 2003; Fig. 3) is described in section "Benenson's Finite Automaton."

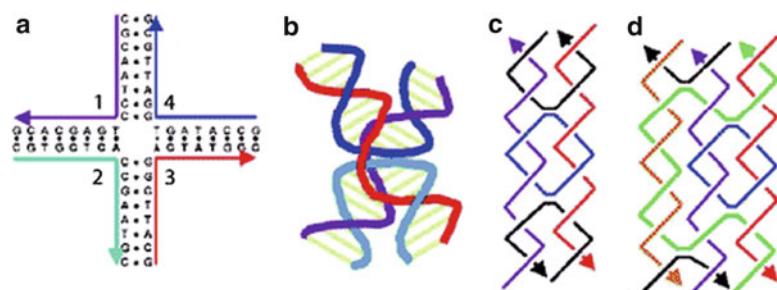
DNA has also been applied in the construction of Wang tiles and general self-assembly algorithms. Erik Winfree was the first to note that planar self-assembly of DNA molecules can be applied to this form of universal computation (Winfree 1996). DNA molecules analogous to Wang tiles can be constructed from double-crossover (DX) molecules (Fu and Seeman 1993), which consist of two side-by-side DNA duplexes that are joined by two crossovers (Fig. 2). They have a rigid stable body and open

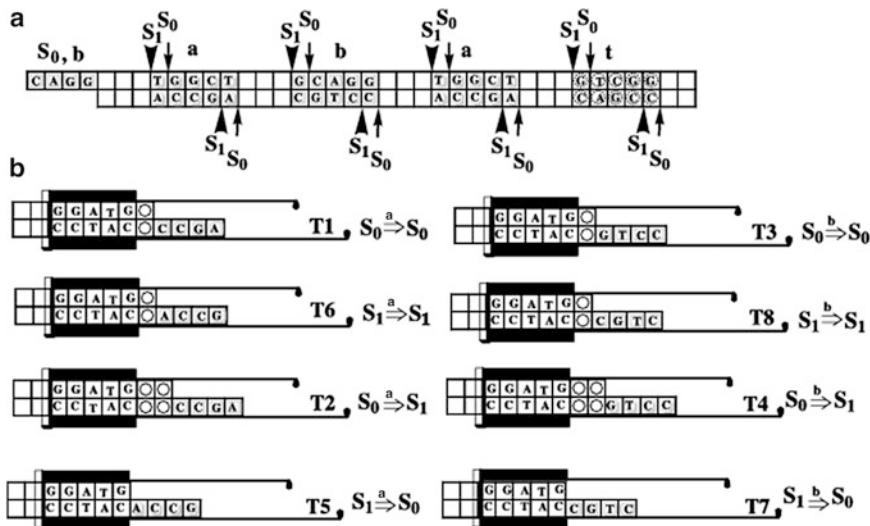
sticky ends for attachment to other DX molecules. The sticky ends of these DNA tiles may be labeled with certain sequences, analogous to the symbols labeling the sides of Wang tiles. This labeling allows the sticky ends to bind only to the tile ends that have a complementary sequence of base pairs, which corresponds to the rule that restricts Wang tiles to only associate with tiles that have matching symbols. Triple-crossover (TX) molecules consisting of three side-by-side DNA duplexes are also good for this purpose (LaBean et al. 2000). It was shown that universal computation could be accomplished by self-assembling these DNA tiles (Winfree et al. 1999). An appealing interpretation is that a two-dimensional self-assembling structure can be used to represent (in one dimension) the time development of a one-dimensional (in the other dimension) cellular automaton. The first example of computing performed by DNA self-assembly (Mao et al. 2000), a four-bit cumulative XOR, is described in section "DNA Self-Assembly." This is followed in section "Algorithmic DNA Self-Assembly" by a detailed description of algorithmic self-assembly of DNA tiles (Barish et al. 2005; Rothemund et al. 2004; Rothemund and Winfree 2000) used in the construction of a binary counter and a Sierpinski triangle.

### Benenson's Finite Automaton

The automaton described by Benenson encodes finite automata states and transitions using DNA and restriction enzymes. The input (I) consists of repetitive sequences, with groups of five base pairs (separated by three base pairs in; Benenson et al. 2003) which denote input symbols (Fig. 3a), and, upon enzymatic cleavage, the state of the

**Molecular Automata, Fig. 2** (a) A four-arm junction and (b) its three-dimensional structure; (c) a DNA DX; and (d) a DNA TX (From Mao 2004)





**Molecular Automata, Fig. 3** (a) The input molecule ( $I$ ) is a double-stranded DNA with an overhang and repeating motifs encoding symbols  $a$  (TGGCT),  $b$  (GCAGG), and, finally, a terminator  $t$  (GTCGG). The same sequences define two states ( $S_0$  and  $S_1$ ) of the finite automaton, depending on where *FokI* cleaves. Thus,  $S_0$  in  $a$  is defined as a GGC overhang and in  $b$  it is defined as a CAGG overhang, while  $S_1$  is defined in  $a$  as a TGGC overhang and in  $b$  as a GCAG overhang. States are similarly defined in  $t$ .

automaton as well. Two symbols in the input are  $a$  (TGGCT) and  $b$  (GCAGG). The cleavage that leaves the first four bases in an overhang is defined as a transition to an  $S_0$  state, while the cleavage that leaves the second four bases in an overhang is defined as a transition to an  $S_1$  state.

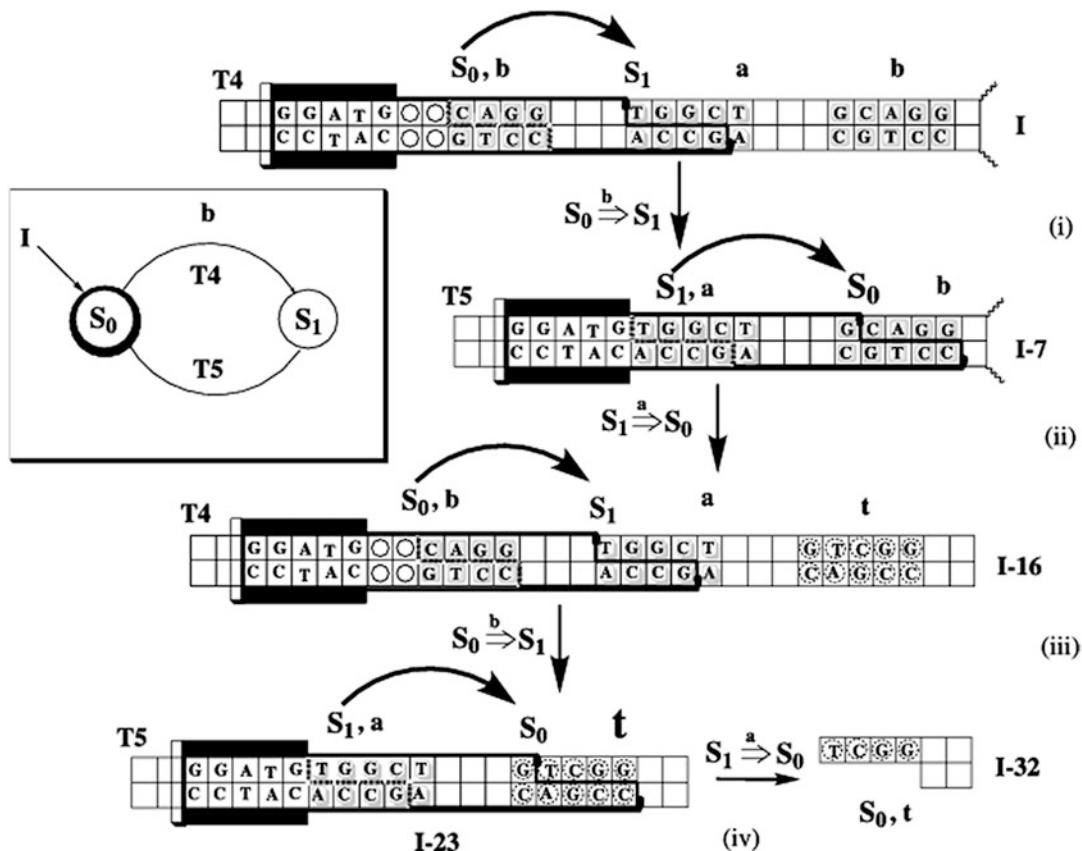
Interactions of an input with transition rules occur as follows: The readable input starts with an overhang of four bases, and this overhang is recognized by a complementary overhang on a transition rule (software). All transition rules are complexed with *FokI* (this is called the software-hardware complex). *FokI* recognizes the “guide” sequence in transition rules (GGATG) and then, upon interactions with input, cleaves the input at the constant position, nicking the DNA helix 9 and 13 positions away from the guide sequences (Fig. 4). In this way, the transition rule directs the cleavage of an input. As this process generates a new overhang, within the next symbol, we say that cleavage moves the automaton to read the next symbol and that at the same time it defines

(b) All possible transition rules in complex with *FokI* (software-hardware complex) for a two-state, two-symbol automaton. Spacer sequences (squares with circles) between guide sequence (GGATG) and targeting overhangs define the change of states, because they serve to adjust the cleavage position within the next symbol. Overhangs define the initial state and symbol that is read by the transition rules

the new state of the automaton. The structure of the transition rules defines whether a state of the automaton changes or not, by a distance between an overhang and the guide sequence.

This process is a cascade, because a new overhang can be recognized by another (or the same) transition rule. With two states and two symbols, we can have a total of eight transition rules (Fig. 3b), defining all possible transitions, upon reading a symbol (Keinan and colleagues also reported three-state, three-symbol automata; Soreni et al. 2005).

We say that this automaton (or in biochemistry a cascade) is capable of consuming input strings over a two-letter alphabet  $\{a,b\}$  and transitioning between states depending on the input and in accordance with a preprogrammed control, defined through the presence of an arbitrary mixture of transition rule complexes (T1–T8). The mechanism of this cascade is best explained through an example, and we provide one in Fig. 4. For example, the transition function



**Molecular Automata, Fig. 4** The effect of two transition rules, T4 and T5, on an input I from Fig. 3a is shown. The overhang at I (CAGG) defines an initial state ( $S_0$ ) and a current input symbol,  $b$ . The input encodes the string  $baba$ . For clarity, only bases in overhangs, symbols, and the guide sequence are shown. (i) Complexation between T4 and I directs the cleavage of I to I-7: Overhang CAGG on I is complementary to an overhang GTCC on the T4 complex. Thus, the automaton is at the beginning of its “calculation” in the state  $S_0$  and will “read” the next symbol  $b$  on the input. T4 complexes with the input and cleaves at the position 9/13 bases away within the region defining the symbol  $a$ . Upon this cleavage, the whole complex disintegrates, leaving the shortened input (I-7) with a new overhang in place, TGGC, representing the next state  $S_1$  and the current symbol  $a$ . Thus,

consisting of only transition rules T4 and T5 can move the automaton back and forth between states  $S_0$  and  $S_1$  endlessly, as long as it is moving along an input made of symbols  $a$  and  $b$  in alternation,  $baba\cdots$ , with the final result  $S_0$ . We invite the reader to try any other combination of transition rules in Fig. 3b on the same input. Some of them will stall the automaton.

the automaton performed according to a transition rule T4 an elementary step reading the first input symbol,  $b$ , and transitioning from state  $S_0$  into state  $S_1$ , moving along the input (tape) to the second symbol,  $a$ . (ii) The T5 complex recognizes overhang at the I-7 and the input again at the 9/13 position, which is now within the next symbol,  $b$ ; this leaves the new CAGG ( $S_0$ ) overhang. Thus, the automaton transitions from state  $S_1$  into state  $S_0$ , having read the input symbol  $a$ . It also moved to the following input symbol  $b$ , on the (ever shrinking) input, now at I-16. (iii) The automaton (actually, the T4 transition rule) recognizes and cleaves I-16 and moves to the next symbol  $a$ , transitioning in this process to  $S_1$  state and producing I-23. (iv) In the last step, I-23 is cleaved by T5 to I-32, producing the state  $S_0$  in the terminator symbol

Thus, through selecting a set of transition rules, or software/hardware complexes (transition rules of the form: if you find the system in state  $S_m$  and read a symbol  $\sigma \in \{a,b\}$  from the input, go to state  $S_n$  and move to the next input symbol), one could write a molecular “program” able to read and degrade an input DNA molecule, which encodes a defined set of symbols. Input molecules for

which transitions always succeed will be degraded up to a terminator symbol, which will read the final state of an automaton (“answer”). If there is no transition rule present in the system that matches the current state and the current symbol of the input at a certain stage of the degradation, the automaton stalls and the input molecule is not degraded to a terminator symbol. Thus, it can be said that this system indeed works as a finite-state automaton and thus recognizes the language abstractly specified by its set of transition rules.

While the laboratory demonstration was of finite-state automata, one could envisage push-down automata or Turing machines; indeed Shapiro has a patent on a molecular Turing machine (Shapiro and Karunaratne 2001).

#### DNA Self-Assembly

The first DNA self-assembling automaton was a four-bit cumulative XOR (Mao et al. 2000). The Boolean function XOR evaluates to 0 if its two inputs are equal, otherwise to 1. The “cumulative” (i.e., multi-argument) XOR takes Boolean input bits  $x_1, \dots, x_n$  and computes the Boolean outputs  $y_1, \dots, y_n$ , where  $y_1 = x_1$ , and for  $i > 1$ ,  $y_i = y_{i-1} \text{XOR } x_i$ . The effect of this is that  $y_i$  is equal to the even or odd parity of the first  $i$  values of  $x$ . Eight types of TX molecules were used in the implementation: two corner tiles, two input tiles, and four output tiles. The corner tiles connect a layer of input tiles to a layer of output tiles. The input tiles represent  $x_i = 0$  and  $x_i = 1$ . There are two ways to get each of the two possible outputs of a bitwise XOR, and so four output tiles were used, leaving some interpretation to the human observer. One output tile represents the state with output bit  $y_i = 0$  and input bits  $x_i = 0$  and  $y_{i-1} = 0$ ; another, the state with output bit  $y_i = 0$  and input bits  $x_i = 1$  and  $y_{i-1} = 1$ . The remaining output tiles represent the two states with  $y_i = 1$ . The actual computation of the XOR operation is accomplished by harnessing the way the output tiles connect to the input tiles. Each output tile ( $y_i$ ) attaches to a unique combination of one input tile ( $x_i$ ) and one output tile ( $y_{i-1}$ ) and leaves one sticky end open that encodes its own value ( $y_i$ ) so that another output tile may attach to it. Thus, only the

output tiles that represent the correct solution to the problem are able to attach to the input tiles.

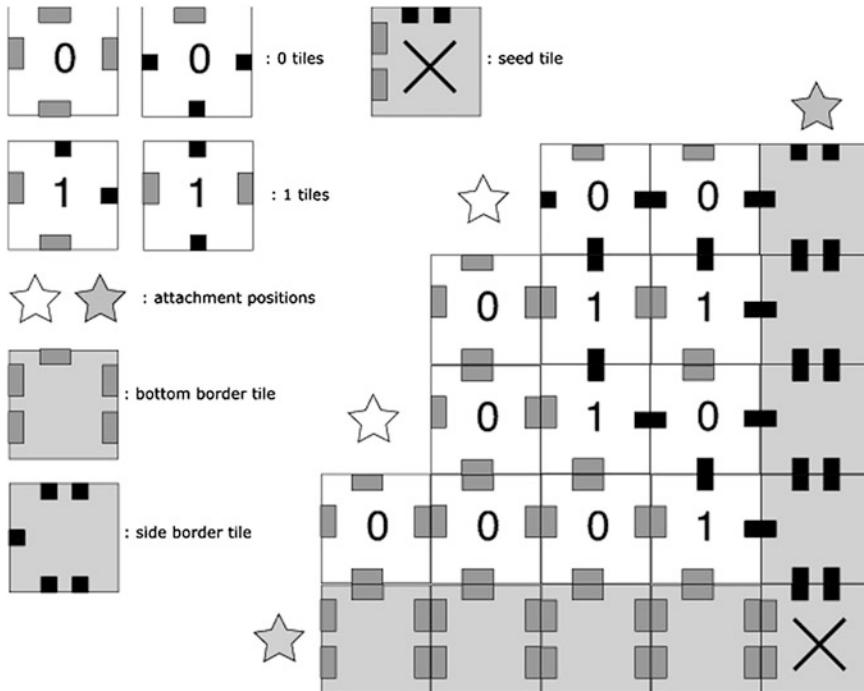
#### Algorithmic DNA Self-Assembly

As another type of DNA-based paradigm capable of autonomous computing, we give an example of algorithmic self-assembly of DNA tiles, as first suggested, and then implemented by Winfree. Algorithmic self-assembly is an extension of various DNA nanotechnologies developed over the years by Seeman’s group and is based on a vision that one can encode in a set of tiles the growth of an aperiodic crystal. While aperiodic crystals seem on the surface irregular, the position of each component of such crystal is actually precisely encoded by a program.

The approach is based on a rigid set of DNA tiles, such as double- or triple-crossover tiles. Unlike standard (i.e., crossover-free) structures, such as three- and four-way junctions, these molecules are sufficiently rigid to define precise positions of other tiles interacting with them, which could lead to a regular crystal growth. This has led to many impressive periodic 2D structures. Interactions between tiles occur through complementary overhangs. For example, each of the DAO-E tiles in Fig. 7 projects four overhangs (schematically presented as different shapes), each of which can be independently defined.

We examine two computations that have been demonstrated by means of algorithmic self-assembly: a binary counter and a Sierpinski triangle.

The binary counter (Barish et al. 2005; Rothemund and Winfree 2000) uses seven different types of tiles: two types of tiles representing 1, two types representing 0, and three types for the creation of a border (corner, bottom, and side tiles). The counter (Fig. 5) works by first setting up a tile border with the border tiles – it is convenient to think of the “side” border tiles as being on the right, as then the counter will display the digits in the customary order. Two border tiles bind together with a strong, double bond, while all other tiles bind to each other and to border tiles with a single bond. Since any tile except a border tile must bind to two additional tiles in order to have two bonds, but a border tile and another border tile of the correct type will form a double



**Molecular Automata, Fig. 5** A binary counter in the process of self-assembly. The seed tile starts off the assembly. The right side and bottom border tiles connect to each other with double bonds, while all the other tiles connect

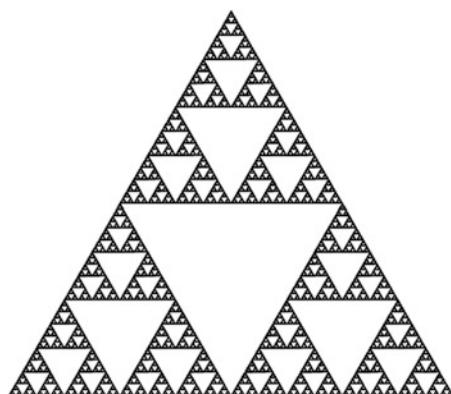
bond with each other, a stable border forms before other stable formations, composed of nonborder tiles, are created. The bottom and side border tiles are designed such that the only tile that may bind in the border's corner (to both a side and a bottom border tile) is a specific type of 1 tile. Only one of the 0 tiles may bind to both this 1 tile and the bottom of the border, and this type of 0 tile may also bind to itself and the bottom of the border and thus may fill out the left side of the first number in the counter with leading zeros. The only type of tile which may bind both above the 1 in the corner and to the right side of the border is the other type of 0 tile, and the only tile which may bind to the left of it is a 1 tile – we get the number 10 or two in binary. The tile-binding rules are such that this can continue similarly up the structure, building numbers that always increment by one.

We now look at how an aperiodic crystal corresponding to the so-called Sierpinski triangle was constructed (Rothenmund et al. 2004). First we consider what this fascinating figure is in the

with single bonds. A tile needs two single bonds (or one double bond) to form a stable attachment to the structure; the marked attachment positions show where a tile can form a stable attachment

abstract. The Sierpinski triangle (or *gasket*), properly speaking, is a fractal, obtained from a triangle by iteratively removing an inner triangle half its size ad infinitum; Fig. 6 is an approximation up to ten iterations. Any physical realization will provide only a finite number of iterations and can thus only be an approximation of the true fractal; some realizations have been fabricated in the past with interesting physical properties (Gordon et al. 1986).

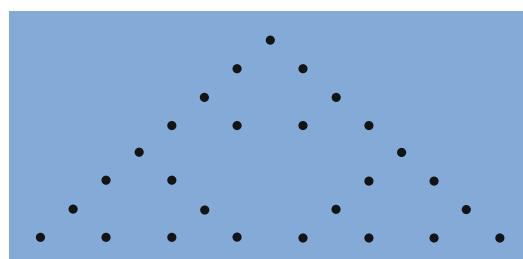
A peculiar connection exists with Pascal's triangle (Table 1), the table of binomial coefficients  $\binom{n}{k}$ . Each entry in Pascal's triangle is the sum of the two entries right above it, according to the equality  $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ . If we turn an initial section consisting of  $2^m$  rows of Pascal's triangle into an image by mapping odd numbers to black and even numbers to white (Table 2), we get an approximation to the Sierpinski triangle; various generalizations are possible, see (Holter et al. 1986; Pickover 1990).



**Molecular Automata, Fig. 6** A Sierpinski triangle as a fractal image

**Molecular Automata, Table 1** The first eight rows of Pascal's triangle

**Molecular Automata, Table 2** Marks indicate the odd entries in the first eight rows of Pascal's triangle



Thus, we could generate the Sierpinski triangle by first generating Pascal's triangle, row by row, using the above formula. But instead of computing the entries of Pascal's triangle, which are large numbers, we can compute modulo 2, i.e., only keep track of whether the numbers are odd or even. Supposing numbers were written in binary, this is equivalent to keeping only the least

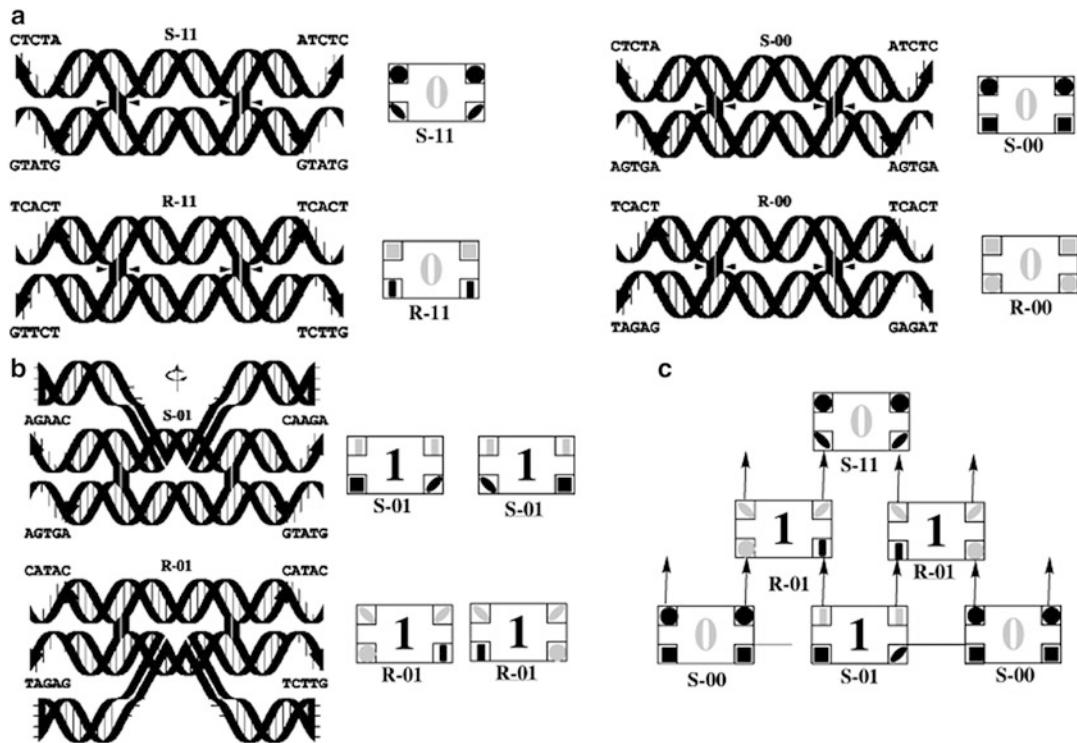
**Molecular Automata, Table 3** The computational schema for the construction of an approximation to a Sierpinski triangle

$$\begin{matrix}
 & & & & 1 \\
 & & & 1 & 0 & 1 \\
 & & 1 & 0 & 0 & 0 & 1 \\
 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
 \end{matrix}$$

significant bit. In that case, the addition operation degenerates into an XOR. This is the chosen method for DNA self-assembly of a Sierpinski triangle (Table 3).

The set of tiles that worked the best for the synthesis of the Sierpinski triangle consists of six tiles, three **R** and three **S** tiles (Fig. 7). The **R** tiles interact with the **S** tiles, and vice versa. The **R** tiles in row  $i$  calculate (display their overhang) outputs to the **S** tiles in the row  $i + 1$ , based on inputs (interactions with complementary overhangs) from the **S** tiles in row  $i - 1$  (Fig. 7c). Thus, each row is composed of either **R** or **S** tiles. There are four tiles (**S-00**, **S-11**, **R-00**, **R-11**) which are binary-encoded with values of 0 (Fig. 7a). These are incorporated in the growing crystal between two adjacent tiles in row  $i - 1$  that have both the value 0 or both the value 1. There are two tiles (**R-01** and **S-01**) with values of 1 (Fig. 7b), and they have additional loops that will show as bright regions on the atomic force microscopy (AFM) characterization (cf. Fig. 9). These tiles are incorporated in between two tiles in row  $i - 1$  that display different values. Due to the C2 symmetry of the displayed overhangs two tiles, there is no need to define any further tiles. In fact, the tile **R-11** is never used in this particular calculation and is redundant.

Assembly proceeds as follows: Winfree and colleagues first constructed a long-input (scaffold strand) DNA, which contained binding sites for the **S-00** tiles at regular distances. The input is necessary to ensure longer periods of growth (more than 70 individual rows were grown from the input). The input also contained a small number of



**Molecular Automata, Fig. 7** The DAO-E Sierpinski set of tiles, their corresponding schematic representations, and the mechanism of their assembly. Each tile is assigned a schematic rectangular representation. An overhang is

randomly distributed sites to which the **S-01** tile binds. This is an initiation site, around which aperiodic growth occurs. Thus, the first row, immediately on the input strand, contains mostly **S-00** tiles and here and there a single **S-01** tile. The second row of **R** tiles assembles on these tiles, with **R-00** between all **S-00** tiles and **R-01** on each side of **S-01** (i.e., between **S-01** and **S-00** tiles). An example of this type of crystal growth is shown in Fig. 8, while Fig. 9 shows the actual AFM results.

In principle, tiling of surfaces is Turing complete. While it is certain that many interesting aperiodic crystals can be encoded using DNA tiles, the main limitation for further progress at this moment is the high error rate (1–10 %) of the addition of individual tiles. Explicit error correction using some form of redundancy or “self-healing” tile sets to recover from errors is called for (Winfree 2006).

represented by the shapes in each corner, with complements being assigned the same shapes but in black and gray. In c we give an example of assembly

## Molecular Automata as Transducers and Controllers

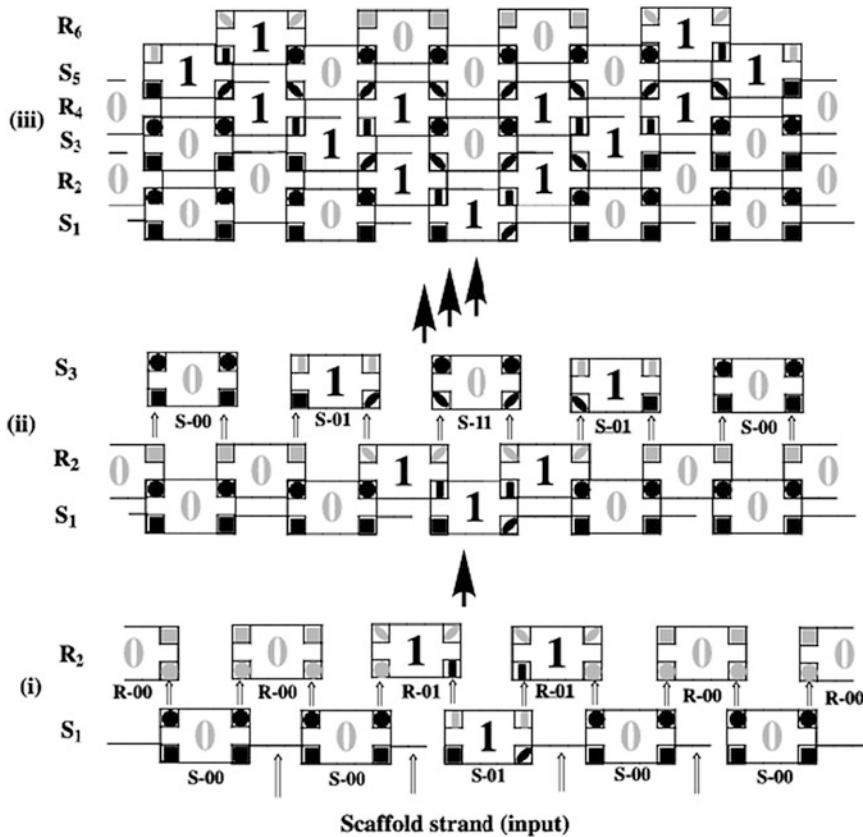
### Preliminaries

#### Finite-State Transducers

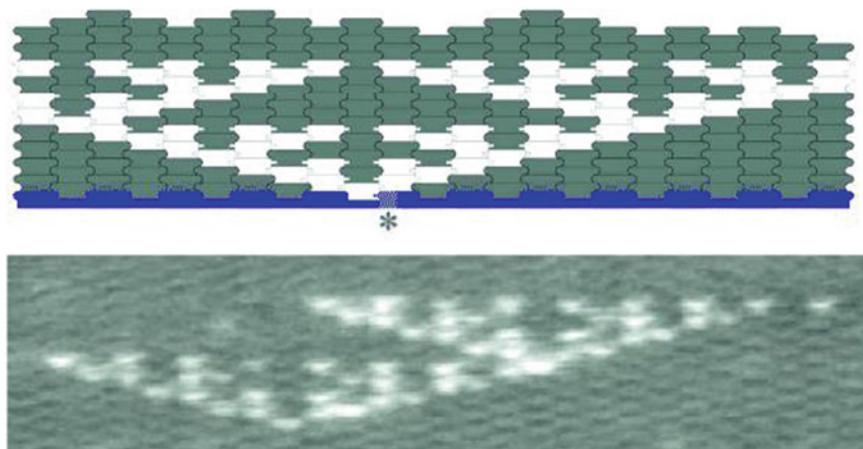
A finite-state transducer is a variation of the deterministic finite-state automaton which, in addition to consuming one input symbol at each step of operation, produces as output a string over some alphabet. A special case is the Mealy automaton (Mealy 1955), which emits one output symbol at each step, so that the length of the output is the same as the length of the input; an example is shown in Fig. 10.

#### Molecular Automata as Language Recognizers

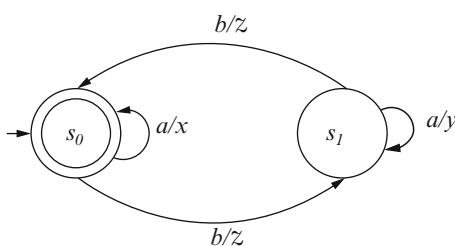
Shapiro’s group used the same type of cascades as in their finite automata to create a molecular



**Molecular Automata, Fig. 8** A schematic example of the assembly of an aperiodic 2D crystal from tiles encoding the Sierpinski triangle



**Molecular Automata, Fig. 9** AFM picture (*bottom*) of a representative aperiodic crystal encoding the Sierpinski triangle in its structure (*schematic, top*) (From Rothemund et al. 2004)



**Molecular Automata, Fig. 10** Example Mealy automaton. The input alphabet, the states, and the transition function are as in Fig. 1. The output alphabet is  $\{x,y,z\}$ . Successive runs of  $a$ s in the input are transduced alternately into runs of  $x$ s and runs of  $y$ s;  $b$ s are transduced into  $z$ s

transducer as a proof-of-concept for a diagnostic automaton (Benenson et al. 2004). This design is explained in section “[Therapeutic and Diagnostic Automata](#).”

Using a conceptually different model, Stojanovic and Stefanovic created molecular transducers from deoxyribozyme-based logic gates (see section “[Deoxyribozyme-Based Logic Gates as Transducers](#)”), which have been used as components for the construction of adders (section “[Adders and Other Elementary Arithmetic and Logic Functions](#)”) and automata for game strategies (sections “[Automata for Games of Strategy](#)” and “[Improved Automata for Games of Strategy](#)”).

**Deoxyribozyme-Based Logic Gates as Transducers**  
Simple DNA logic-gate transducers have been constructed from deoxyribozymes by Stojanovic et al. (2002). Deoxyribozymes are enzymes made of DNA that catalyze DNA reactions, such as the cleavage of a DNA strand into two separate strands or the ligation of two strands into one. These can be modified to include allosteric regulation sites, to which specific control molecules can bind and so affect the catalytic activity.

There is a type of regulation site to which a control molecule must bind before the enzyme can complex with (i.e., bind to) the substrate: thus, this control molecule promotes catalytic activity. Another type of regulation site allows the control molecule to alter the conformation of the enzyme’s catalytic core, such that even if the substrate has bound to the enzyme, no cleavage

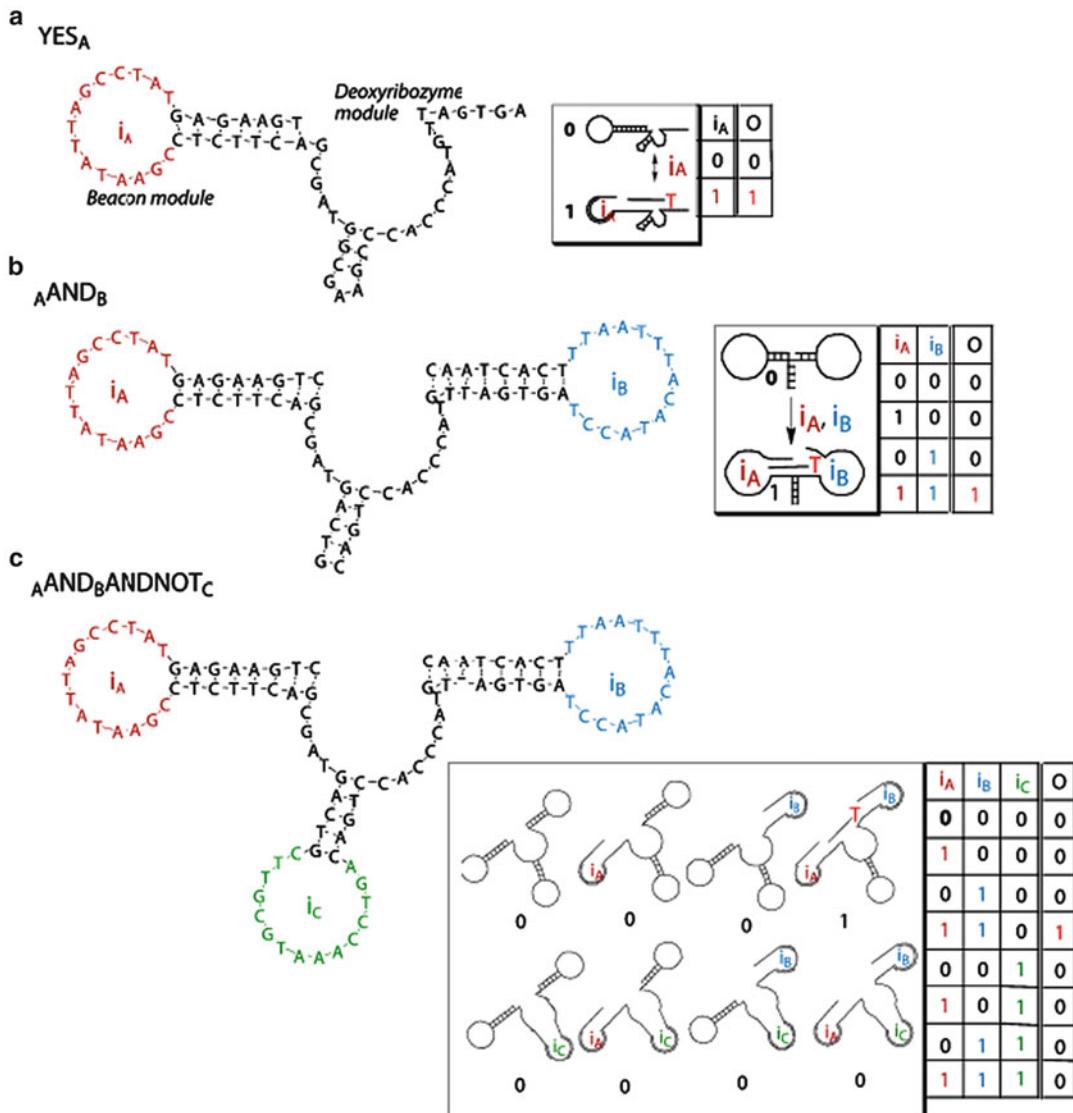
occurs; thus, this control molecule suppresses or inhibits catalytic activity. This allosterically regulated enzyme can be interpreted as a logic gate, the control molecules as inputs to the gate, and the cleavage products as the outputs. This basic logic gate corresponds to a conjunction, such as, e.g.,  $a \wedge b \wedge \neg c$ , here assuming two promotory sites and one inhibitory site and using  $a$  and  $b$  as signals encoded by the promotor input molecules and  $c$  as a signal encoded by the inhibitor input molecule. Deoxyribozyme logic gates are constructed via a modular design (Stojanovic et al. 2001, 2002) that combines molecular beacon stem-loops with hammerhead-type deoxyribozymes, Fig. 11. A gate is active when its catalytic core is intact (not modified by an inhibitory input) and its substrate recognition region is free (owing to the promotive inputs), allowing the substrate to bind and be cleaved.

Correct functioning of individual gates can be experimentally verified through fluorescent readouts (Fig. 21). The inputs are compatible with sensor molecules (Stojanovic and Kolpashchikov 2004) that can detect cellular disease markers. Final products (outputs) can be tied to release of small molecules. All products and inputs (i.e., external signals) must be sufficiently different to minimize the error rates of imperfect oligonucleotide matching, and they must not bond to one another. The gates use oligonucleotides as both inputs and outputs, so cascading gates is possible without external interfaces. Two gates are coupled in series if the product of an “upstream” gate specifically activates a “downstream” gate. A series connection of two gates, an upstream ligase and a downstream phosphodiesterase, has been experimentally validated (Stojanovic et al. 2005).

## Prototypes

### Therapeutic and Diagnostic Automata

Using similar principles to their finite automata (section “[Benenson’s Finite Automaton](#)”), Shapiro’s groups created a molecular transducer to assess the presence or absence of a series of oligonucleotides, as a proof of concept for a diagnostic automaton (Benenson et al. 2004). The automaton analyzes up- and downregulated

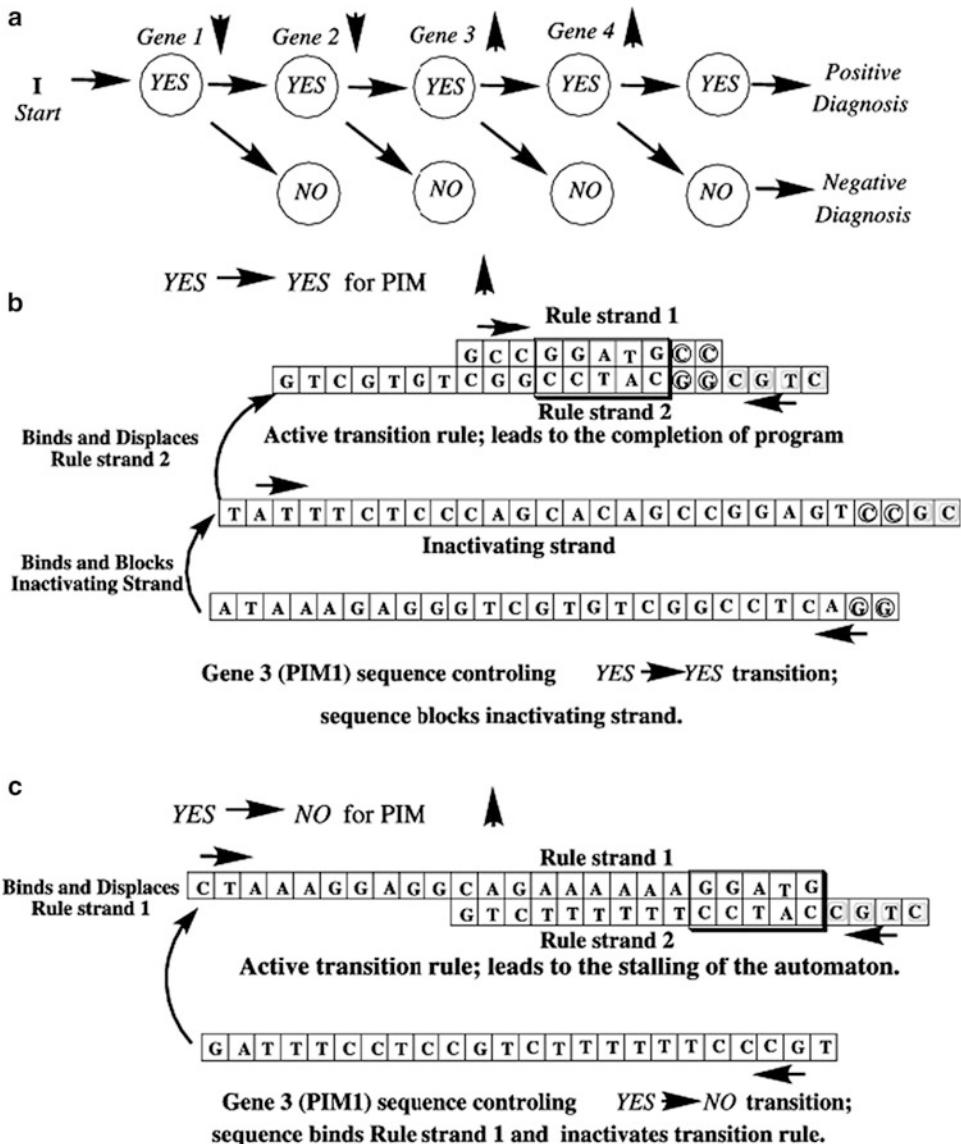


**Molecular Automata, Fig. 11** Examples of deoxyribozyme-based logic gates. (a) A YES gate showing deoxyribozyme core and molecular beacon allosteric controlling region; this gate is active in the presence of one activating input. (b) A two-input AND produces

outputs only if both inputs are present. (c) A three-input ANDANDNOT gate produces outputs only if the two activating inputs (a, b) are present and the inhibiting input (c) is absent

mRNA molecules that are characteristic for some pathological states (Fig. 12a). The authors introduced several new concepts in their work, primarily in order to maximize the tightness of control over a diagnostic cascade. We present here a somewhat simplified explanation that focuses on the underlying principles and the ability of these finite automata to analyze oligonucleotide inputs.

Input molecules for diagnostic automata are similar to those in Fig. 3. Each “symbol” in the input now contains recognition sites (“diagnostic” symbol), which are, upon cleavage by *FokI*, recognized by transition rules. In turn, these transition rules are regulated by the expression of genes (i.e., mRNA) characteristic for certain types of cancer.



**Molecular Automata, Fig. 12** Therapeutic automata: (a) A diagnostic procedure is based on an analysis of a set of genes (e.g., Genes 1–4) which have the levels of expression characteristic for a particular disease; these genes regulate the transition rules that process the input molecule (see below). Input (I) molecule, similar to Fig. 3, encodes the symbols that are cleaved by these transition rules. Symbols are recognized by two types of transition rules regulated by the mRNA molecule transcribed from a gene; one type leads to continuation of the cascade, while

the other leads to no further processing. (b)  $YES \rightarrow YES$  transition rule for an upregulated gene 3 in this cascade (e.g., PIM1); in the absence of the gene, this transition rule is deactivated by the inactivating strand that binds stronger to strand 2 of the rule, displacing strand 1. (c)  $YES \rightarrow NO$  Transition rule Gene 3 expressed is formed from two software strands, by a displacement of a blocking (protector) strand. (d) Transition rule Gene 1 not expressed is degraded in the presence mRNA for Gene 1, thus stalling the automaton

If we are looking to diagnose an increase in certain mRNA level, there are, potentially, two types of transition rules that can cleave each of

the symbols after these are exposed as overhangs through an action by *FokI*: (1)  $YES \rightarrow YES$  transitions, which will lead to the next step of

processing the input because a new overhang, recognizable by the next set of transition rules, will be exposed; and (2) YES → NO transitions, in which *FokI* cleaves off from the site that is recognized by the next set of transition rules, and, thus, these transitions cause the automaton to stall (i.e., no existing transition rules recognize the new overhang).

The YES → YES transition is activated by the presence of mRNA, through the removal of an inactivating (blocking) oligonucleotide from its complex with one of the strands making a transition rule (Fig. 12b). In contrast, the YES → NO transitions are deactivated by an mRNA, because mRNA will compete for binding to one of the strands in the transition rule, thus, effectively breaking up the transition rule. In case we want to diagnose a downregulation of mRNA (genes 1 and 2 in Fig. 12a), the situation is reversed: The lack of characteristic mRNA leaves the YES → YES transition active, while the presence of it would break this transition rule apart. In contrast, the presence of mRNA will activate YES → NO transition by removing the inactivating oligonucleotide.

A diagnostic automaton can be turned into a “therapeutic” automaton by modifying the events at the end of the cascade. Instead of releasing a terminator symbol (cf. Figs. 3 and 4, Shapiro and colleagues described a degradation of a stem-loop and a release of a linear oligonucleotide, which had the potential for a therapeutic (antisense) action. A further interesting detail was that a negative diagnosis could be coupled to a parallel cascade, which can release another oligonucleotide, complementary to and blocking the activity of a therapeutic oligonucleotide. The authors also described independent and parallel action of two different automata in the same solution. From the perspective of this review, it is also important that Winfree proved that these automata are capable of general Boolean computing (Soloveichik and Winfree 2005).

The “therapeutic automaton” is in many respects the most challenging ex vivo molecular system that has ever been constructed. Although there are many reasons to doubt eventual practical applications as a “doctor in cell,” it will always

remain an extremely impressive intellectual exercise, which will serve as an inspiration for different systems.

#### Adders and Other Elementary Arithmetic and Logic Functions

Another type of transducing automaton is a number adder, which accepts as input two numbers expressed as strings of bits (binary digits) and produces as output the string of bits corresponding to their sum. On the surface, these may not appear to have direct relevance with diagnostic and therapeutic automata; however, adders serve as extremely important computational components, and their development begins to hint at the power in automaton building through the multiple layering of logic gates.

In electronics, binary adders commonly use as a building block the full adder, which is a device that takes as input three bits (normally two digits from two numbers and a carry-in bit) and produces a sum bit and a carry bit. The full adder is commonly realized using two half-adders; a half-adder takes as input two bits and produces a sum bit and a carry bit. A half-adder, in turn, is realized using simple logic gates. A logic gate takes some number of input bits (often just one or two) and produces one output bit as a Boolean function of the inputs, such as a negation (NOT gate) or a conjunction (AND gate). Adders and logic gates are simple circuits with obvious applications. Numerous molecular devices with logic gate and adder functions (as well as mixed-mode devices such as molecular-optical) have been introduced in recent years (Collier et al. 1999; Credi et al. 1997; de Silva et al. 1999, 1993, 1997; de Silva and McClenaghan 2000; Ellenbogen and Love 2000; Huang et al. 2001; Lederman et al. 2006; Yurke et al. 1999).

Logic gates built from deoxyribozymes were first described by Stojanovic and Stefanovic (Stojanovic et al. 2002) and were subsequently used to build both a half-adder (Stojanovic and Stefanovic 2003a) and a full adder (Lederman et al. 2006). A half-adder was achieved by combining three two-input gates in solution, an AND gate for the carry bit, and an XOR, realized using two ANDNOT gates (gates of the form  $x \wedge \neg y$ ) for the

sum bit. The two substrates used are fluorogenically marked, one with red tetramethylrhodamine (T) and the other with green fluorescein (F), similar to Fig. 21, and the activity of the device can be followed by tracking the fluorescence at two distinct wavelengths. The results, in the presence of  $Zn^{2+}$  ions, are shown in Fig. 13. When both inputs are present, only the green fluorescein channel (carry bit) shows a rise in fluorescence. When only input  $i_1$  is present or only input  $i_2$  is present, only the red tetramethylrhodamine channel (sum bit) rises. With no inputs, neither channel rises. Thus, the two bits of output can be reliably detected and are correctly computed.

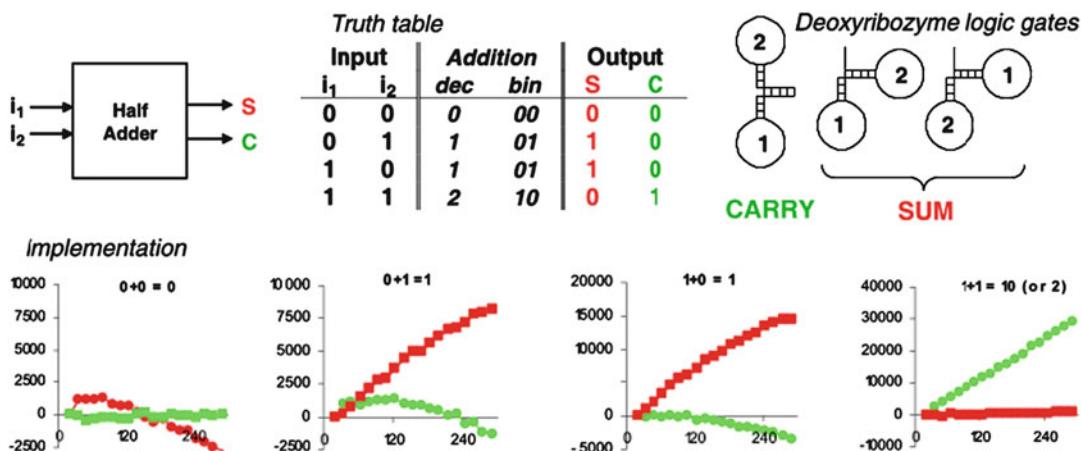
A molecular full adder was similarly constructed, requiring seven molecular logic gates in total: three AND gates for computation of the SUM bit and four 3-input gates for calculation of the CARRY bit (Fig. 14). The requirement for a 3-input ANDAND gate as well as several ANDNOTANDNOT gates necessitated additional deoxyribozyme logic gate design. This was achieved by precomplexing gates with a complementary oligonucleotide that would subsequently be removed by addition of inputs (Fig. 15). The length of input increased to 30 nucleotides and served a dual function – to activate controlling elements directly and to remove precomplexed

oligonucleotides when necessary. The combination of stem-loop regulation in cis and controlling elements supplied in trans is reminiscent of some classical *in vivo* regulatory pathways. This simple demonstration points toward the construction of even more complex artificial networks, which could mimic natural systems through the use of multifunctional regulatory components.

### Automata for Games of Strategy

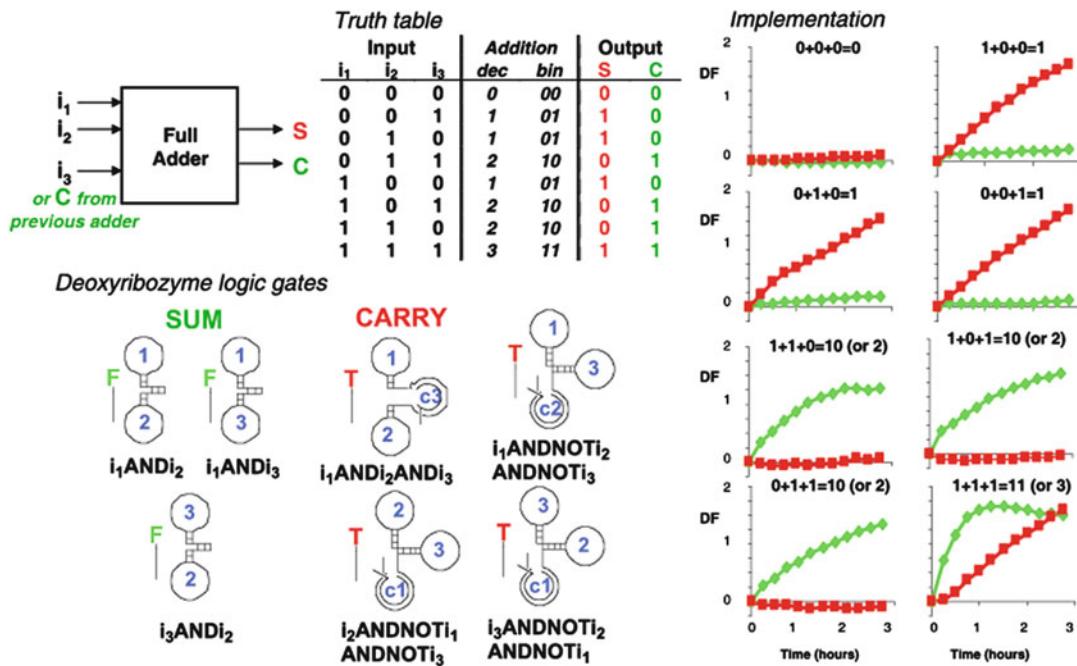
Initial construction of networks capable of simple arithmetic operations suggested the possibility that circuits of highly regulated deoxyribozymes could support complex molecular decision trees. This was subsequently demonstrated by Stojanovic and Stefanovic in the construction of the first interactive molecular transducing automaton, designed to perfectly play the game of tic-tac-toe against a human opponent (Stojanovic and Stefanovic 2003b). To understand how this was achieved, the structure of the game will be briefly examined.

A sequential game is a game in which players take turns making decisions known as moves. A game of perfect information is a sequential game in which all the players are informed before every move of the complete state of the game. A strategy for a player in a game of perfect



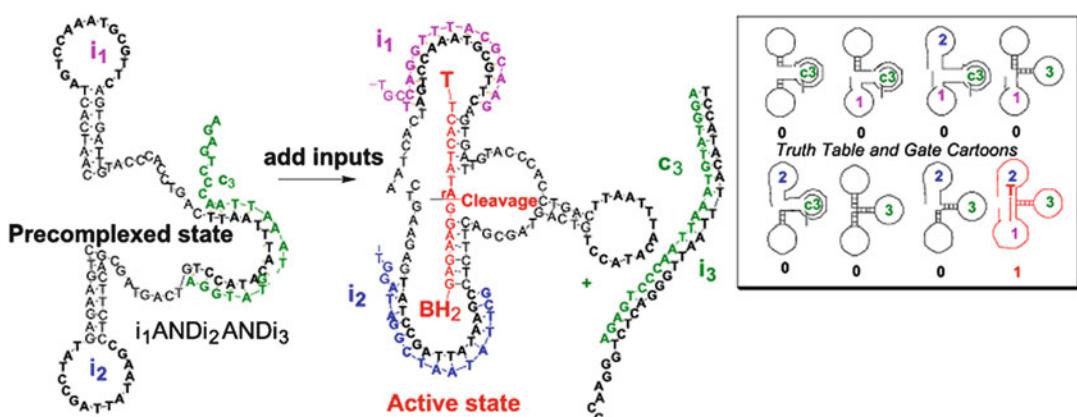
**Molecular Automata, Fig. 13** Components of a molecular half-adder, which adds two binary inputs to produce a sum and carry digit. *Top:* Circuit diagram, truth table of calculations and deoxyribozyme logic gates required for construction. *Bottom:* implementation of the

deoxyribozyme-based half-adder. The sum output is seen by an increase in red channel (tetramethylrhodamine) fluorescence ( $0 + 1 = 1$  and  $1 + 0 = 1$ ), and the carry output is seen by an increase in green channel (fluorescein) fluorescence ( $1 + 1 = 2$ )



**Molecular Automata, Fig. 14** Components of a molecular full adder, which adds three binary inputs to produce a sum and carry digit. *Left:* circuit diagram, truth table of calculations and deoxyribozyme-based logic gates required for construction. *Right:* implementation of the

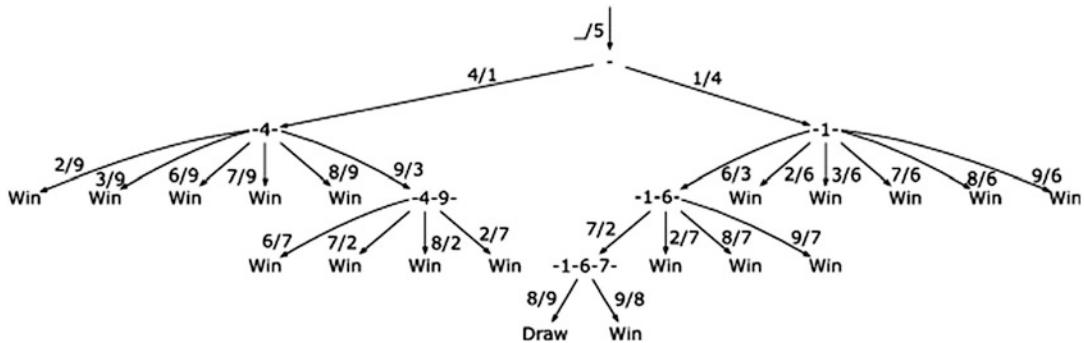
deoxyribozyme-based full adder. The sum output is seen by an increase in red channel (tetramethylrhodamine) fluorescence, and the carry output is seen by an increase in green channel (fluorescein) fluorescence



**Molecular Automata, Fig. 15** Three-input ANDAND gate structure

information is a plan that dictates what moves that player will make in every possible game state. A strategy tree is a (directed, acyclic) graph representation of a strategy (an example is shown in Fig. 16). The nodes of the graph represent reachable game states. The edges of the graph represent

the opponent's moves. The target node of the edge contains the strategy's response to the move encoded on the edge. A leaf represents a final game state and can, usually, be labeled either win, lose, or draw. Thus, a path from the root of a strategy tree to one of its leaves represents a game.



**Molecular Automata, Fig. 16** The chosen strategy (game tree) for the symmetry-pruned game of tic-tac-toe, drawn as the diagram of a Mealy automaton. Each state is

In a tree, there is only one path from the root of the tree to each node. This path defines a set of moves made by the players in the game. A player's move set at any node is the set of moves made by that player up to that point in a game. For example, a strategy's move set at any node is the set of moves dictated by the strategy along the path from the root to that node. A strategy is said to be feasible if, for every pair of nodes in the decision tree for which the opponent's move sets are equal, one of the following two conditions holds: (1) the vertices encode the same decision (i.e., they dictate the same move) or (2) the strategy's move sets are equal. A feasible strategy can be successfully converted into Boolean logic implemented using monotone logic gates, such as the deoxyribozyme logic gates.

In the first implementation of the tic-tac-toe automaton, the following simplifying assumptions were made to reduce the number and complexity of needed molecular species. The automaton moves first and its first move is into the center (square 5, Fig. 17). To exploit symmetry, the first move of the human, which must be either a side move or a corner move, is restricted to be either square 1 (corner) or square 4 (side) (Any other response is a matter or rotating the game board.)

The game tree in Fig. 16 represents the chosen strategy for the automaton. For example, if the human opponent moves into square 1 following the automaton's opening move into square 5, the automaton responds by moving into square 4. If the human then moves into square 6, the automaton responds by moving into square 3. If the

labeled with the string of inputs seen on the path to it. Each edge is labeled  $a, b$ , where  $b$  is the output that is activated on input  $a$

**Molecular Automata,**

**Fig. 17** The tic-tac-toe game board with the field numbering convention

1	2	3
4	5	6
7	8	9

human then moves into square 7, the automaton responds by moving into square 2. Finally, if the human then moves into square 8, the automaton responds by moving into square 9, and the game ends in a draw.

This strategy is feasible; therefore, following a conversion procedure, it is possible to reach a set of Boolean formulas that realize it, given in Table 4 (For a detailed analysis of feasibility conditions for the mapping of games of strategy to Boolean formulas, see; Andrews 2005).

Human interaction with the automaton is achieved by the sequential addition of nine input oligonucleotides ( $i_1-i_9$ ). Each oligonucleotide is 15–17 nucleotides long and has a unique sequence of A, T, C, and Gs, with no more than 4 nucleotides in common between each oligonucleotide sequence (Fig. 18). The input numbering directly matches the square numbering of the tic-tac-toe game board (Fig. 17). Hence, to signify a move into square 9, the human would choose input  $i_9$  for addition.

The game is played in a standard laboratory plate, with nine wells representing each square of the tic-tac-toe game board. The automaton is prepared by the addition of fluorescent substrate and a specific set of deoxyribozyme-based logic gates in each well. Thus, the automaton moves are

**Molecular Automata, Table 4** Boolean formulas resulting from the tic-tac-toe game tree. The inputs are designated  $i_k$ , and the outputs are designated  $o_k$

$$\begin{aligned}
 O_1 &= i_4 \\
 O_2 &= (i_6 \wedge i_7 \wedge \neg i_2) \vee (i_7 \wedge i_9 \wedge \neg i_1) \vee (i_8 \wedge i_9 \wedge \neg i_1) \\
 O_3 &= (i_1 \wedge i_6) \vee (i_4 \wedge i_9) \\
 O_4 &= i_1 \\
 O_5 &= 1 \\
 O_6 &= (i_1 \wedge i_2 \wedge \neg i_6) \vee (i_1 \wedge i_3 \wedge \neg i_6) \vee (i_1 \wedge i_7 \wedge \neg i_6) \vee (i_1 \wedge i_8 \wedge \neg i_6) \vee (i_1 \wedge i_9 \wedge \neg i_6) \\
 O_7 &= (i_2 \wedge i_6 \wedge \neg i_7) \vee (i_6 \wedge i_8 \wedge \neg i_7) \vee (i_6 \wedge i_9 \wedge \neg i_7) \vee (i_9 \wedge i_2 \wedge \neg i_1) \\
 O_8 &= i_9 \wedge i_7 \wedge \neg i_4 \\
 O_9 &= (i_7 \wedge i_8 \wedge \neg i_4) \vee (i_4 \wedge i_2 \wedge \neg i_9) \vee (i_4 \wedge i_3 \wedge \neg i_9) \vee (i_4 \wedge i_6 \wedge \neg i_9) \vee (i_4 \wedge i_7 \wedge \neg i_9) \vee (i_4 \wedge i_8 \wedge \neg i_9)
 \end{aligned}$$

*i1* 5' TCT GCG TCT ATA AAT  
*i2* 5' ATC GTA TGT TGT TCA  
*i3* 5' GTA TAG TCT GTT TGT  
*i4* 5' G TAA GTG CTC AAA TGT C  
*i6* 5' G TCT AAT TCT CAC GGT C  
*i7* 5' TAG TCT GTG TGT TGT  
*i8* 5' TCT ATA TGA GCG TAA  
*i9* 5' TGT CCA TCT AAA TCC

**Molecular Automata, Fig. 18** Sequences of the nine oligonucleotide inputs used to indicate human move positions in the MAYA automaton

predetermined by the inherent logic provided in each well, and in the chosen arrangement, the automaton never loses because it plays according to a perfect strategy. The arrangement of deoxyribozyme logic gates corresponding to the above formulas is given in Fig. 19. This is the initial state of the nine wells of a well-plate in which the automaton is realized in the laboratory. The automaton was named MAYA since it uses a molecular array of YES and AND gates to determine responses to human moves.

An example game played against MAYA is shown in Fig. 20. The play begins when  $Mg^{2+}$  ions, a required cofactor, are added to all nine wells, activating only the deoxyribozyme in well 5, i.e., prompting the automaton to play its first move into the center, since this well contains a single active deoxyribozyme which begins cleaving to produce fluorescent output. The human player detects this move by monitoring wells for increase in fluorescence emissions using a fluorescent plate reader.

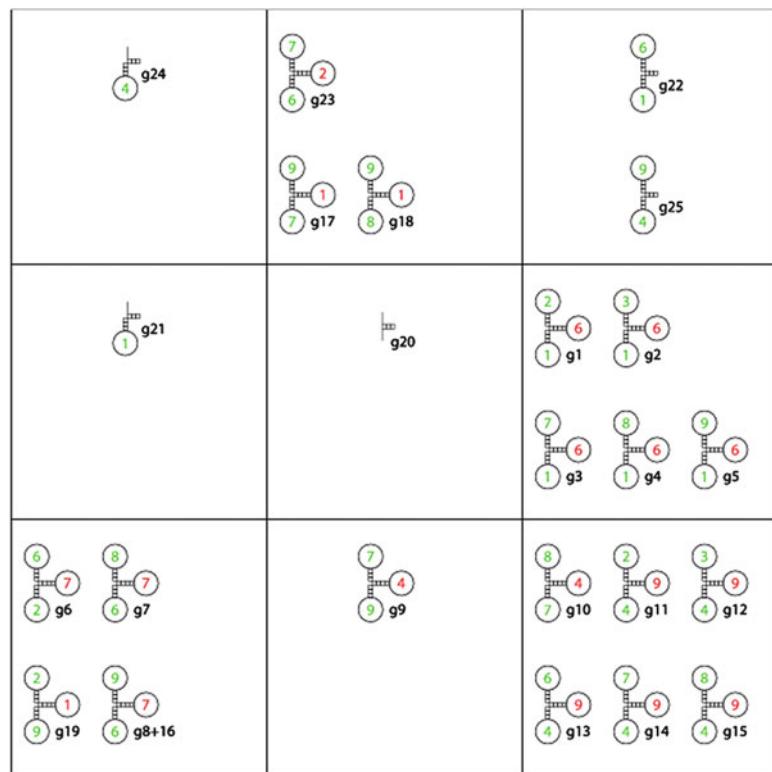
After that, the game branches according to the human's inputs. In response to the automaton's first move, the human player may choose either

well 1 or well 4 in this restricted game and thus will add input oligonucleotide  $i_1$  or  $i_4$  to every well of the game board, so that each well of the automaton receives information on the position of the human move. This creates a chain reaction among the deoxyribozyme logic gates in each well, opening individual stem-loops complementary to the chosen input. However, only one gate in one well of the board will become fully activated in response to this first human move: the YES  $i_1$  gate placed in well 4 (if the human added  $i_1$  to every well) or the YES  $i_4$  gate in well 1 (if the human added  $i_4$  to every well). The activated gate subsequently cleaves to produce fluorescent outputs, which is again detected by the human via fluorescence monitoring.

Subsequent human moves are unrestricted, so the human may choose to play in any of the remaining wells where neither player has yet moved. However, at this point, the automaton is poised to win the game, either through a diagonal three in a row (wells 1, 5, 9; if the human chose to play in well 4) or through a horizontal three in a row (wells 4, 5, 6; if the human chose to play in well 1). Assuming perfect play, the human will choose to block an automaton win by playing into well 9 (adding input  $i_9$  to each well) or well 6 (adding input  $i_6$  to each well). This would again cause a chain reaction with complementary stem-loops in every well, but again, only a single logic gate would become fully activated (either gate  $i_4$  AND  $i_9$  or  $i_1$  AND  $i_6$  in well 3, depending on the human's initial moves). The game thus continues by repeated human input addition and

**Molecular Automata,**

**Fig. 19** Realizing a tic-tac-toe automaton using deoxyribozyme logic. The center well contains a constitutively active deoxyribozyme. Each of the eight remaining wells contains a number of deoxyribozyme logic gates as indicated. In the schematic, green numbers are indices to the inputs that appear as positive literals in conjunctions; red as negative



subsequent fluorescent monitoring until either the human makes an error leading to an automaton win or there is a draw as illustrated in Fig. 20.

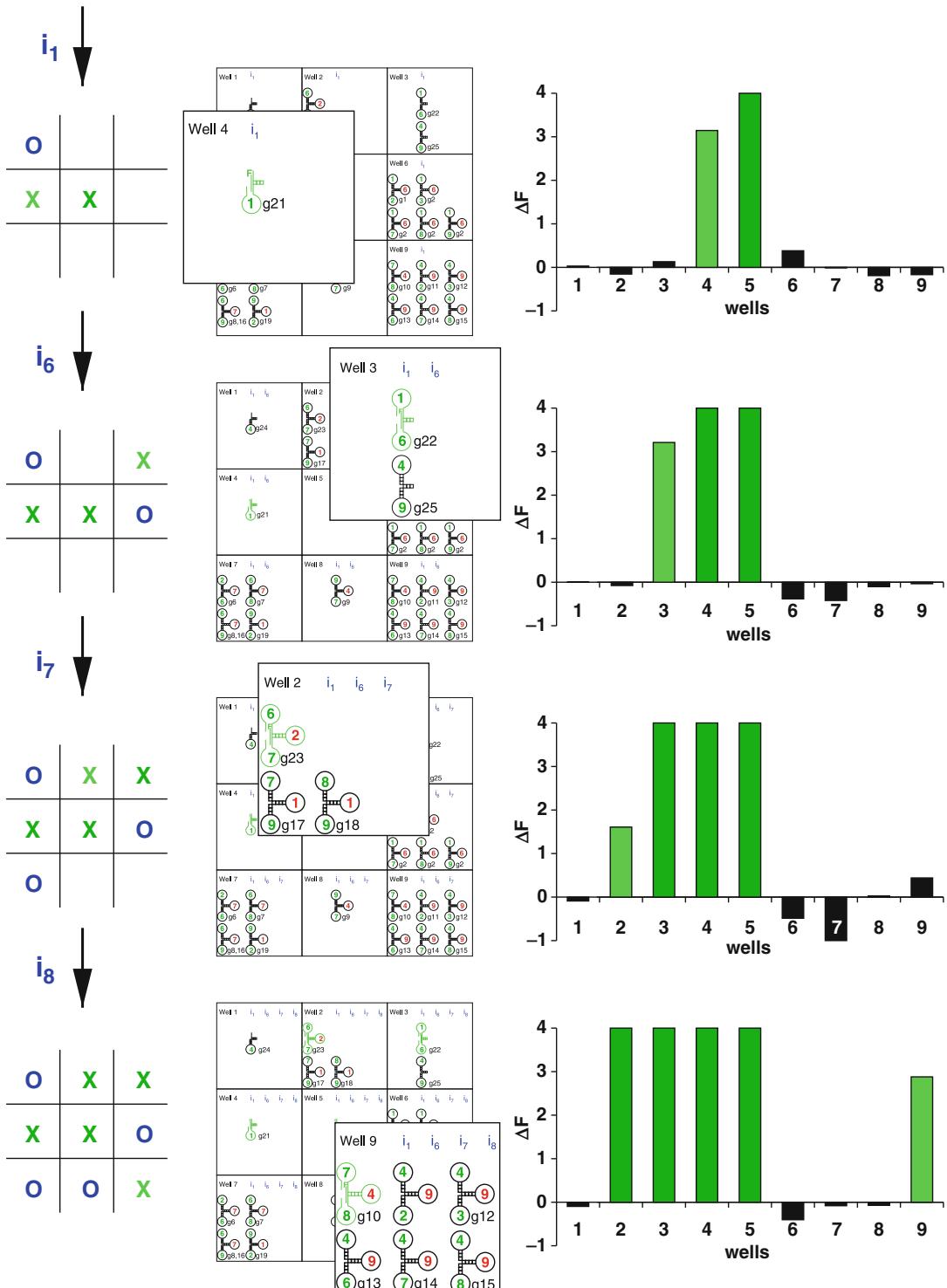
#### Improved Automata for Games of Strategy

In order to further probe the complexity with which a molecular automaton could be rationally constructed, a second version of MAYA was constructed (Macdonald et al. 2006). MAYA-II plays a nonrestricted version of tic-tac-toe where the automaton still moves first in the middle square, but the human player may choose to respond in any of the remaining squares. The new automaton was also designed to be more user-friendly than the original MAYA, allowing monitoring of both the automaton and human moves using a dual-color fluorescence output system, similar to the previously constructed half and full adder (Lederman et al. 2006; Stojanovic and Stefanovic 2003a; Fig. 21).

The complexity of MAYA-II encompasses 76 possible games, of which 72 end in an automaton win and 4 end in a draw. The required arrangement

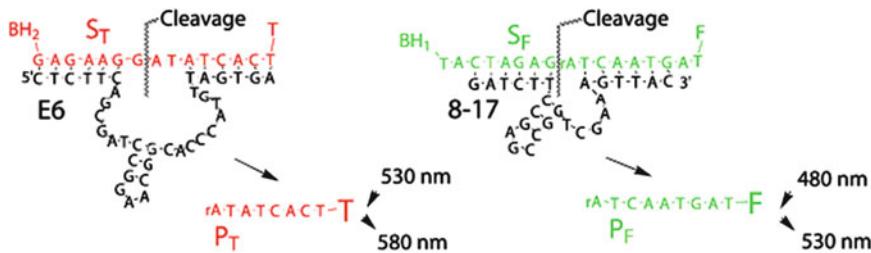
of logic gates for a perfect automaton strategy was determined by computer modeling. However, using the gate limitations of the time, a strategy using only nine inputs could not be determined by the program. Acceptable logic could be predicted by increasing the number of inputs to 32 different oligonucleotides. These inputs were coded  $i_{nm}$  where  $n$  refers to the position of the human move (1–4, 6–9) and  $m$  corresponds to the move order (1–4). For instance, to signify a move into well 9 on the first move, the human would add input  $i_{91}$  to all wells.

Using this input coding, the final arrangement of logic gates for MAYA-II's chosen strategy used 128 deoxyribozyme-based logic gates (Fig. 22); 32 gates monitor and display the human player's moves, and 96 gates calculate the automaton's moves based on the human-added inputs. Essentially, successive automaton moves are constructed as a hierarchy of AND gates, with YES gates responding to the first human move. Some NOT loops are included to prevent secondary activation in already played wells or are redundant and included to minimize cumulative nondigital



**Molecular Automata, Fig. 20** A game of tic-tac-toe that ends in a draw because both the automaton and its human

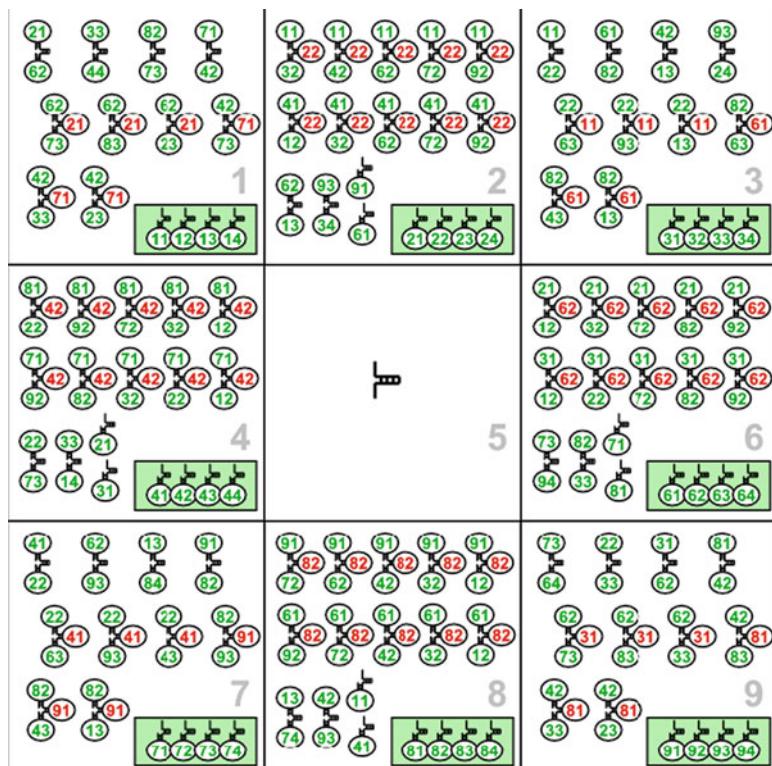
opponent play perfectly. As the human adds input to indicate his moves, the automaton responds with its own move,



**Molecular Automata, Fig. 21** Dual-color fluorescence output system utilized in MAYA-II. Automaton moves are determined using logic gates based on the deoxyribozyme E6 (Breaker and Joyce 1995; Lederman et al. 2006; Stojanovic and Stefanovic 2003a), which cleaves substrate  $S_T$  (dual labeled with tetramethylrhodamine and black hole quencher 2) to produce

product  $P_T$  and an increase in tetramethylrhodamine fluorescence. Human moves are displayed using logic gates based on the deoxyribozyme 8.17 (Lederman et al. 2006; Santoro and Joyce 1997; Stojanovic and Stefanovic 2003a), which cleaves substrate  $S_F$  (dual labeled with fluorescein and black hole quencher 1) to produce product  $P_F$  and an increase in fluorescein fluorescence

**Molecular Automata, Fig. 22** Realizing the MAYA-II automaton using deoxyribozyme logic. The center well contains a constitutively active deoxyribozyme. Each of the eight remaining wells contains a number of deoxyribozyme logic gates as indicated; boxed gates monitor human moves and the rest determine deoxyribozyme moves. In the schematic, green numbers are indices to the inputs that appear as positive literals in conjunctions; red as negative



**Molecular Automata, Fig. 20** (continued) activating precisely one well, which is shown enlarged. The newly activated gate is shown in light green. The bar chart shows the measured change in fluorescence in all the wells. Wells

that are logically inactive (contain no active gates) have black bars, and wells that are logically active have green bars (the newly active well is light green)

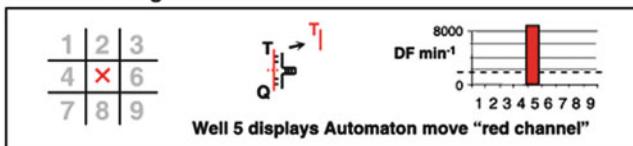
behavior in side wells over several moves. In doing this, MAYA-II is a step toward programmable and generalizable MAYAs that are trainable to play any game strategy.

Coordination of such a large number of rationally designed molecular logic gates was an unprecedented molecular engineering challenge, taking several years of construction. While spurious binding of oligonucleotides was predicted to cause serious problems, this was not observed in the building of the system. Instead, the most

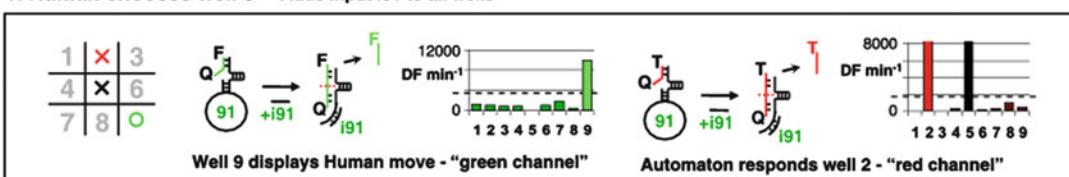
challenging aspect was titrating individual gates to produce similar levels of fluorescent signals, since the individual oligonucleotide input sequence could affect the catalytic activity of the molecule.

MAYA-II perfectly plays a general tic-tac-toe game by successfully signaling both human and automaton moves. An example of play against the automaton is shown in Fig. 23. It could be argued that by integrating more than 100 molecular logic gates in a single system, MAYA-II represents the first “medium-scale integrated molecular circuit”

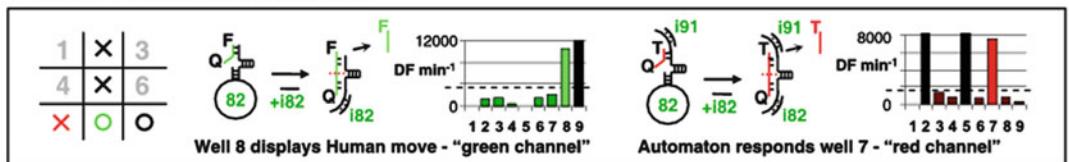
#### 0. Automaton goes first - well 5



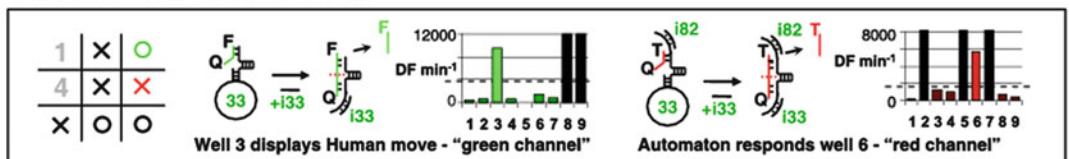
#### 1. Human chooses well 9 - Adds input i91 to all wells



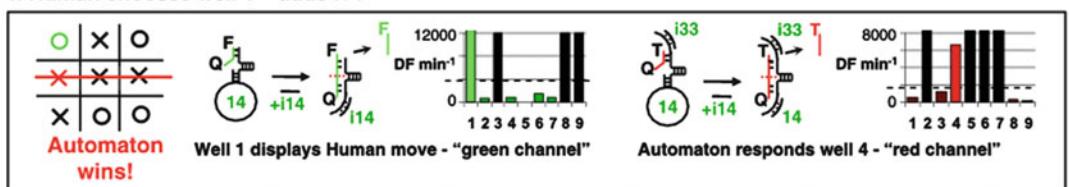
#### 2. Human chooses well 8 – adds i82



#### 3. Human chooses well 3 – adds i33



#### 4. Human chooses well 1 – adds i14



**Molecular Automata, Fig. 23** MAYA-II example game. A game played against MAYA-II ends in an automaton win, since the human opponent made an error in move 3. The human adds input to indicate their moves to every well, which causes a chain reaction to activate precisely

one well for each fluorescent color. Human moves are displayed in the fluorescein (green) fluorescence channel, and automaton moves are displayed in the tetra-methylrhodamine (red) fluorescence channel

in solution. This level of rationally designed complexity has important implications for the future of diagnostic and therapeutic molecular automata. Moreover, the increased complexity of MAYA-II enabled refinement of the deoxyribozyme logic gate model, allowing the development of design principles for optimizing digital gate behavior and the generation of a library of 32 known input sequences for future “plug and play” construction of complex automata. This library of known sequences is already being employed in the construction of other complex DNA automata (Macdonald 2007).

## Future Directions

This review mostly focused on DNA-based automata, an area of research that has its roots in Adleman’s early computing experiments (Adleman 1994). The early attempts to find applications for this type of DNA computing mostly focused on some kind of competition with silicon, for example, through massively parallel computing. After more than a decade of intensive research, we can safely conclude that without some amazing new discovery, such applications are very unlikely. Instead, it seems that the most likely applications will come from the simplest systems, in which groups of molecules will have some diagnostic or therapeutic role. Further, approaches such as Winfree’s can lead to completely new thinking in material sciences. But aside from practical considerations, experiments in which mixtures of molecules perform complex functions and execute programs are of great interest for basic science. Mixtures of molecules perform complex tasks in organisms, and some of the systems we described provide us with tools to understand how such complexity may have arisen at first.

## Bibliography

### Primary Literature

Adar R, Benenson Y, Linshiz G, Rosner A, Tishby N, Shapiro E (2004) Stochastic computing with biomolecular automata. *Proc Natl Acad Sci USA (PNAS)* 101(27):9960–9965

- Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266(5187):1021–1024
- Andrews B (2005) Games, strategies, and Boolean formula manipulation. Master’s thesis, University of New Mexico
- Bailly C (2003) Automata: the golden age, 1848–1914. Robert Hale, London
- Barish RD, Rothemund PWK, Winfree E (2005) Two computational primitives for algorithmic self-assembly: copying and counting. *Nano Lett* 5(12):2586–2592
- Benenson Y, Paz-Elizur T, Adar R, Keinan E, Livneh Z, Shapiro E (2001) Programmable and autonomous computing machine made of biomolecules. *Nature* 414:430–434
- Benenson Y, Adar R, Paz-Elizur T, Livneh Z, Shapiro E (2003) DNA molecule provides a computing machine with both data and fuel. *Proc Natl Acad Sci USA (PNAS)* 100(5):2191–2196
- Benenson Y, Gil B, Ben-Dor U, Adar R, Shapiro E (2004) An autonomous molecular computer for logical control of gene expression. *Nature* 429:423–429
- Breaker RR, Joyce GF (1995) A DNA enzyme with Mg<sup>2+</sup>-dependent RNA phosphoesterase activity. *Chem Biol* 2:655–660
- Collier CP, Wong EW, Belohradský M, Raymo FM, Stoddart JF, Kuekes PJ, Williams RS, Heath JR (1999) Electronically configurable molecular-based logic gates. *Science* 285:391–394
- Credi A, Balzani V, Langford SJ, Stoddart JF (1997) Logic operations at the molecular level. An XOR gate based on a molecular machine. *J Am Chem Soc* 119(11):2679–2681
- de Silva AP, McClenaghan ND (2000) Proof-of-principle of molecular-scale arithmetic. *J Am Chem Soc* 122(16):3965–3966
- de Silva AP, Gunaratne HQN, McCoy CP (1993) A molecular photoionic AND gate based on fluorescent signalling. *Nature* 364:42–44
- de Silva AP, Gunaratne HQN, McCoy CP (1997) Molecular photoionic AND logic gates with bright fluorescence and “off-on” digital action. *J Am Chem Soc* 119(33):7891–7892
- de Silva AP, Dixon IM, Gunaratne HQN, Gunnlaugsson T, Maxwell PRS, Rice TE (1999) Integration of logic functions and sequential operation of gates at the molecular-scale. *J Am Chem Soc* 121(6):1393–1394
- de Solla Price DJ (1964) Automata and the origins of mechanism and mechanistic philosophy. *Technol Cult* 5(1):9–23
- Ellenbogen JC, Love JC (2000) Architectures for molecular electronic computers: 1. Logic structures and an adder built from molecular electronic diodes. *Proc IEEE* 88(3):386–426
- Fu TJ, Seeman NC (1993) DNA double-crossover molecules. *Biochemistry* 32:3211–3220
- Garzon M, Gao Y, Rose JA, Murphy RC, Deaton RJ, Franceschetti DR, Stevens SE Jr (1998) In vitro implementation of finite-state machines. In: *Proceedings 2nd*

- international workshop on implementing automata WIA'97. Lecture notes in computer science, vol 1436. Springer, London, pp 56–74
- Gordon JM, Goldman AM, Maps J, Costello D, Tiberio R, Whitehead B (1986) Superconducting-normal phase boundary of a fractal network in a magnetic field. *Phys Rev Lett* 56(21):2280–2283
- Holter NS, Lakhtakia A, Varadan VK, Varadan VV, Messier R (1986) On a new class of planar fractals: the Pascal-Sierpinski gaskets. *J Phys A Math Gen* 19:1753–1759
- Huang Y, Duan X, Cui Y, Lauhon LJ, Kim KH, Lieber CM (2001) Logic gates and computation from assembled nanowire building blocks. *Science* 294(9):1313–1317
- LaBean TH, Yan H, Kopatsch J, Liu F, Winfree E, Reif JH, Seeman NC (2000) Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *J Am Chem Soc* 122:1848–1860
- Lederman H, Macdonald J, Stefanovic D, Stojanovic MN (2006) Deoxyribozyme-based three-input logic gates and construction of a molecular full adder. *Biochemistry* 45(4):1194–1199
- Macdonald J (2007) DNA-based calculators with 7-segment displays. In: The 13th international meeting on dNA computing, Memphis, 2007
- Macdonald J, Li Y, Sutovic M, Lederman H, Pendri K, Lu W, Andrews BL, Stefanovic D, Stojanovic MN (2006) Medium scale integration of molecular logic gates in an automaton. *Nano Lett* 6(11):2598–2603
- Mao C (2004) The emergence of complexity: lessons from DNA. *PLoS Biol* 2(12):2036–2038
- Mao C, LaBean TH, Reif JH, Seeman NC (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature* 407:493–496, erratum, *Nature* 408 (2000), p 750
- Mealy GH (1955) A method for synthesizing sequential circuits. *Bell Syst Tech J* 34:1045–1079
- Peppé R (2002) Automata and mechanical toys. Crowood Press, Ramsbury
- Pickover CA (1990) On the aesthetics of Sierpinski gaskets formed from large Pascal's triangles. *Leonardo* 23(4):411–417
- Riskin J (2003) The defecating duck, or, the ambiguous origins of artificial life. *Crit Inq* 29(4):599–633
- Rothenmund PWK (1996) A DNA and restriction enzyme implementation of Turing machines. In: Lipton RJ, Baum EB (eds) DNA based computers. American Mathematical Society, Providence, pp 75–120
- Rothenmund PWK, Winfree E (2000) The program-size complexity of self-assembled squares. In: STOC'00: the 32nd annual ACM symposium on theory of computing. Association for Computing Machinery, Portland, pp 459–468
- Rothenmund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biol* 2(12):2041–2053
- Santoro SW, Joyce GF (1997) A general purpose RNA-cleaving DNA enzyme. *Proc Natl Acad Sci USA (PNAS)* 94:4262–4266
- Shapiro E, Karunaratne KSG (2001) Method and system of computing similar to a Turing machine. US Patent 6,266,569 B1
- Soloveichik D, Winfree E (2005) The computational power of Benenson automata. *Theor Comput Sci* 344:279–297
- Soreni M, Yogev S, Kossoy E, Shoham Y, Keinan E (2005) Parallel biomolecular computation on surfaces with advanced finite automata. *J Am Chem Soc* 127(11):3935–3943
- Stojanovic MN, Kolpashchikov D (2004) Modular aptameric sensors. *J Am Chem Soc* 126(30):9266–9270
- Stojanovic MN, Stefanovic D (2003a) Deoxyribozyme-based half adder. *J Am Chem Soc* 125(22):6673–6676
- Stojanovic MN, Stefanovic D (2003b) A deoxyribozyme-based molecular automaton. *Nat Biotechnol* 21(9):1069–1074
- Stojanovic MN, de Prada P, Landry DW (2001) Catalytic molecular beacons. *Chem Bio Chem* 2(6):411–415
- Stojanovic MN, Mitchell TE, Stefanovic D (2002) Deoxyribozyme-based logic gates. *J Am Chem Soc* 124(14):3555–3561
- Stojanovic MN, Semova S, Kolpashchikov D, Morgan C, Stefanovic D (2005) Deoxyribozyme-based ligase logic gates and their initial circuits. *J Am Chem Soc* 127(19):6914–6915
- Wang H (1963) Dominoes and the AEA case of the decision problem. In: Fox J (ed) Mathematical theory of automata. Polytechnic Press, New York, pp 23–55
- Winfree E (1996) On the computational power of DNA annealing and ligation. In: Lipton RJ, Baum EB (eds) DNA based computers. American Mathematical Society, Providence, pp 199–221
- Winfree E (2006) Self-healing tile sets. In: Chen J, Jonoska N, Rozenberg G (eds) Natural computing. Springer, Berlin, pp 55–78
- Winfree E, Yang X, Seeman NC (1999) Universal computation via self-assembly of DNA: Some theory and experiments. In: Landweber LF, Baum EB (eds) DNA based computers II, DIMACS workshop 1996 (Princeton University: Princeton, NJ), American Mathematical Society, Princeton. Series in discrete mathematics and theoretical computer science, vol 44, pp 191–213; Errata: <http://www.dna.caltech.edu/Papers/self-assem.errata>
- Yurke B, Mills AP Jr, Cheng SL (1999) DNA implementation of addition in which the input strands are separate from the operator strands. *Biosystems* 52(1–3):165–174

## Books and Reviews

- Chen J, Jonoska N, Rozenberg G (eds) (2006) Natural computing. Springer, Berlin
- Lewis HR, Papadimitriou CH (1981) Elements of the theory of computation. Prentice-Hall, Englewood Cliffs
- Seeman NC (2002) It started with Watson and Crick, but it sure didn't end there: pitfalls and possibilities beyond the classic double helix. *Nat Comput Int J* 1(1):53–84



---

## Nanocomputers

Ferdinand Peper

National Institute of Information and Communications Technology, Kobe, Japan

### Article Outline

#### Glossary

Definition of the Subject

Introduction

Wires and Devices

Nanofabrication Techniques and Architectures

Heat Dissipation

Fault-Tolerance

Cellular Automaton-Based Architectures

Crossbar Array-Based Architectures

Neural Network-Based Architectures

Future Directions

Bibliography

#### Glossary

**Adiabatic switching** Switching with asymptotically zero speed with the aim of reducing power consumption in a circuit.

**Asynchronous circuit** Circuit that is designed to work in the absence of a clock.

**Babbage engine** Mechanical calculator built by Charles Babbage in the early nineteenth century.

**Bottom-up fabrication** Fabrication method employing the natural ability of physical structures (including atoms and molecules) to organize themselves into desired structures.

**Brownian motion** Random movement of micrometer-sized particles due to collisions with molecules. The term is also used to indicate random movement of smaller-sized particles, or the mathematical model of such movements.

**Carbon nanotube** Nanometer-scale tube consisting of a graphite sheet rolled up into a

seamless cylinder. Carbon nanotubes intended for nanoelectronic applications are mainly single-walled.

**Cellular automaton** Discrete regular array of cells, each of which is in one of a finite number of states. A cell is updated in discrete time steps according to a transition rule that takes the states of the cell and its direct neighbors and determines the next state of the cell. If all cells are updated at the same time, the model is called *synchronous*, otherwise *asynchronous*.

**Complementary metal-oxide-semiconductor (CMOS)** Currently dominant technology used to implement digital logic circuits as well as a wide variety of analog circuits such as image sensors and data convertors.

**Computational universality** Ability of a computing system to compute every function in a certain class of systems.

**Computer architecture** Functional and structural design and operational specification of a computer system.

**Construction universality** Ability of a logic system to construct every arbitrary logic structure in a certain class of systems.

**Crossbar array** Array consisting of two layers of wires, whereby wires within one layer are parallel to each other but perpendicular to the wires in the other layer. The wires are connected to each other at their crosspoints through devices.

**Coulomb blockade** Increased resistance at certain voltages to the flow of electrons in a tunnel-junction, which is a thin insulating barrier between two conducting electrodes.

**Defect-tolerant** Ability of a system to remain relatively unaffected by the occurrence of permanent defects.

**Delay-insensitive circuit** Asynchronous circuit in which the outcomes of operations are unaffected by delays in wires and functional elements.

**Device** Functional element in a circuit that takes input signals and produces output signals. A transistor is a well-known example of a device.

**Dielectric material** Medium that is a poor conductor of electricity, but an effective supporter of electric fields. A dielectric with a high dielectric constant  $\kappa$  is the preferred material used as a gate-dielectric in a transistor, since it allows for a thicker insulating layer between gate and channel given a certain gate capacitance. A low- $\kappa$  dielectric is the preferred material used for insulating layers between wires, since it allows for smaller wire pitches.

**Entropy** Measure of uncertainty associated with a certain information-theoretic or physical variable.

**Error correcting code (ECC)** Encoding that adds redundancy to information to increase the ability to correct or detect errors caused by noise or other impairments.

**Fault-tolerant** Ability of a system to remain relatively unaffected by the occurrence of errors.

**Field-effect transistor (FET)** Transistor in which the conductivity of the channel depends on the electric field controlled by the transistor's gate.

**Fine-grained parallelism** Scheme for the subdivision of tasks in a large number of small sub-tasks that can be simultaneously executed by a large number of simple information processing elements.

**Finite automaton** Logical scheme consisting of a finite number of states, including a start state and an accept state, as well as a specification of state changes under the influence of inputs.

**Field-programmable gate array (FPGA)** Type of chip containing logic components and programmable wires that can be programmed to perform a wide variety of combinational logic functions.

**Gate** A logic gate is a digital device that carries out a Boolean bit-operation, such as the AND, OR, NOT, etc. A transistor gate is the part of the transistor that controls the conductivity of the channel between the transistor's source and drain.

**Hamiltonian** Property corresponding to the total energy of a system's state that is determined by some sequence of physical operations. A Hamiltonian is mathematically expressed by a unitarian operator  $H$ .

**Heisenberg uncertainty principle** Relationship in quantum mechanics, giving a lower bound on the product of the uncertainties of two

physical observables. These two observables may be position and momentum, or, alternatively, energy and time.

**Lithography** Microfabrication process in which a pattern is transferred to a photosensitive material to selectively remove parts of a thin substrate.

**Majority gate** A logic gate in which the value of the output bit is set to the logic value occurring in the majority of the input bits. A majority gate usually has an odd number of input bits.

**Markov chain** Discrete-time stochastic process of which the next state solely depends on the present state and not directly on previous states. In other words, the process is *memoryless*.

**Molecular electronics** Electronics in which the components (wires and devices) are realized in terms of molecules. These molecules are usually organic.

**Moore's law** Trend along which integration density of microelectronics has developed since the 1960s, the time Gordon E. Moore first observed this trend. Integration density according to this trend increases exponentially, doubling approximately every two years.

**MOSFETS** FETs implemented by CMOS technology.

**Neural network** Mathematical model based on the biological nervous system, consisting of *neurons* that receive (usually) analog values from each other through weighted interconnections. Learning in a neural network takes place through updating the weights based on the values of the neurons and the values of the input signals.

**Neuron** Biological neuron is a cell in the nervous system that processes and transmits information. The central part of a neuron is its *soma* (cell body), and it has an extension called *axon* to transmit information to other neurons via axon terminals called *synapses*. This information is received by a neuron through its *dendrites*. A neuron in an artificial neural network is modeled after a biological neuron. It receives inputs via weighted interconnections, which model the strengths of synapses.

**NP-complete** Class of decision problems for which no polynomial-time (expressed in terms of the input size) algorithms are known.

Any member of the wider class of NP (Non-deterministic Polynomial time) problems can be transformed in an NP-complete problem, in which the time overhead of the translation is at most a polynomial factor.

**Parallel** A computation is parallel if it is divided in smaller computations that can be executed simultaneously.

**Perceptron** A type of neural network model invented in 1957 by Frank Rosenblatt that is used for classification. Though originally this model consisted only of a layer of input neurons and a layer of output neurons, modern uses of the term includes the possible presence of one or more hidden layers of neurons. The perceptron is a *feedforward* neural network, which means that information flows in one direction, from the input to the output, so there are no backward connections between neurons.

**Pitch** Term used in integrated circuits to denote distance between elements, such as between cells in RAM memory or between the centers of two wires. Commonly used is the term *half-pitch*, indicating half this distance.

**Processor** The computing part of a computer also called Central Processing Unit (CPU).

**Quantum dot cellular automaton** Cellular automaton in which cells based on quantum dots containing electrons interact with each other through electrostatic forces.

**Ratchet** Device that allows a process (such as the movement of particles) to take place in only one direction.

**Repeater** Logic device placed on a wire to reproduce signals input to it. Usually implemented in terms of NOT-gates, repeaters speed up the propagation of signals along wires in highly integrated microelectronics.

**Resonant clock** Timing mechanism on synchronous chips that employs the resonance of oscillators to achieve sharply reduced power consumption and increased precision in timing.

**Resonant tunneling device** Device using quantum effects to allow very efficient transmission of electrons through a double barrier tunneling structure.

**Reversible computation** Computation of a function that is one-to-one.

**Scanning tunneling microscope (STM)** Type of electron microscope to view surfaces at the atomic level with resolutions of up to 0.1 nm lateral and 0.01 nm in depth. The STM employs a tip from which electrons tunnel to the surface, whereby the tunnel current depends on the distance between the tip and the surface as an exponential function. The STM can also be used to manipulate individual atoms and molecules.

**Single electron tunneling device** Device based on the tunneling of individual electrons through one or more tunneling barriers, which are thin insulating layers between electrodes.

**Spintronic device** Device based on the magnetic spin states of electrons.

**Superposition of states** Linear combination of states in a quantum system describing a situation in which a physical observable possesses two or more values simultaneously.

**Synapse** Receptor on a neuron's axon that connects to a dendrite of a neuron to transmit information.

**Top-down fabrication** Fabrication method in which structures are formed under the control of a master plan. Optical lithography is the usual top-down method for fabricating micro-electronic chips.

**Tunneling** Quantum mechanical phenomenon in which a particle passes through an energy barrier that would, given the particle's kinetic energy, be too high for it to pass by classical physical laws.

**Tunneling phase logic** Logic that uses the phases of waves to conduct logic operations.

**Turing machine** Abstract logic model consisting of a tape, a reading and writing head, and a finite automaton to control the head. This simple machine model is used to study computation and the relations between computational models.

**Very large scale integration (VLSI)** Microelectronics technology with millions of devices on a chip.

**Voltage encoding** Encoding of the value of a signal by the level of a voltage. This is the most commonly used method to encode signals in electronics. Opposite of *charge encoding*, in

which the value of a signal is determined by the presence of a small number of elementary electrical charges.

**von Neumann neighborhood** Set of cells in a cellular automaton that neighbors orthogonally to a cell. The von Neumann neighborhood of a cell in a 2-dimensional cellular automaton consists of the cell's northern, eastern, southern, and western neighboring cells. Often the cell itself is also included in the definition of neighborhood.

## Definition of the Subject

Nanocomputers are (not-yet-realized) computers that will be based on technology employing devices and wires with feature sizes in the order of a few nanometers ( $10^{-9}$  m). If the increase in integration density of microelectronics according to Moore's law (Moore 2003) continues at the same pace as it has for almost 40 years, such computers will be around in a few decades. Computational power and speed of nanocomputers will likely dwarf those of most contemporary computers if trends from the past continue. It is anticipated that silicon-based CMOS technology can be extended to up to the year 2015; beyond that, major scientific and technological breakthroughs will be required, according to the International Technology Roadmap for Semiconductors (ITRS) (International Roadmap Committee 2005a). Many of these breakthroughs will take place on the physical level, via the development of new devices, but progress on an architectural and algorithmic level will also be indispensable. Among the issues that need to be addressed in this context are the following:

- How to build circuitry with feature sizes well below what can be potentially realized by extensions of traditional manufacturing technologies, like those based on optical lithography, even if taking into account major advances in technology? Will manufacturing based on molecular self-assembly and self-organization be able to deliver such a feat? What will the impact of the adoption of such bottom-up manufacturing have on the architectures of computers built by

it? Will it require structures to be highly regular or random, and how will this affect the way such computers are configured and programmed?

- How to cope with heat dissipation, which will be especially debilitating at the extremely high integration densities expected in nanocomputers? Cooling techniques seem to be insufficient in this context, leaving only the reduction of heat dissipation as an option, which will inevitably involve the reduction of energy consumption. Will it be sufficient to this end to develop new devices, like carbon nanotube-based devices, resonant tunneling devices, single-electron tunneling devices, spintronic devices, molecular devices, etc., or will other innovations be required? Will it be necessary to use asynchronous circuits, i.e., circuits that work without a clock? Alternatively, will it be necessary and possible to use reversible and adiabatic computing schemes (Reversible Computing)? And how will these schemes affect a nanocomputer's architecture?
- How to cope with the unprecedented rates of errors and manufacturing defects that are expected in nanocomputers? In case of manufacturing defects, will it be possible to reconfigure a nanocomputer around such defects, and if so, how to detect the locations of such defects and how to reconfigure circuitry around them? In case of transient errors, which are associated with noise and fluctuations, and which are expected to occur at rates far beyond those experienced in CMOS technology, how to resolve such errors?

Some of above issues have attracted interest in the past, but, ironically, the success of CMOS technology has been a delaying factor in the initiation of systematic studies into them.

## Introduction

In his 1959 lecture entitled *There's plenty of Room at the Bottom* (Feynman 1992), Feynman predicted a future in which a single bit would require a mere 100 atoms to be stored. The same lecture pointed out the possibility of computers with

devices miniaturized to similar scales. Nowadays such machines are called *nano*computers. Feynman's proposal, though visionary, was not made in a vacuum. Molecular conduction, which is of paramount importance to molecular electronics, was investigated by Mulliken and Szent-Gyorgyi in preceding decades (Hush 2003). Furthermore, as early as 1956 Arthur von Hippel advanced his view toward the engineering of materials from atoms and molecules (von Hippel 1956), based on a firm theoretical understanding of material properties. These ideas made their way into the electronics industry through Westinghouse, which started a Molecular Systems Engineering program in 1957 (Choi and Mody 2007). The first conference on Molecular Electronics was held in 1958, jointly organized by the Air Research and Development Command, as well as by the National Security Industrial Association, and attracting approximately 300 participants (Choi and Mody 2007). The interest in military circles for this new field stemmed especially from the need to reduce size, weight, and cost of devices, as well as to increase reliability of electronics, which at the time was far worse than today. Westinghouse's endeavor in molecular electronics received funding from 1959 to 1962 from the US Air Force, but it was discontinued due to a lack of concrete results, especially with respect to manufacturing issues, though it appears to have had a positive spin-off on clean room techniques and scanning electron microscopy (Choi and Mody 2007). The effort disappeared silently from public view in successive years (see p. 181 in Ceruzzi 1998). In retrospect, the idea was far ahead of its time.

A decade later, in the mid-1970s, molecular electronics received renewed attention, when Aviram pursued his radical vision of devices based on individual molecules together with Ratner (Aviram and Ratner 1974). Around the same time, Carter began work on molecular wires, switches, molecular logic elements, and molecular computers (Carter 1983a, b, 1984), thereby becoming one of the pioneers of nanocomputer architecture.

The 1980s saw an intense interest in the use of physics for the implementation of computation (Benioff 1980; Bennett 1988; Fredkin and Toffoli 1982), with special focus on cellular automaton-

like architectures. Drexler, around the same time, started to popularize nanotechnology and computers built by it to a general audience (Drexler 1986, 1992).

In the 1990s, interest widened to a variety of architectures and methods with potential for nanocomputing. Reliability issues of nanocomputers also started to attract attention. Representative of these efforts is the Teramac computer (Heath et al. 1998).

This trend appears to continue in the first decade of the twenty-first century, with unconventional computing models finding an increasing audience.

Though nanocomputers have always been overshadowed by their silicon-based counterparts, they receive attention due to CMOS technology being thought of in danger of running into serious physical limitations. Some predictions for limitations over the years are listed in Table 1 (see Iwai 2004). Though most of the expected limitations in the past have been overcome through a reexamination of the assumptions underlying them, coupled with human ingenuity, this may become difficult as the limitations increasingly assume a fundamental physical nature. Important fundamental physical limits include the following (Lloyd 2000; Meindl et al. 2001):

Thermal limit	Refers to the energy required for a binary switching operation, which should at least exceed the energy of thermal fluctuations, less an error occurs. This amounts to an energy of $(\ln 2)/kTJ$ , whereby $k = 1.38 \cdot 10^{-3} \text{ J K}^{-1}$ is the Boltzmann constant and $T$ the temperature in Kelvin. The probability $P$ of a thermal fluctuation of energy $E_s$ within a response time $\tau$ of the circuit follows the Boltzmann relation (Mead and Conway 1980) $P = \exp[-E_s/kT]$ , which implies a rate of $(1/\tau) \exp[-E_s/kT]$ failures per second per device. The probability due to error can thus be reduced by increasing the signal energy. To be on the safe side, a limit of $100 \text{ kTJ}$ per switching operation is usually assumed. The limit associated with this minimum level of reliability is still four orders of magnitude below the switching energy for 100 nm CMOS technology
---------------	--

(continued)

Quantum limit	Relates the signal switching energy transfer $\Delta E$ to the transition time $t$ via the Heisenberg uncertainty principle, $\Delta E \cdot t \geq h$ , where $h \approx 6.6260 \times 10^{-34}$ is Planck's constant. Resulting from the wave nature of the electron and the associated uncertainty in the energy-time (or the position-momentum) relations, this limit imposes an upper bound on device switching speed given a certain average switching energy. CMOS circuits operate far above the quantum limit	information, these computers have extremely small devices, but they also will likely be slow. Most efforts in this framework find their roots in biochemistry. DNA-computers (Adleman 1994) (DNA Computing) have attracted much interest in this context, though their much-touted use for solving NP-complete problems appears to run into practical limits (Hartmanis 1995)
Speed-of-light limit	Imposes an upper bound on the distance traveled by a signal in one clock cycle. Given that electrical signals travel half the speed of light ( $c \approx 3 \cdot 10^8$ m/s) in typical materials, a distance of merely a few centimeters can be covered within one clock cycle on a 10 GHz chip	Quantum nanocomputers (Quantum Computing) aim to exploit quantum effects such as superpositions of states to achieve vastly improved performance for certain algorithms (see Shor (2004) for an overview of recent progress in quantum algorithms). Though quantum effects increasingly play a role as feature sizes decrease, they have mostly been considered of significance in the context of devices, rather than in a quantum computational framework. Quantum nanocomputers seem not to be within the reach of practical applications before the year 2020 (Ball 2006)

An extensive analysis of limits, ranging from a fundamental level, materials, devices, circuits, up to systems, is given in Meindl et al. (2001).

Nanocomputers come in various flavors and are distinguished according to the underlying mechanisms involved. Montemerlo et al. (1996) distinguish four types:

Electronic nanocomputers	The natural successors of electronic digital computers, these computers represent and process information through the manipulation of electrons. Though conventional transistor technology will be prevalent in the coming decade, alternatives that use tunneling of electrons or their spins have received serious attention, because of their low power potential
Mechanical nanocomputers	These computers are miniaturized equivalents of Babbage engines that conduct their operations through moving molecular-scale rods and rotating molecular-scale wheels, as envisioned in Drexler (1986, 1992). When combined with electrical signals, switches in such computers have excellent On/Off ratios. The high mass of atoms, as compared to electrons, however, gives these computers a speed disadvantage to their electronic counterparts
Chemical nanocomputers	Based on processes involving making and breaking chemical bonds to store and process

(continued)

This chapter focuses mostly on electronic nanocomputers and mechanical nano-computers, these being the most likely successors of current computers. It is organized as follows. After an overview of devices and wires for nanoscale implementations in section “Wires and Devices”, nanofabrication techniques are discussed in section “Nanofabrication Techniques and Architectures”, because they have a direct impact on the architectures of nanocomputers. Top-down techniques, such as optical lithography, face serious limitations in this context. Unsurprisingly there is an increasing interest in bottom-up techniques, which can fabricate structures varying from highly regular on one hand to highly random on the other hand. Heat dissipation, and especially strategies to minimize it, will be the topic of section “Heat Dissipation”. Asynchronous timing and reversible logic receive particular attention, but computation schemes that aim to cope and even to exploit noise are also discussed. Section “Fault-Tolerance” discusses fault-tolerance, distinguishing faults occurring during

**Nanocomputers, Table 1** Past predicted limitations on gate lengths for downscaling of MOSFETs. The listed values are of gate lengths, which are smaller than the generation size listed in the International Technology Roadmap for Semiconductors (ITRS) (International

Period	Expected limit (nm)	Cause
Late 1970s	1000	Short -channel effects, lithography limitations
Early 1980s	500	Source / Drain resistance
1980	250	Tunneling leakage, dopant fluctuation (Mead and Conway 1980)
Late 1980s	100	Red brick wall: various
1999	30	Lower limit to supply voltage (Davari 1999)
2004	50	Red brick wall: various
2004	10	Fundamental

operation – which can be corrected by employing redundancy of some resource in the architecture – and faults of a permanent character, caused during manufacturing – which tend to lend themselves more to reconfiguration techniques. Following these basic topics are three sections describing architectures with particular promise for nanocomputers, i.e., Cellular Automata in section “[Cellular Automaton-Based Architectures](#)”, Crossbar Arrays in section “[Crossbar Array-Based Architectures](#)”, and Neural Networks in section “[Neural Network-Based Architectures](#)”. This chapter finishes with a discussion about future directions.

## Wires and Devices

Three decades of development has left the basic concepts and structures of wires and devices implemented by CMOS relatively unaffected (Wong et al. 1999). This may change, however, as feature sizes decrease further, and technological limits are approached.

The function of an interconnect or wiring system is to distribute clock and other signals and to provide power/ground, to and among, the various circuits/systems functions on a chip (International Roadmap Committee 2005c). The ever-increasing integration densities of chips are accompanied by a quest for increasing speeds of signal transmissions over wires. Unfortunately, the expected delay of a wire of a certain length, which is

Roadmap Committee 2005a). For example, a 30 nm gate length corresponds to the 100 nm technology generation. The term *red brick wall* is named after the color-coded tables in the ITRS report, and indicates a limit beyond which no technology solutions are known (at the time)

proportional to the wire’s resistance as well as to its capacitance, tends to compare unfavorably to gate delays when feature sizes decrease. In practice, the increased delays of wires relative to gate delays under scaling are mostly felt by wires whose lengths cannot be downscaled, due to them running across the chip. Wires that do scale down in length are less sensitive (though not immune) to this problem (Ho et al. 2001). Since most architectures require a substantial level of non-local interconnects, many wires will need to be sufficiently wide to offer the decreased delays associated with lower resistance. This has resulted in designs with an interconnection hierarchy on chip layouts of local thin wires at the bottom, and intermediate wires connecting functional units in the middle, to fat wires for global routing at the top of the hierarchy (Theis 2000). There is much incentive to develop new conductor and dielectric materials, to keep pace with technology requirements. Notable has been the replacement of aluminum by copper for conductors from around the late 1990s, as well as the introduction of low- $\kappa$  dielectrics, which allow decreased pitch (interspacings) between wires.

Whereas transistor delay used to be the bottleneck in past technology generations, it has been superseded by wire delays. The reason for this lies in the increased needs for relatively long wires due to the increased complexities of designs (Ho et al. 2001). The growing gap between transistor delays and wire delays can be addressed through the use in wires of *repeaters* at constant-length intervals to

each other. Though this tends to result in smaller delays, it goes at the expense of increased silicon area and power consumption (Ho et al. 2001). Ultimately the solution lies in shorter wires, and this favors architectures with local interconnection schemes (Beckett and Jennings 2002). Absent the use of such architectures, other radical concepts and solutions to the interconnect problem need to be contemplated. The ITRS of 2005 (International Roadmap Committee 2005c) mentions the following options:

1. Different signaling methods, like *Raised Cosine Signaling* (Bashirullah and Liu 2002), which uses raised cosine pulses instead of square pulses to cope with noise crosstalk problems. Also being investigated is *Resonant Clocking* (O'Mahony et al. 2003b), which uses on-chip inductors to effectuate resonance of clock pulses, and which leads to substantially decreased power dissipation.
2. Innovative design and package options that have optimization of the interconnects as their prime objectives.
3. Three-dimensional interconnects, to decrease the average wire length (Rahman and Reif 2000). It may be a challenge to remove heat from such structures, however; it may also be necessary to use new system architectures and design tools (Srivastava and Banerjee 2004).
4. Different physical principles by which to transfer signals. Among these, optical interconnects (Miller 2000) promise high propagation speeds and bandwidths, high precision clock distribution, absence of crosstalk, and reduced power dissipation. Another possibility is signal transmission by Radio Frequency (RF) microwaves (O et al. 2003), which is accomplished through the placement of small antennas on a chip that are able to receive a global signal, like a clock signal. Finally, guided terahertz waves (Knap et al. 2002) and plasmons (e.g. Weeber et al. 2005), which are hybrids of RF and optical signaling, use transmission frequencies around 1 THz and may result in significantly increased bandwidth.
5. Nanoelectronics-based solutions. Carbon nanotubes (Kreup et al. 2004) are especially

interesting due to their high conductance, and their potential to be used as semiconductors as well. Other solutions may include the use of conducting channels, such as molecular interconnects, and quantum-mechanical interactions, such as spin coupling and tunneling (see Wang et al. 2005) for an overview).

Connecting wires to each other on nanometer-scales is nontrivial due to alignment issues. For this reason, wires are often interconnected through a perpendicular arrangement, like in crossbar arrays, which are discussed in more detail in section “[Crossbar Array-Based Architectures](#)”.

The function of devices is the processing of signals input to them and, in response, outputting signals according to certain specifications. The Field-Effect Transistor (FET) has long been the basis for successive generations of CMOS. The generation commercially available from the end of 2007 – processor chips based on a 45 nm process – will feature high-*K* dielectrics and metal gates (Bohr et al. 2007). This combination allows for a thicker gate insulation layer given a certain gate capacitance, thus substantially reducing leakage through tunneling – the reason for this being that tunneling decreases exponentially with an increasing thickness of the insulation.

Key factors in the success of MOSFETs and circuits based on them have been noise tolerance of digital circuits and a good ability for fan-out due to the high signal gains of transistors (Robert and Keyes 1985), and these factors may also feature prominently in nano-electronics. Moreover, in order to be competitive with CMOS, nano-electronic devices and circuits should satisfy the following requirements (International Roadmap Committee 2005b; Stan et al. 2003):

- Scalable by several orders of magnitude beyond CMOS.
- High information/signal processing capacity. This may be achieved through high switching speeds or a high degree of parallelism.
- Energy dissipation that is much less than in CMOS.
- Room temperature operation.

Nanoelectronic devices also need to satisfy requirements specific to their small feature sizes. Alignment, for example will be much more difficult for three-terminal devices, like transistors, than if only two terminals are involved, like in FETs based on crossed nanowires (Cui and Lieber 2001).

Another important requirement for nanometer-scale devices is that they constitute a *universal set of operators*, which means that every desired circuit can be constructed as a circuit based on these devices. Most designers take it for granted that any arbitrary logic circuit can be constructed from p-type and n-type transistors, but this requirement may become less trivial when alternative devices are used. What then are promising candidates to become successors of MOSFETs?

Carbon Nanotube (CNT) FETs are considered to be the foremost candidate in this context. The channel through which current is rectified by a gate in CNT FETs consists of a single-walled carbon nanotube with semiconducting transport properties (e.g. see Appenzeller et al. 2003). There are still many open issues with CNTs, including the need for a better understanding of the physical mechanisms underlying their operation, problems with the synthesis of CNTs with appropriate characteristics, and fabrication issues like placement, material integration, On/Off ratios, etc. (International Roadmap Committee 2005b).

Resonant tunneling devices (Frazier et al. 1993; Maezawa and Förster 2003) include resonant tunneling diodes (RTD), which have two terminals, and resonant tunneling transistors, with three terminals. An RTD consists of a double barrier structure through which electron transmission is highly efficient about certain resonance energy levels. These devices have novel characteristics, such as negative differential resistance, and they have potentially very high switching speeds and could result in circuits with a reduced number of components and power dissipation. However, they seem to be unable to form a universal set of operators, since no designs are known for memory cells and logic gates only involving RTDs (Nikolic and Forshaw 2003).

Single Electron Tunneling (SET) devices (Uchida 2003) are based on the controlled motion of individual electrons. These devices involve

junctions, through which tunneling takes place, and Coulomb islands that temporarily store electrons and that are usually implemented as quantum dots. SET devices have the potential for high density and power efficiency, while allowing fast switching, but they tend to suffer from low noise immunity as well as from the problem that their gain is insufficient to allow satisfactory fan-out (International Roadmap Committee 2005b).

Molecular devices (Mayor et al. 2003; Petty 2007) utilize single molecules – typically small organic ones – acting as electronic switches and storage elements. Extremely high densities may be achieved with these devices, and they are thought to be very suitable for fabrication by chemical synthesis. Though efficient power usage is expected, these devices may offer switching speeds that are not much higher than those possible by CMOS. Experimental devices have been built, but it has not been made sufficiently clear yet whether the observed device properties are those of the molecules, or rather influenced by those of the used electrodes (International Roadmap Committee 2005b).

Spin logic devices (Wolf et al. 2001), also referred to as magnetoelectronics or spintronics, encode information by the magnetic spin orientations of electrons. Based on the manipulation of the magnetic spin of electrons by magnetic fields or by an applied voltage, these devices may allow very low power dissipation, but unfortunately their projected switching speeds may be limited.

Other devices that may play a future role in nanocomputers are Rapid Single Flux Quantum (RSFQ) devices (Likharev and Semenov 1991), molecular-scale electromechanical devices (Collier et al. 1999, 2000), and quantum interference devices (Debray et al. 1999; Worschech et al. 1999).

## Nanofabrication Techniques and Architectures

There is a close relationship between the technology used to manufacture computers and their architecture. The irregular, aperiodic, structures of current computers are only possible through the use of *top-down* microfabrication technology,

like optical lithography (e.g. see Liebmann 2003; Madou 2002). Notwithstanding the physical limitations faced by lithography, its life-time has been extended over and again, through for example resolution enhancement techniques (Liebmann 2003), which can carry feature sizes below the illumination wavelength. Alternatives to optical lithography are nano-imprinting (Chou et al. 1996), in which a master with nanometer-scale features is made using techniques like electron-beam lithography. This master is then used as a stamp and pressed onto a (soft) target surface that can then be filled in with for example metal to create wires. Expected to allow manufacturing down to 10 nm, these techniques will impose relatively few restrictions on structures, though the number of times a master can be used is limited. As top-down fabrication techniques are being stretched to smaller feature sizes, they will increasingly impose restrictions on layouts, as recent developments involving resolution enhancement techniques show (Liebmann 2003).

There is a lack of consensus as to what is beyond the year 2015 (International Roadmap Committee 2005a) in the context of top-down technologies, though – given the investments made in them – they are unlikely to disappear for the foreseeable future. That said, building the facilities for top-down fabrication will gradually run into economic limitations, as their costs tend to increase by a factor of two for every chip generation – a trend known as *Moore's second law*. Moreover, feature sizes of less than 10 nm may be intractable for top-down mass fabrication, necessitating additional techniques based on different principles. Denoted as *bottom-up*, these techniques exploit interactions between atoms and between molecules to create useful structures. Bottom-up manufacturing is expected to come at the expense of decreased complexity in architectures, because of the lesser degree of control over the manufacturing process. Architectures possible through bottom-up techniques can be divided in three classes: regular, random, and – a combination of these two – quasi-random.

*Regular* architectures are characterized by a repetitive pattern of simple features that cannot be changed after fabrication. Typically, regular

architectures require the use of some form of molecular *self-assembly*, which is defined as the autonomous organization of components into patterns or structures without human intervention (Whitesides and Grzybowski 2002). Self-assembly is related to, but not synonymous with, self-organization, the latter being mainly concerned with pattern formation, and the former mainly involving pre-existing components (separate or distinct parts of a disordered structure) that are designed with a certain outcome of the process in mind (Whitesides and Grzybowski 2002). Self-assembly as a nanofabrication method offers a number of advantages (Parviz et al. 2003):

- It is an inherently parallel process,
- It can generate structures with sub-nanometer precision, and
- It can generate 3-dimensional structures.

When self-assembly's outcome is controlled through external forces or geometrical constraints that are supplementary to the original interactions driving it, it is called *directed self-assembly*. The control allowed by directed self-assembly extends from merely tuning the interactions between individual assembling components to positioning the components at a desired location (Parviz et al. 2003), like the positioning of wires through an electrical field (Smith et al. 2000). Manufacturing by self-assembly requires sufficient control over the process to guarantee regularity. This implies the existence of some mechanism that – though in itself lacking the control of top-down fabrication – can check and correct errors in the fabrication process. The manufacturing of structures based on the self-assembly of DNA tiles (Le et al. 2004; Pinto et al. 2005; Rothemund et al. 2004; Winfree et al. 1998) is an example of this. Such tiles form a skeleton that can be filled in later, using techniques to transport matter towards certain locations in DNA tiles (Sherman and Seeman 2004). Though practical architectures based on such tiles are yet to be realized, computing on such structures in the future can be envisioned. An important question in this context is how complex each tile should be. If its complexity approaches that of a conventional processor, it is

unlikely that it can be manufactured by self-assembly, because of the absence of much regularity, even in the case the processor is not very complex. Self-assembly would require a much simpler tile, with a complexity not exceeding that of a simple finite automaton. This has led various researchers to investigate *Cellular Automata*, which are based on regular (usually 2-dimensional) arrays of tiles (cells), each filled with a finite automaton (Cellular Automata as Models of Parallel Computation). Section “[Cellular Automaton-Based Architectures](#)” describes the use of this model for nanocomputer architectures.

Another regular architecture attracting attention from researchers is the *Crossbar array*. It employs a 2-dimensional mesh of wires that have switches at their crosspoints (see section “[Crossbar Array-Based Architectures](#)”). Regular architectures require post-fabrication configuration to impose a certain desired functionality on them. Configuration may take place, for example, by software initializing cell states of a cellular automaton or setting the switches of a crossbar array. To allow reuse of the architecture for computation, one usually assumes that it can be configured more than once in various configurations. In other words, the architecture should be *reconfigurable*.

At the other end of the spectrum are *random* or *unstructured* architectures. The irregularity of such architectures tends to be mainly caused by the inability of the manufacturing process to perfectly control the formation of structures. The resulting randomness is usually considered part of the architecture to begin with. The elements that are not part of an unstructured architecture will be considered defects, because they are not planned. Lacking a structure imposed on them at manufacturing time, such architectures rely on post-fabrication configuration. To configure a circuit – which may for example be as complex as a processor – on the underlying hardware, it is necessary to have detailed information in advance about the hardware’s structure. This is usually accomplished by an external host computer. In other words, scanning of the underlying hardware is not self-contained in the architecture. This

indicates that random structures will be mostly used in combination with one-time configuration of a circuit, since on-line scanning of the hardware by a host each time that a circuit needs to be mapped may be impractical and too time-consuming. Examples of random architectures are Neural Networks (Türel et al. 2005) and – related to them – the Nanocell (Tour et al. 2002) (see section “[Neural Network-Based Architectures](#)”).

Randomness is hard to avoid in bottom-up fabrication of nanometer-scale features, due to the occurrence of defects. For this reason, perfectly regular architectures may be hard to achieve in nanocomputers. When defects can be dealt with by a one-time configuration of the hardware around them, the term *quasi-regularity* is used. See section “[Fault-Tolerance](#)” for a discussion of defect-tolerance and fault-tolerance.

It is unlikely that one day mankind suddenly wakes up in a world dominated by bottom-up manufacturing. Rather, it will be gradually introduced, incorporated at first in the framework of top-down manufacturing, and increasing its share over the years. This will likely result in architectures in which CMOS is combined with nanoelectronics, like in Chen et al. (2003); Li et al. (2004); Likharev and Strukov (2005); Snider and Williams (2007); Strukov and Likharev (2005); Zhong et al. (2003).

## Heat Dissipation

The huge integration densities facilitated by nanotechnology will have far-reaching consequences for heat dissipation. Being a growing problem in VLSI chips, the heat dissipated by nanoelectronics per unit of area will reach impractical values if techniques, materials, devices, circuits, and architectures continue to be used as they have been in the past. Chips dissipate power proportional to the number of devices  $N$ , to the clock frequency  $f$ , to the probability  $p$  (typically 0.1) that a device is active during a clock cycle, and to the average energy dissipation  $E$  of an active device in one clock cycle (Nikolic and Forshaw 2003; Petty 2007), giving a power dissipation of  $P = NfpE$ . CMOS designs have been by and large governed

**Nanocomputers, Table 2** Technological scaling rules for MOSFETs. From the 1970s on, each new chip generation – introduced at intervals of two to three years – has seen its minimum feature sizes reduced by a factor  $K \approx 0.7$ . The first item *device dimensions* applies to channel length, oxide thickness, gate width, etc. The table assumes unchanged electric fields in devices over successive generations

Device parameter	Scaling factor
Device dimensions	$1/K$
Voltage	$1/K$
Current	$1/K$
Gate capacitance	$1/K$
Delay time	$1/K$
Power dissipation	$1/K^2$

by the scaling rules proposed in Dennard et al. (1974). Table 2 shows some parameters in this context: the power dissipation per device decreases quadratically with downward scaling, and this is indeed what happened in the 30 years since the publication of the table. Coupled with a quadratically increasing number of devices per unit of area, this rule implies a constant power dissipation per unit area over successive generations. There is a limit in this regard, however, because the required downscaling of the power supply voltage causes an increased susceptibility to noise (Kish 2002) and increased off-state drain leakage currents (Meindl 1995). Ultimately the decreased switching energies associated with downscaling run into the thermal limit, discussed in section “Introduction”. To make matters worse, the number of devices  $N$  and the frequency  $f$  will increase as feature sizes move into the range of nanometers. To see where this ultimately leads to, Zhirnov et al. take the physical parameters of an imaginary system to its extreme, resulting in a “limit technology”, in which a chip contains the smallest binary switches, packed to maximum density and operating at the lowest possible energy per bit. This would give a power density in the range of a few  $\text{MW cm}^{-2}$  (Zhirnov et al. 2003), which dwarfs the power density of the sun’s surface by three orders of magnitude. Unfortunately, the capacity to remove heat is limited to several hundred  $\text{W cm}^{-2}$  when known cooling methods for two-dimensional structures are used

(Zhirnov et al. 2003). Additional measures thus seem justified, and this section looks for them in terms of architectures and algorithms.

Goldstein (2005) proposes to reduce heat dissipation problems by spreading circuits out in space, reducing the number of times each part of a circuit is used in a computation. The resulting increase in parallelism allows for higher clock rates. Since the circuitry is spread over a larger area, a lower power density results, but this goes at the cost of longer wires, which causes increased delay and power consumption.

Other approaches seek to reduce heat dissipation in logic circuits by focusing on the clock. Each time clock signals are distributed over a circuit, energy will be pumped into wires, much of which gets lost in the form of heat. A novel approach to clocking problems is to use *resonant clocks*. These designs employ clock distribution circuitry that act as transmission lines with uniformly distributed capacitors and inductors, in which electromagnetic waves propagate long distances without the need for repeaters to amplify signals (Banu and Prodanov 2007; Chan et al. 2005; Mizuno et al. 2000; O’Mahony et al. 2003a; Wood et al. 2001). Since electrical charges on parasitic capacitors can be extracted and preserved by the inductors, this approach allows for a significantly reduced power consumption, but it has problems of its own, like unpredictability due to noise and reflections, difficulties to distribute clock signals in a clock-tree due to impedance mismatches, and the inability to yield constant phase and magnitude of clock signals simultaneously (Banu and Prodanov 2007). To deal with some of these problems, while preserving the important advantage of low-power, Banu and Prodanov (2007) propose to use two identical side-by-side transmission lines with linear topologies in which clock signals travel in opposite directions, allowing the extraction of an absolute phase at any point of the clocking circuitry by averaging the two signals’ timings. The overhead of the averaging circuitry, however, may compromise the efficiency of this solution.

A radical way to deal with clocking problems is to remove the clock all together, resulting in *asynchronous circuits*, which have a range of

advantages. Apart from not having the clock's overhead of increased energy consumption and increased area, such circuits have the fortunate tendency to restrict energy consumption to only those parts that are actively switching. In other words, asynchronous circuits have an inherent mechanism to selectively shut off parts of them – even down to fine granularities of individual devices – a feature for which synchronous circuits require special *clock gating* circuitry.

Apart from the reduced power consumption and heat dissipation offered by asynchronous electronics – provided it is well-designed – there are also other advantages:

- Problems with the distribution of clock signals disappear, like *clock-skew* (timing differences at which different parts of a circuit receive the clock-signal) and *race conditions* (the failure of signals to reach their destinations within a clock cycle). These problems tend to get worse, otherwise, with increasing integration densities.
- Less noise and electromagnetic interference.
- Insensitivity to physical implementations and conditions. This implies more freedom in the timing of signals and layouts of circuits, which is useful in the reconfigurable architectures expected in nanocomputers (Goldstein 2005).
- Average rather than worst-case performance. An asynchronous circuit operates as fast as switching times of devices allow, rather than being limited by the slowest parts of a circuit, which in synchronous systems imposes an upper limit to the global clock rate.
- Modularity. Asynchronous circuits can be designed in terms of modules that can be combined without considerations of timing restrictions. This facilitates flexible rearrangement of circuit elements into various circuits.

Asynchronous circuits come in many flavors (Davis and Nowick 1997; Hauck 1995), which are distinguished by the extent as to which assumptions are made about timing of signals, both of a circuit and in the interaction of a circuit with its environment. The most general class of asynchronous circuits assumes certain bounds on delays of signals (Unger 1969), but unfortunately it is the

least robust to unexpected behaviors, which complicates their design. Forming the second class are *self-timed circuits*, which only make assumptions on timing within circuit elements, like bounded delay etc., but not on timing between modules (Seitz 1980). *Speed-independent circuits*, the third class, assume unbounded delays in circuit elements, but zero or negligible delays on wires between the elements (Muller and Bartky 1959). The fourth and fifth class form the *quasi-delay-insensitive circuits* (Manohar and Martin 1995) and the *delay-insensitive circuits* (van de Snepscheut 1985). The latter are robust to delays in both circuit elements as well as wires, but this is at the cost of significant design complexity. For this reason quasi-delay-insensitive circuits are more common. They differ from delay-insensitive circuits in their requiring a so-called *isochronic fork* (Martin 1990), which is a fanout element of which all output branches have a similar delay. In practice, this class of circuits shares many similarities with speed-independent circuits.

Notwithstanding the advantages of asynchronous circuits, their use has been limited to few applications. This is partly due to the much smaller base of design and testing infrastructure when compared to synchronous technology, though asynchronous design support technology is increasingly being developed (Taubin et al. 2007). One of the key motivations to use synchronous circuits in the first place used to be ease of design, as there is – as compared to asynchronous circuits – less unexpected circuit behavior accompanying changes in the values of signals. The advantages of synchronous design have become less evident today than in the past, as timing problems have become more urgent with the increase of clock frequency, but synchrony still defines the mindset of generations of circuit designers.

The conditions surrounding nanoelectronics may change that. Not only will problems accompanying a clock become more severe with the increase of integration densities, the entire nature of systems will change. Whereas signals in digital logic have been voltage-encoded in virtually all systems, they may be represented in quite different ways on nanometer scales, to better reflect the physical environment. As signals more and more

approach the scales of individual particles, a token-based representation in which signals have a discrete indivisible character may be more appropriate. Circuits based on single electron tunneling (Uchida 2003) are already moving in this direction (Ono et al. 2005), as do experimental circuits based on molecule cascades (Eigler et al. 1819; Heinrich et al. 2002). Asynchronous timing is very suitable for token-based circuits, and may thus play an important role in this context (Peper et al. 2003).

Another approach to minimize heat dissipation aims to minimize the energy required for switching, even eliminate it altogether. This is possible – at least in theory – by making computations reversible (Bennett 1973; Landauer 1961) (Reversible Computing). The key to this line of thought is the observation that the information stored on a computer is ultimately represented in terms of physical entities (Landauer 1992). Operations in a computer that change the entropy of this information will accordingly change the thermodynamical entropy of the physical system. The erasure of a bit by overwriting it with a new value, is such an operation (Landauer 1961). Whereas the bit's entropy corresponds to two possible (unknown) states of the bit before the operation, the bit's value will be fixed into one state by the operation, which decreases the entropy. This implies a decrease of the bit's physical degrees of freedom, and – to compensate for this – there is an increase of unobservable degrees of freedom such as in the microscopic motion of molecules (Lloyd 2000), which usually translates into the dissipation of heat. The energy loss associated with such an operation would be at least  $kT \ln 2$  J, according to the thermal limit in section “Introduction”. While Landauer (1961) initially believed that no universal computations could be performed without a change in entropy, Bennett (1973) proved this belief wrong by constructing a universal Turing-machine that conducts its operations reversibly, implying that it would cause no change in entropy.

This line of thought, which started in the early 1960s, is widely known today, but it has not resulted yet in practical circuits. Arguably, it will be difficult to overcome noise and friction, which

are fundamental to physical processes. Most proponents of reversible computing agree that these factors are an impediment to dissipation-less computing, but they tend to argue that, absent these factors, reversible computing would *in principle* be possible. This would make it a matter of technological progress to approach the zero average switching energy limit arbitrarily close, rather than there being a hard physical lower limit.

One way to go about in this context is to slow down switching speed. Called *Adiabatic Switching* (Athas et al. 1994), this approach decreases the dissipation caused by the charging and discharging of a gate capacitance  $C$  through an associated resistance  $R$ . The dissipated energy for charging such a gate is given by Athas et al. (1994):

$$E = \frac{RC}{\tau} CV^2$$

whereby  $\tau$  is the time required to charge the gate, and  $V$  is the voltage supplied to the circuit. An increase in  $\tau$  thus implies a corresponding decrease in the dissipated energy, the other factors remaining equal. Schemes of adiabatic switching usually follow the requirement that, first, a switch is only closed when there is no voltage difference between its terminals and, second, a switch is only opened when no current flows through it. Adiabatic schemes of switching tend to be presented in combination with a reversible mode of computation (Frank 2005). When reversibility is left out, the resulting adiabatic scheme is usually characterized as *charge recovery* or *energy recycling* (Sathe et al. 2005).

Reversible computing appears to have particular appeal for nanoelectronics, since physical interactions at nanometer scales are typically reversible. Notwithstanding the reversible computing paradigm's wide acceptance, there are still concerns about its feasibility (Cavin et al. 2005; Jablonski 1990; Porod et al. 1984). Concerns based on problems with noise and friction have been pointed out in Porod et al. (1984) in the early years of reversible computing. This is a recurrent theme with skeptics of reversible computation. The overwhelming responses generated

by Porod et al. (1984) are particularly educating Benioff (1984); Bennett (1984); Landauer (1984); Toffoli (1984), (see also the accompanying reporting in Robinson (1984)). Other concerns (Jablonski 1990) highlight some principal reasons why reversible computing would only work under unrealistic conditions, such as the necessity to have an infinite information-theoretic temperature of the scheme. There are also concerns (Cavin et al. 2005) about possible problems when reversible schemes are combined with adiabatic schemes, the first supposedly being thermally isolated from the environment, while the second scheme is usually not. Other fundamental problems are pointed out in Cavin et al. (2005) as well with adiabatic switching: apart from thermal noise interfering with the reliability of a simple adiabatic bit-flipping operation, this noise is also responsible for a less than ideal charging curve of a circuit's capacitances, which causes an increased power consumption. Finally, a problem in Cavin et al. (2005) is claimed with the power supply of adiabatic circuits, which, in order to be the approximate ideal current source needed for adiabatic charging, would require a high internal resistance. The latter would cause much heat dissipation, not in the circuit itself, but in the power supply, so that the overall energy efficiency of the scheme would be inferior to non-adiabatic schemes. A more optimistic note about the prospects of reversible adiabatic computing is found in Frank (2005), though it is admitted that there remain significant challenges, the most fundamental of which are the requirement of switching devices that are less resistive and less leaky, and the requirement of higher quality power supplies and clocks, like the resonant clocks mentioned earlier in this section.

The problems with clocks in the framework of reversible computing have led some researchers to consider whether reversible computing can be combined with asynchronous circuits, especially asynchronous circuits that are token-based and delay-insensitive (Lee et al. 2004, 2006; Morita 2003). Circuits, in which only one single signal token moves around at a time, do not need a clock, so they would qualify as reversible and asynchronous. Though computational universal circuits can be

constructed in this way, they have obviously a problem in terms of efficiency. The alternative is circuits in which multiple tokens move around in parallel. This may lead, however, to situations in which reversibility is hard to define (Lee et al. 2004). A circuit element having two or more tokens as asynchronous inputs for an operation will – in order to preserve a strict one-to-one relationship between input and output as defined through a (unitary) Hamiltonian operator (Feynman 1985) – need to monitor the arrival times of the individual tokens, but this requires a large overhead and it is counterintuitive to how a circuit is supposed to operate, suggesting that the underlying physical model may need reconsideration.

An unconventional strategy to reduce energy consumption is to use switching operations with a reduced reliability. Palem and co-workers (Korkmaz et al. 2006; Palem 2005) assert that by accepting a reduced reliability of switching, less energy will be required to conduct a switching operation. Palem (2005) shows that if a switching operation is correct with a probability  $p(>1/2)$ , then its energy consumption – assuming that the operation is irreversible – is bounded from below by  $kT J \ln(2p)$ . A reduced reliability of switching limits the range of possible algorithms to those that accept some degree of randomness, like probabilistic algorithms (Motwani and Raghavan 1995), image processing algorithms, and classification algorithms (e.g., Fukś 2002). Somewhat related is a scheme proposed by Kish (2006) that drives computations by noise. Shown is a simple model circuit containing a comparator with error rate around 0.5. Inspired by the highly efficient way in which information is processed in the brain, this thermal noise driven computing scheme requires a minimum energy requirement of  $1:1kTJ$  per bit. This scheme may require extensive error correction to make it work. The use of noise and fluctuations as a computational resource is also outlined in Dasmahapatra et al. (2006), as likely being a driving force for mechanisms in biological organisms (Yanagida et al. 2007). Brownian motion provides a free search mechanism, allowing a molecule, signal, or other entity to explore its environment as it randomly moves around, thereby undergoing certain operations (like molecular binding) when it arrives at

certain locations (like molecular binding sites). An early mentioning of Brownian motion for computations is made by Bennett in Bennett (1982), who uses it for reversible computation in case thermal noise cannot be avoided. Bennett considers a Brownian implementation for a mechanical Turing machine, in which signals move around randomly, searching their way through the machine's circuitry, only to be restricted by the circuit's topology. This search would, in principle, not require any energy to be provided from a power supply, the drawback being that the computation would move forward and backward evenly without any bias, and thus would take a long time to complete. Funneling the random motions into preferred directions would require the selective use of ratchets, but ratchets tend to consume energy, as shown by Feynman et al. (2006). A mechanism somewhat similar to a Brownian Turing machine is found in nature, in the form of the Brownian tape-copying machine embodied by RNA polymerase (Bennett 1982). The efficient – in terms of energy consumption – use of Brownian motion in various biological mechanisms, such as those involving Brownian molecular motors, suggest that biological organisms may provide inspiration for future nanocomputing systems in this respect.

Noise and fluctuations have also been used in the simulated annealing process of a Boltzmann machine that can be implemented by SET devices (Yamada et al. 2001) (see also section “[Neural Network-Based Architectures](#)”), though reductions in power consumption have not been the main purpose of this work.

## Fault-Tolerance

When a system experiences a failure in one of its parts and can continue to operate, possibly at a reduced level, rather than – as most systems would do – fail completely, it is called *fault-tolerant*. A fault-tolerant system may experience a reduced throughput or an increased response time in the face of one (or more) of its components failing. Fault-tolerance is an important property for nanocomputers, since the reliability of their components will likely be low due to:

- The probabilistic nature of interactions on nanometer scales, rooted in thermodynamics, quantum mechanics, etc.
- The individual behavior of particles, molecules, etc. gaining importance over their averaged behavior due to their decreased numbers, causing the “Law of Large Numbers” to become invalid below the device level (Birge et al. 1991; DeHon 2004).

When faults have a permanent character – either incurred at manufacturing time or later – they are usually referred to as *defects*. A nanocomputer architecture that can cope with defects is called *defect-tolerant*.

When faults have a transient character, i.e., when they occur during operation and are temporary, they are referred to as *transient faults*, or, simply, *faults*, like the more general term. They are mainly caused by environmental conditions, such as thermal noise, quantum fluctuations, and electromagnetic perturbations. Cosmic radiation, which is a major source of transient faults in CMOS technology, may be less of a concern, on the other hand, for nanoelectronics based on materials different from semiconductor silicon (Gil et al. 2007). The term used to describe robustness of an architecture to the occurrence of transient faults is *fault-tolerance*, like the more general term.

Faults may also be *intermittent*, meaning that they occur during a certain (extended) time interval, then disappear, and may (or may not) later reappear. Intermittent faults tend to be caused by unstable or marginal hardware, due to for example manufacturing residues and process variations. These problems tend to increase with the increase in integration density (Constantinescu 2007). Changes in operational parameters, such as temperature, voltage, and frequency changes may trigger intermittent faults (Constantinescu 2007). Though transient faults and intermittent faults have many characteristics in common, the latter differ in that they tend to occur in bursts, at the same location. Repairing a part affected by intermittent faults usually eliminates them, whereas transient faults cannot be prevented by repairs.

Tolerance to faults and defects can be accomplished at various levels of abstraction (Graham

and Gokhale 2004): the *physical device level*, the *architectural level*, and the *application level*. An example relating to the physical device level is the tolerance of digital devices to noise due to the digital encoding of signals. Tolerance at the architectural level compensates for malfunctioning of devices by organizing them in structures with a certain level of *redundancy*. Such a structure may consist, for example, of replicated (identical) components, operating in parallel, on which majority voting is used to extract the result of an operation. Tolerance at the application level refers to the ability of a computing application to operate correctly despite defects and faults in the underlying hardware. Research aiming to improve the tolerance at the application level may be found in the framework of probabilistic algorithms or image processing algorithms that tolerate some noise in their input. As tolerance at the physical device level tends to decrease with increasing integration density, the tolerance at the architectural and application levels need to improve to compensate for this. It also works the other way around: it may make less sense, for example, to improve tolerance at the physical device level, if the application level is able to cope with a certain level of faults or defects on its own. Most research efforts on fault-tolerance in the nanocomputing field are in practice directed to the architectural level, and this section is no exception on that.

Redundancy in an architecture is achieved by equipping it with additional resources (Lala 2001). Especially important are the following types of redundancy:

- *Information redundancy* refers to the addition of information to the original data such as to be able to recover it in case of faults. *Error Correcting Codes (ECC)* are the usual form to represent redundant information (e.g. Mac Williams and Sloane 1978). Used extensively to achieve reliable operation in communications and memory, error correcting codes provide a way to cope with the corruption of bits by encoding messages as code words that contain redundant information. This information is used to reconstruct the original code word in case errors occur.

- *Hardware redundancy* refers to the inclusion of additional hardware to the system. This may encompass the use of extra components working in parallel to mask any faults that occur – a technique called *static redundancy*, which includes the use of ECC. When only one element of the extra components are used at a time, to be switched out and replaced by a spare in case of a fault, the term *dynamic redundancy* is used. Dynamic redundancy is usually achieved through periodic testing of circuits or self-checking of circuits, followed up by the *reconfiguration* of a circuit, or even its *self-repair*.
- *Time redundancy* involves the repetition of operations in a system. This may take place after an error is detected, in which case a recovery procedure needs to be started up. Alternatively, as a standard procedure operations may be repeated a fixed number of times, after which the consecutive results are compared to each other. Time redundancy is only suitable to cope with non-permanent faults, since repetitions of operations by a defective component will only repeat incorrect results.

The different nature of defects as opposed to faults is reflected in the different strategies usually followed to make an architecture robust to them, though there tends to be an overlap.

The permanent nature of a defect implies that it needs to somehow be isolated from non-defective parts to prevent it from affecting a system's correct functioning. Strategies to cope with defects thus tend to first detect defects and then reconfigure the system around them. This requires spare components that become activated due to the reconfiguration, while the defective components (or defective connections) are made inaccessible. Absent reconfiguration, a system's performance will be adversely affected by a defect, in terms of throughput, speed, reliability, etc., but this may be ameliorated to an acceptable level if sufficient spare resources are available.

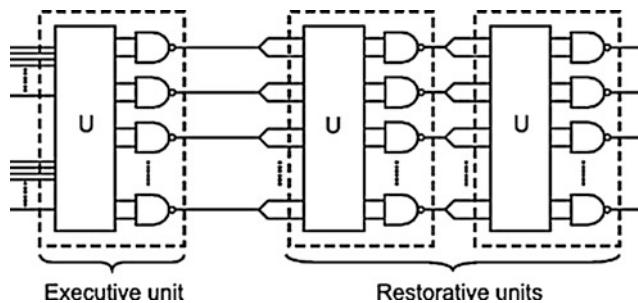
Depending on whether reconfiguration takes place at the same time that a nanocomputer is processing data or not, the terms *on-line* or *off-line* reconfiguration, respectively, are used. On-line reconfiguration implies a continuous monitoring of

a computer for defects and accordingly taking action if they are detected. Usually envisioned as being self-contained in the computer, this type of operation needs the detection of defects and the associated reconfiguration to be robust to defects and faults themselves, which complicates its realization. Off-line reconfiguration is usually done under the direction of a host-computer, which creates a defect-table and reconfigures the system accordingly. While easier to realize than on-line configuration, it may be only feasible if the size of a nanocomputer is limited, since otherwise the detection and reconfiguration operations would consume prohibitive amounts of time. A well-known example of off-line configuration of a prototype defect-tolerant nanocomputer is the Teramac (Heath et al. 1998). Consisting of  $10^6$  gates operating at 1 MHz, it is based on Field Programmable Gate Arrays (FPGAs), 75 percent of which contain at least one defect. Having a total of three percent defects out of a total of 7,670,000 resources including wiring and gates, the Teramac needs a rich interconnection structure to find alternative routes around defects. To this end, it employs a crossbar array, which is described in more detail in section “[Crossbar Array-Based Architectures](#)”.

Fault-tolerance requires a different strategy than defect-tolerance, due to the temporary nature of faults. Such a strategy usually involves a combination of information redundancy and hardware redundancy.

Motivated by the low reliability of electronic components in his time, von Neumann studied how reliable computation could be obtained

through the use of unreliable components (von Neumann 1956). His *multiplexing* technique employs a circuit built from faulty universal gates – all of the same type – to simulate a single gate of this type, but with increased reliability. Each (input- or output-) wire of the gate is replaced by a bundle of  $N$  wires in the circuit. The circuits investigated by von Neumann are based on the NAND-gate, but other gates could also be used in principle. The logic state 1 or 0 of a bundle of wires is determined by whether the fraction of wires in state 1 – called its *excitation level* – lies above resp. below a certain threshold. The scheme consists of three stages, of which the first – the *Executive Unit* – conducts the logic operation, and the last two – the *Restorative Units* – act as nonlinear amplifiers that restore the excitation level of the output bundle to its nominal level (see Fig. 1). The resulting circuit has two input bundles of wires and one output bundle of wires. Wires from the first input bundle are randomly paired by a permutation network labeled  $U$  with the wires from the second bundle; the resulting pairs of wires are then used as inputs to the faulty NAND-gates in the Executive Unit. The wires in the output bundle are split, and again randomly paired to form the inputs to the faulty NAND-gates in the first Restorative Unit. This is repeated for the second Restorative Unit. This scheme suffers from a limited reliability, while requiring a high redundancy. For example, an error rate of 0.005 per device requires a redundancy of 1000 to achieve a probability of failure of



**Nanocomputers, Fig. 1** Von Neumann’s fault-tolerant circuit that is used to replace a single faulty NAND-gate. It consists of three stages: The Executive Unit conducts the actual logic operation, whereas the two later Restorative

Units reduce the fault-rate of the Executive Unit’s output. The bundle of output wires of the circuit will effectively be reduced to one wire (not shown here), of which the value is determined by the excitation level of the bundle

the gate of merely 0.027. A similar scheme by von Neumann that uses three-input *majority gates*, instead of NAND-gates, does not fare better.

A drawback of von Neumann's scheme is that it entirely focuses on the error rate of gates, but not on that of wires. While the latter can be modeled in terms of the former if wires have a constant length, this condition will not be valid in densely connected circuits, causing failure rates of gates to depend on the length of their input wires (Pippenger 1990). Von Neumann's scheme is at the root of many follow-up schemes proposed in the decades thereafter. Most of these efforts tend to minimize the number of required gates, given a certain failure probability of the components. Dobrushin and Ortyukov, for example, show in Dobrushin and Ortyukov (1977) that a function computed by  $m$  fault-free elements can be reliably computed by  $O(m \log m)$  unreliable elements with a high probability. This is further improved in Pippenger (1985, 1989) to  $O(m)$  unreliable elements. An excellent review of these models is found in Pippenger (1990). This review also shows some variations on this line of research, in which it is not gates that may fail, but contacts of switches. Thought to have become irrelevant with relays becoming obsolete for computational purposes, these variations regain significance in the context of nanocomputing, given that nanoscale switching devices with mechanical contacts have attracted renewed interest (Collier et al. 1999, 2000).

Another method, *R-fold modular redundancy* (Depledge 1981), achieves tolerance to faults by employing  $R$  copies of a unit ( $R$  preferably an odd number), of which the majority establishes the output in accordance with an operation conducted by a majority gate. If  $R = 3$ , this technique is called *Triple Modular Redundancy* (TMR) (Depledge 1981). When the outputs of three TMR units are combined by a majority gate on a second level and so on in a hierarchy of levels, the result is a model with increased reliability higher in the hierarchy: *Cascaded Triple Modular Redundancy* (CTMR) (Spagocci and Fountain 1999). TMR, CTMR, and von Neumann's multiplexing technique lead to high redundancies, as compared to reconfiguration techniques (Nikolic

et al. 2002), but on the other hand they are more suitable for transient faults. Von Neumann's original multiplexing technique is reconsidered in Sadek et al. (2004), with claims that its redundancy can be reduced by using output wire bundles as input bundles to other gates, rather than – as von Neumann advocated – reducing them to a single wire between the outputs bundles of a gate-equivalent unit and the inputs bundles of the next unit. Von Neumann's multiplexing technique is applied in a hierarchical way in combination with reconfiguration techniques in Han and Jonker (2003). The claim is of a significantly reduced redundancy in this design, even when the device error rate is up to 0.01. The usefulness of Markov chains in the analysis of these models is reviewed in Han et al. (2005).

The constructions of von Neumann and others have some resemblances with error correcting codes, especially to so-called *repetition codes*, which are codes (but not very efficient ones at that) in which each symbol of a message is repeated a number of times to create redundancy. The use of error correcting codes in this context leads to a scheme in which computation takes place in the encoded space, whereby errors are corrected locally, and encoding and decoding is only necessary at the beginning and the end, respectively, of the computation. This line of thought is followed in Spielman (1996), but then through the use of *generalized Reed-Solomon codes* (e.g., Mac Williams and Sloane 1978), rather than repetition codes, to realize a fault-tolerant computing model with improved reliability.

Fault-tolerance in architectures based on cellular automata and on crossbar arrays are discussed in the sections describing these architectures, i.e. in sections “[Cellular Automaton-Based Architectures](#)” and “[Crossbar Array-Based Architectures](#)”, respectively.

## Cellular Automaton-Based Architectures

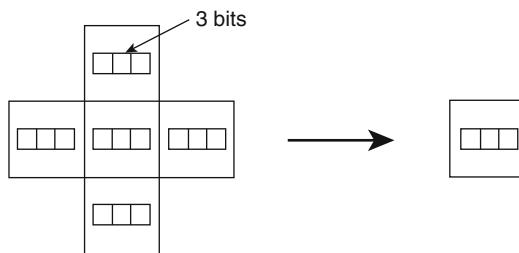
A Cellular Automaton (Cellular Automata as Models of Parallel Computation) consist of cells that are organized in a regular array. Each cell

contains a finite automaton that, taking as input the state of the cell and the states of its neighboring cells, changes its state according to a set of transition rules. These rules are designed such as to produce certain global behavior. When cellular automata are used as architectures for nanocomputers, this behavior is usually general-purpose computation, but application-specific behavior, like pattern recognition tasks, has also been investigated.

Originating in von Neumann's research on self-reproduction in the 1940s, cellular automata have experienced a resurgence in interest in recent years, especially in the context of nano computing architectures. Their simplicity – evidenced by a tiling-based structure in combination with local neighborhoods of cells with finite automata-like complexity – has much potential for bottom-up fabrication by molecular self-assembly. Moreover, their local interconnection structure implies a constant length of wires, which makes them efficiently scalable to high integration densities (Beckett and Jennings 2002). Cellular automata are unlikely to be used in combination with top-down fabrication methods, because of their significant overhead. Apart from the low percentage of cells that typically participate simultaneously in a computation, there may also be – depending on the design – relatively high hardware requirements for each cell, given its limited functionality as compared to conventional VLSI circuit designs. One of these factors may be improved, but this tends to be at the cost of efficiency with respect to the other factor: increasing the functionality of each cell, may increase the percentage of them that engage in a computation, since configurations of cells implementing certain tasks can be smaller, but on the other hand a cell will require more hardware resources to implement its functionality. A cellular automaton's overhead, which may exceed a factor of 10, could still be acceptable if it can be compensated for by the low-cost availability of cells in the huge quantities that can be expected with bottom-up fabrication. Even when only bottom-up fabrication methods are available, however, it is important to minimize the complexity of cells, because success in this area translates directly into more efficient physical realizations.

Though traditionally the complexity of a cellular automaton's cells is measured in terms of the number of cell states, this is inadequate for a nanocomputing framework, as it leaves unmentioned a cell's functionality in terms of the (number of) transition rules. The centralized storage of the table of transition rules, which is common in traditional cellular automata or software to simulate them, is infeasible for nanocomputers, because it violates one of the key tenets of cellular automata, i.e., locality. That leaves only storage of the rule table in each cell individually as a viable option, implying that this table should be as small as possible, less a huge overhead occurs. The alternative to storage of the rule table is finding an efficient representation through inherently available physical interactions implied by the physical structure of a cell, and this is ultimately what many researchers aim for when using cellular automata for nanocomputers. This alternative, however, imposes even stricter limitations on transition rules.

A cell's complexity is determined by the information required to represent its functionality, and this includes the cell's state as well as the transition rules or any equivalent physical structure or mechanism. There are various ways to represent this information. A representation requiring a decreased number of bits for storage of transition rules may require increased resources, both in time and hardware, to decode the information. This implies that it is important to consider the overall costs, including the required supporting circuitry, to be taken into account when evaluating a cell's complexity. Absent a concrete physical implementation, it tends to be easiest to settle for a cost measurement in terms of the number of bits to represent the cell states plus the number of bits to represent the transition rules, under the assumption that decoding of this information is done through a straightforward and simple scheme. For the cellular automaton in Lee et al. (2003), for example, three bits are required to represent a cell's five states. The von Neumann neighborhood of a cell in this model requires each transition rule to encode in its left-hand side the states of a cell itself and the states of four neighboring cells, and in the right-hand side the new state of the



**Nanocomputers, Fig. 2** Bits required to encode a transition rule in a cellular automaton with 5-state cells and von Neumann neighborhood. The left-hand side of a transition rule requires three bits for the cell state itself and three bits each for the four neighboring cells' states. The right-hand side requires three bits to represent the new state of the cell after the transition

cell (see Fig. 2). Given that there are 58 transition rules in this model, this results in a total of  $3 + 58 \times 3 \times (1 + 4 + 1) = 957$  bits that are required to encode the functionality in a cell. A similar calculation yields 148 bits of information for the model in Peper et al. (2003) and 100 bits of information for the model in Peper et al. (2004). A further reduction may be required to allow feasible physical realizations.

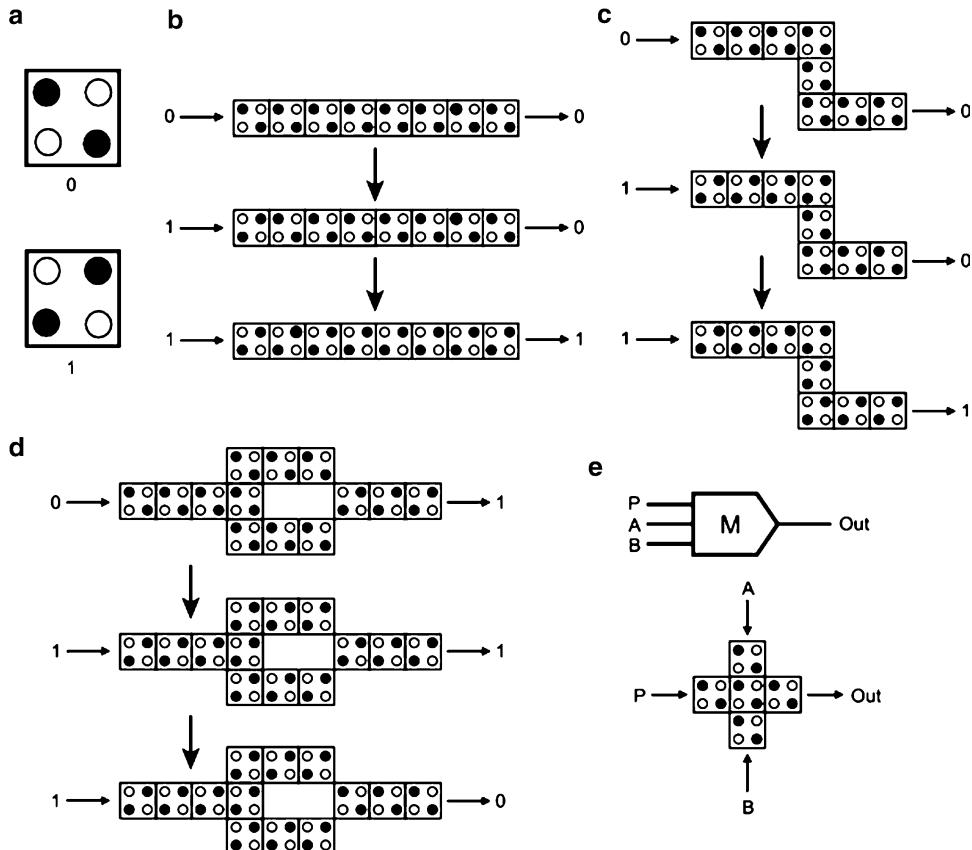
The use of cellular automata for nanocomputers originates in the late 1970s work of Fredkin, Toffoli, and Margolus, who aimed for computation inspired by physical models. A major issue in this research is the reduction of energy consumption as a road to high-performance high-integration computers. Signals in this paradigm have a discrete particle-like character and are conserved throughout operations. Moreover, a reversible scheme of logic is adopted.

One of the first proposals for a (2-dimensional) cellular automaton based on molecular interactions is due to Carter (1984). Written from a chemist's perspective, this proposal focuses on how particular operations of cells -including propagation in wires and switching by various automata – could be realized by means of the cascaded exchange of single and double bonds in molecules. This bond exchange – termed *soliton propagation in conjugated systems* – is relatively slow, but due to the small distances at which switches can be placed (in the order of 20 nm),

switching times in the subnanosecond time scale are claimed to be feasible. Follow-up on this work is needed to show how the proposed operations could be combined into a cellular automaton capable of useful computations, but unfortunately the work has remained relatively unknown in the computer science community.

The Quantum Dot Cellular Automaton (QCA) (Lent et al. 1993; Porod 1998; Tougaw and Lent 1994) has cells that consist of four quantum dots, two of which contain an electron each (Fig. 3a). Due to Coulomb interactions inside a cell, the electrons will settle along one of two possible polarizations, interpreted as signals 0 and 1, respectively. On a larger scale, there are also Coulomb interactions between cells through the electrons in them. These interactions can be used for the propagation of signals along cells (Fig. 3b, c). Logic operations are also possible, like the NOT-gate in Fig. 3d, or the majority gate in Fig. 3e, which covers both an AND gate and an OR gate in its functionality. The QCA model is not a cellular automaton in the true sense, since it assumes a layout of cells along the topology of the circuitry they represent. This may even include layouts in which a cell is placed with an offset of half a cell at the side of another cell (Tougaw and Lent 1994). QCA promise extremely low power consumption as their operation does not involve flows of electrons, but rather tunneling of electrons on a local scale, i.e., between quantum dots within a cell. The problem with QCA, however, is that they tend to be very sensitive to noise, imposing a low operating temperature on them, which is an obstacle to their efficient application. A supposedly more practical variation on QCA is the Magnetic QCA (Cowburn and Welland 2000), in which the dots are implemented as submicron magnetic dots and which may allow operating temperatures as high as room temperature. There is considerable doubt among some researchers (Brillouët 2007) whether QCA will ever provide a significant competitive advantage with respect to CMOS in terms of integration density, speed, power consumption, and the implementation of complex systems.

A 2-dimensional cellular automaton using optical signals to trigger transitions has been proposed by Biafore (1994). Employing physical

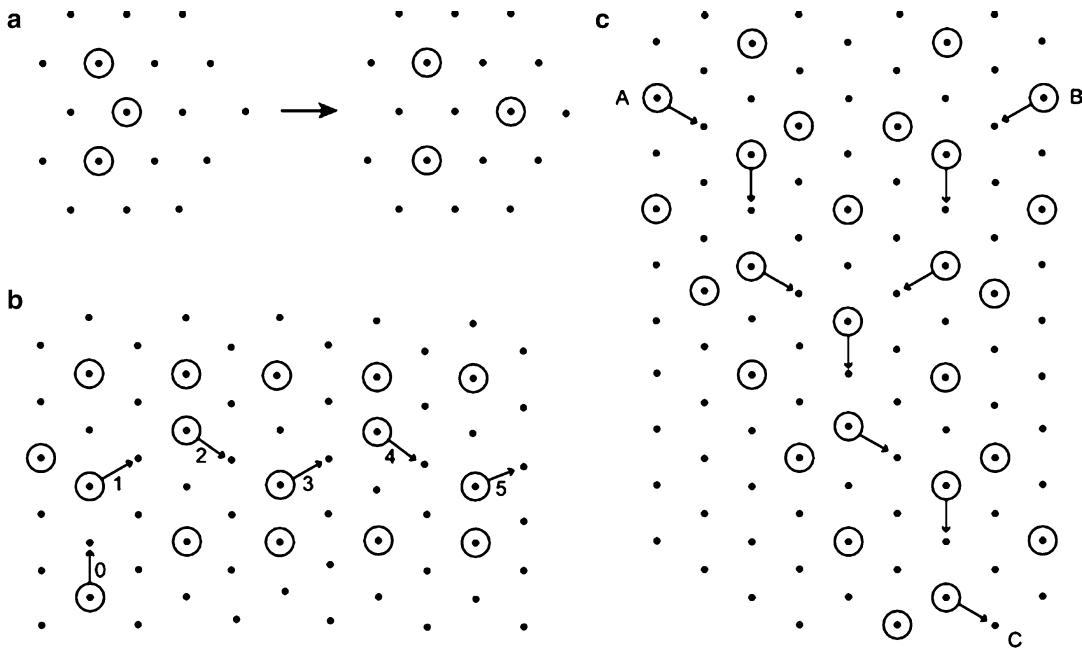


**Nanocomputers, Fig. 3** Quantum Dot Cellular Automaton (QCA). (a) A cell in the QCA contains four quantum dots storing two electrons. Since the electrons repel each other, they are likely to be in opposite corners of the cell. This allows for two possible cell configurations, which are interpreted as signals 0 and 1. (b) Due to the Coulomb interactions between electrons in quantum dots of neighboring cells, signals propagate along cells. A wire with cells in configuration 0 will change from the left to the right into a wire with cells in configuration 1 if the left-most

cell is kept in configuration 1. (c) The same mechanism also works for wires taking left- and right-turns. (d) A NOT-gate formed from cells in the QCA. Change of input signal 0 to 1 results in the output signal changing from 1 to 0. (e) A majority-gate formed from cells in the QCA. From the majority-gate an AND-gate can be constructed by setting one of the inputs to the constant value 0. An OR-gate can be constructed by setting one input to 1

interactions rather than explicit transition rules, this model has very simple partitioned cells that are structured such as to give control over both (a) which cells interact as neighbors and (b) when these interactions occur. Its design is based on Margolus' reversible cellular automaton (Margolus 1984) implementing Fredkin's billiard ball model (BBM) (Fredkin and Toffoli 1982) (Reversible Cellular Automata). The cells are suitable for implementation by quantum-dot devices that are organized such that electrical charge is transferred under the control of optical clock

signals of different wave lengths supplied in a specific order. A similar way to control operations by electromagnetic or optical signals in a cellular automaton – though then applied to binary addition rather than universal computation – is described in Benjamin and Johnson (1997). Somewhat related is the cellular automaton model with less regular topologies in Benjamin and Johnson (1999). A 1-dimensional cellular automaton in which transitions are triggered by electromagnetic pulses of well-defined frequency and length is described in Lloyd (1993). Apart



**Nanocomputers, Fig. 4** Molecule Cascades based on CO molecules (indicated by circles) placed on a grid of copper atoms (indicated by dots). (a) Chevron configuration, in which the center molecule moves one grid point

away due to repulsive forces. (b) Wire based on the Chevron configuration. The arrows indicate the hops made by the CO molecules. (c) AND-gate based on the Chevron configuration ( $c = A \wedge B$ )

from classical operations like the AND, OR, and NOT, quantum-mechanical operations are in principle possible by this scheme.

Cellular automata governed by nonlinear dynamics, so-called *Cellular Neural Networks* (Chua and Yang 1988), hold another promise for nanocomputers, since the required physical implementations may be relatively simple. They are explored for this purpose in more detail in (Kiehl 2006; Yang et al. 2001). The logic states of such cellular automata can be expressed in terms of the electrical phases in a dynamic physical process. A possible implementation of such a model is by *tunneling phase logic* (Yang et al. 2001). Tunneling phase logic still suffers from many problems (Bourianoff 2003), however, ranging from the difficulty to manufacture uniform tunneling diodes to the occurrence of stray background charges.

Implementations of cellular automata utilizing physical interactions between CO molecules arranged on a copper (Cu) surface are described in Eigler et al. (1819); Heinrich et al. (2002). The atoms in the Cu surface form a triangular grid, on

which the CO molecules settle, with the carbon atom of a molecule oriented toward a grid point. When two molecules are in nearest grid points, they will slightly point away from each other, due to a weak repulsive force between them. This force is sufficiently strong in some configurations of CO molecules to move certain CO molecules by one grid point, particularly in the so-called *Chevron-configuration* (Fig. 4a). Organized in suitable configurations, CO molecules will move on the grid in succession, triggering each other's hops, like domino stones falling in succession. These configurations, called *Molecule Cascades*, can be exploited to transmit signals on the grid (Fig. 4b), and to conduct Boolean AND operations on signals (Fig. 4c). More complex circuits can also be formed, and these will require configurations to cross wires, and in some cases OR-gates. As compared to the interactions based on the Chevron configuration, the interactions required for wire crossings and OR-gates are more complicated, and unfortunately less reliable. Based on a wide variety of configurations, a three-bit sorter is even

possible, as shown in Heinrich et al. (2002). A formal analysis of the configurations possible by molecule cascades is given in Carmona et al. (2006). Though molecule cascades allow impressive integration densities – the three-bit sorter would require a area of merely  $200 \text{ nm}^2$  – their speed of computation is very low: it takes about one hour for the three-bit sorter to complete its computation. There is no fundamental reason, though, why different systems of molecules would not be much faster. Setting up a molecular cascade system for computation involves a careful and time-consuming manipulation of individual molecules by a Scanning Tunneling Microscope (STM), so these types of systems are far from practical applications. A distinct disadvantage of the molecule cascades in Heinrich et al. (2002) is that they allow only one-time computation: once a computation ends, the molecules have to be moved back to their initial positions to conduct the computation once more. An important step toward a practical system would be the development of a mechanism inherently present in the system to reinitialize molecules.

The cellular automata discussed above utilize the physical interactions inherent in their design to conduct transitions, an approach perhaps best illustrated in Margolus (1999). Though this tends to result in extremely simple cells, it may be challenging to build additional functionality into cells, such as functionality to configure cells into states necessary to initiate certain computations. This may be especially difficult if such a configuration process needs to take place on only part of the cell space. Likewise, error correction functionality is difficult to implement through simple physical interactions in a cellular space. For these reasons some researchers advocate cells that are more complex, but still sufficiently simple to potentially allow a huge number of them to be fabricated by bottom-up techniques and organized in arrays at nanometer scale integration densities.

One approach along these lines is the *Cell Matrix* (Durbeck and Macias 2001). Each cell in this model, consisting of a memory of less than 100 bytes and a few dozen gates, can be configured to conduct an operation like NAND, XOR, one-bit addition, etc.

The above cellular automaton models are all timed synchronously, requiring the delivery of a central clock signal to each cell. A synchronous mode of timing causes a wide array of problems as pointed out in section “[Heat Dissipation](#)”, like a high power consumption and heat dissipation. For this reason, cellular automata in which the principle of locality is carried one step further to the timing of the architecture, have started to attract interest in recent years. The resulting *asynchronous cellular automata* have a mode of operation in which all cells conduct their transitions at random times and independent of each other. Notwithstanding the advantage of asynchronous timing from a physical point of view, it brings up the question how to actually compute on such cellular automata in a deterministic way. The usual method to do this is to simulate a timing mechanism on an asynchronous cellular automaton to force the cells approximately into lock-step, and then using well-established methods to compute synchronously. Unfortunately, such an approach causes a significant overhead in terms of the number of cell states and transition rules. To simulate an  $n$ -state synchronous cellular automaton on its asynchronous counterpart, one needs, depending on the method, a number of states of up to  $O(n^2)$  (Peper et al. 2003). Worse yet, to synchronize different parts of an asynchronous cellular automaton with each other – necessary, because in the end it is a synchronous cellular automaton that is being simulated – exchange of signals between these parts is required. This exchange takes the form of so-called *synchronization waves* that propagate along the cell space (Peper et al. 2003). Since all cells need to continuously change their states to let these waves pass, there are many transitions being conducted that do not contribute to the computation, even in areas of the cell space where no signals or active configurations are present. Implemented physically, such asynchronous cellular automata would thus need to consume much energy for dummy transitions, making them hardly better candidates for nanocomputer architectures than synchronous cellular automata.

A more efficient approach to conduct computations on asynchronous cellular automata is to

conduct asynchronous computations directly, using only synchronization between cells on strictly local scales. This approach is based on embedding a circuit on the cellular space and then simulating the circuit's operation through the cooperative transitions of the cells. While the circuits in synchronous models are typically standard logic circuits involving AND, OR, and NOT gates, different functionalities are required for asynchronous circuits, but the general idea of embedding circuits on the cell space remains the same. The approach in (Adachi et al. 2004; Lee et al. 2003, 2005; Peper et al. 2003, 2004) is to use delay-insensitive circuits (see section “[Heat Dissipation](#)”), which, due to their robustness to signal delays, combine very well with asynchronous cellular automata, since the requirement no longer holds that signals must arrive at certain locations at certain times dictated by a central clock, as in synchronous circuits. This takes away concerns about variations in the operational speed of cells and considerably simplifies the design of configurations representing circuit elements embedded in the cellular space.

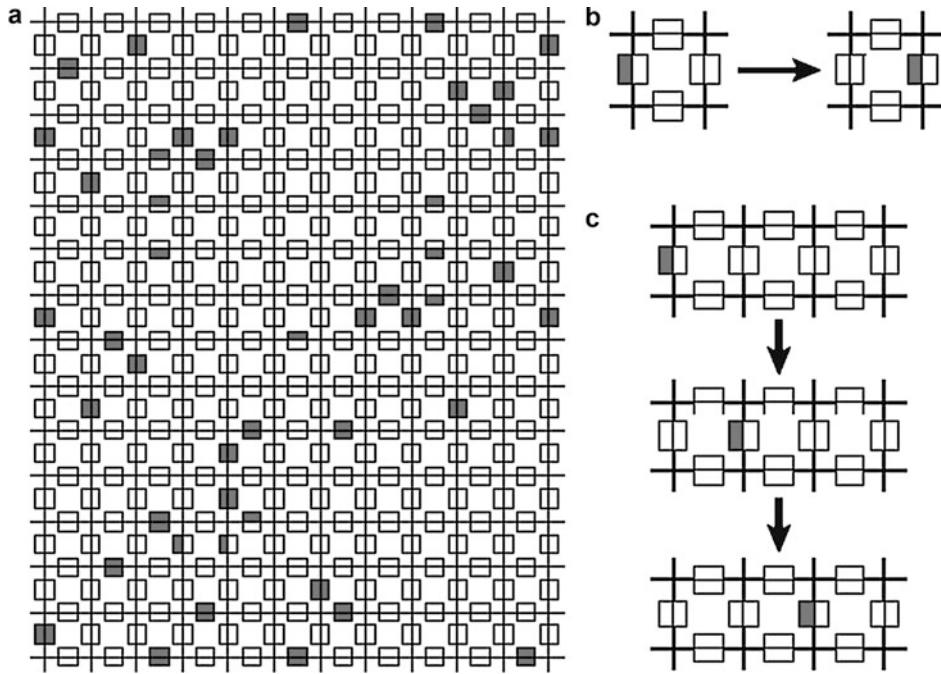
To avoid the randomness in the timing of transitions interfering with proper deterministic operation of an asynchronous cellular automaton, its design involves transition rules that serialize transitions into well-controlled sequences of cell states. Some asynchronous cellular automata designed according to these principles can be found in Adachi et al. (2004); Lee et al. (2003, 2005). Unfortunately, many of these models still require tens of transition rules, which may hinder efficient physical implementations. To this end, alternative models have been proposed, called *Self-Timed Cellular Automata (STCA)* (Peper et al. 2003), in which cells are subdivided in four partitions, each representing a part of a cell's state information (see Fig. 5a). This results in a substantial reduction of the number of transition rules. The model in Peper et al. (2004) employs merely six rules. One of these rules, a rule for signal propagation is shown in Fig. 5b and its effect on a cell configuration of a signal is shown in Fig. 5c.

In order to compute on a cellular automaton, it is necessary to configure it for the computation. One way to do this is by having a separate

mechanism for (re-) configuration, through a separate layer of circuitry to address each individual cell like in a memory, and writing the required state into it. Another way is to have the reconfiguration done by the cells themselves. This requires the transition rules to contain sufficient functionality to represent the reconfiguration mechanism. Such functionality resembles self-reproduction of configurations in cellular automata. In other words, such cellular automata require *construction universality* (Self-Replication and Cellular Automata). Unfortunately, it turns out that implementing construction universality on a cellular automaton tends to give a large overhead in terms of the number of transition rules. A self-contained mechanism for reconfiguration may thus not be practical, absent a novel ingenious way to implement construction universality, probably inspired by some biological mechanisms.

Reconfiguration in cellular automata can also be used to achieve defect-tolerance. In the approach in Isokawa et al. (2003), defects are detected and isolated from non-defective cells through waves of cells in certain states propagating over the cell space, leaving defective cells, which are unable to adhere to the state changes, standing out to be detected as defective (Isokawa et al. 2003). This approach is taken one step further in Isokawa et al. (2007) by additionally scanning the cell-space for defect-free areas on which circuits can be configured. Being self-contained, this scanning process uses reconfiguration mechanisms resembling the self-reproduction functionality on cellular automata. An online approach to defect-tolerance is followed in Isokawa et al. (2006), in which configurations called *random flies* float around randomly in the cellular space, and stick to configurations that are static. A configuration unable to compute due to defects is static in this model and will thus be isolated by a layer of random flies stuck to it. Being highly experimental, these approaches require an overhead in terms of the number of transition rules that may be too high to be of practical significance for nanocomputers.

Approaches to fault-tolerance in cellular automaton-based nanocomputers tend to focus on economic ways to implement redundancy. Early work in this line of research is reported in



**Nanocomputers, Fig. 5** A Self-Timed Cellular Automaton (STCA) consists of cells with four partitions. All four partitions as well as one partition of each of the four neighboring cells are rewritten by transition rules. (a) Example of a small configuration of cells with partitions

having values 0 (white) and 1 (black). (b) Transition rule for signal propagation. (c) Applying this rule to the configuration at the left results in a 1-partition moving to the right

Nishio and Kobuchi (1975). This model can correct at most one error in 19 cells, but to this end each cell requires a neighborhood of 49 cells in its transition rules, which is much higher than usual cellular automata. The increased complexity of cells suggests that they will be very error-prone in physical implementations, limiting the practical value of this work. The model in Harao and Noguchi (1975) suffers from similar problems.

Better fault-tolerance is obtained in Gács (1986, 1989); Gács and Reif (1988) with synchronous cellular automata, and in Gács (1997); Wang (1990) with asynchronous cellular automata simulating synchronous cellular automata: the idea is to organize cells in blocks that perform a fault-tolerant simulation of a second cellular automaton, which on its turn is also organized in blocks, simulating even more reliably a third cellular automaton, and so on. This results in a hierarchical structure with high reliability at the higher levels, like with the CTMR technique mentioned

in section “[Fault-Tolerance](#)”. The cells in these models, however, are too complex to be suitable for efficient physical implementations, since they contain information regarding the hierarchical organization, such as block structure, address within a block, programs selecting transition rules, and timing information. In Isokawa et al. (2004) a fault-tolerant STCA based on *BCH codes* (e.g. Mac Williams and Sloane 1978) is proposed, but computation on this model is inefficient, since only one signal at a time is allowed.

Spielman’s work (1996) on implementing a computation scheme in an encoded space based on an error correcting code, mentioned in section “[Fault-Tolerance](#)”, has resulted in related work (Peper et al. 2004) in the context of STCAs. Each partition of an STCA’s cell and each adjacent partition of a neighboring cell is stored in a memory and encoded by an error correcting code. Up to one third of the memory’s bits can – if corrupted – be corrected by this method.

Cellular automata in which cells have the complexities of processing elements (Fountain et al. 1998; Waingold et al. 1997), rather than (much simpler) finite automata, form the other end of the spectrum of cellular nanocomputer architectures. Since the fabrication of even simple processing elements are likely to be beyond the reach of bottom-up techniques, this approach may be infeasible for the same reasons as to why alternatives to von Neumann architectures are proposed in the first place for nanocomputers.

In conclusion, the main challenges for cellular automaton-based nanocomputers are:

- The design of models requiring a very small number of transition rules i.e., in the order of less than 10,
- The design or discovery of a mechanism to implement transition rules in a physically feasible way,
- To implement reconfiguration ability, while keeping the number of transition rules small,
- To implement fault- and defect-tolerance while keeping the number of transition rules and cell states small.

## Crossbar Array-Based Architectures

A crossbar consists of a set of, say,  $N$  horizontal wires that cross another set of  $M$  vertical wires. The  $N \times M$  cross points of the wires form a matrix of switches, which can be individually set (programmed) in a low-resistance state (closed) or a high-resistance state (opened). The simple structure of two planes of parallel wires allows for a wide variety of bottom-up fabrication techniques, such as self-assembly by fluidics-based alignment (Huang et al. 2001), directed self-assembly based on chemical techniques in combination with an alternating current (AC) electric field (Diehl et al. 2002), or fabrication including top-down techniques like nano-imprinting (Jung et al. 2006; Wu et al. 2005). Alignment of wires in order to connect them – a major problem in nanoelectronics – is relatively straightforward for crossbar arrays, due to the wires being perpendicular. Combined with a significant flexibility in

making and altering connections, this accounts for much of the attention for crossbar arrays in the context of nanocomputers. Nanowires with diameters of only a few nanometers that can be used in implementations of crossbar arrays have already been fabricated, like single-crystal nanowires (Cui et al. 2001; Huang et al. 2001; Kamins et al. 2000; Morales and Lieber 2001) and carbon nanotube wires (Soh et al. 1999). The crossbar in Rueckes et al. (2000) employs carbon nanotubes as its wires that can be programmed by applying a voltage across a crosspoint of two wires. This voltage, which is higher than that for normal (non-programming) operation of the crossbar, results in the wires being bent toward each other to close the switch. After removing the voltage, these deformations are preserved due to van der Waals forces, so memory functionality is inherent to this mechanism. To open the switch again, an opposite voltage is applied, which drives the wires apart. Nanometer scale implementations of crossbar arrays like the above provide significant efficiency as compared to their CMOS-based counterparts, the latter requiring a separate 1-bit memory for each cross point to store the state of the corresponding switch.

Another nanoscale crossbar design places a very thin layer of rotaxane molecules between the two wire planes in the crossbar (Chen et al. 2003). Molecules in this layer at crosspoints can be set into low-resistance or high-resistance states – again through applying a voltage of sufficient strength – and these states tend to remain unchanged throughout successive read operations.

The crossbar is used in the following contexts:

Routing	When used as a interconnection array, the crossbar array allows the routing of any input from a (single) vertical wire to any output on a (single) horizontal wire, by closing the switch between these wires. The resulting flexibility in connectivity provides an efficient way to route around defects
Logic	Logic when implemented by crossbar arrays usually takes the form of wired (N)OR-planes or (N)AND-planes (DeHon 2003; Kuekes et al. 2005b; Snider et al. 2004, 2005), whereby wires

(continued)

	in one plane provide inputs and wires in the other plane provide outputs
Memory	The 1-bit memories at the crosspoints offer the potential of huge storage capacities when combined in large crossbar arrays (Chen et al. 2003; Green et al. 2007; Kuekes et al. 2000; Wu et al. 2005). Reading out a memory bit is usually done by applying a voltage to one of the corresponding wires, resulting in the state of the memory appearing on the other wire. Realizations as associative memories have also been considered, whereby information storage is distributed over the switches such that recovery is possible even if some switches fail (Davis et al. 2004; Snider and Kuekes 2003)
Address decoding	The connections between the two wire planes can also be set such that a demultiplexer is obtained, which can address a single wire out of the $N$ wires in one plane by much less than $N$ wires in the other plane, e.g., down to $O(\sqrt{N})$ or even $O(\log N)$ wires (Chen et al. 2003; DeHon 2003; Williams and Kuekes 2001; Zhong et al. 2003). Apart from use in combination with crossbar arrays for routing, logic, or memory, a demultiplexer is also an excellent interface between micro-scales on one hand and nano-scales on the other, allowing for a limited number of microwires in one plane of a crossbar array to address a much larger number of much thinner nanowires in the other plane. Such demultiplexers tend to be static, i.e., they do not require reprogramming of the switch settings. When used as micro/nano-interfaces, reprogramming may even be impossible as nanowires cannot be accessed directly; prepatterned connections (DeHon 2003) or stochastically assembled connections (Williams and Kuekes 2001) may be suitable options in this case

The cross points in crossbar arrays can take various forms, the most common of which are resistors, diodes – both combined with a switch in each cross point – and Field Effect Transistors (FETs). Resistor crossbars, followed by diode crossbars, are easiest to fabricate by bottom-up techniques, since both are 2-terminal devices, which significantly simplifies alignment problems.

Resistor crossbars, additionally, can be built using metal nanowires, facilitating low output impedances, but their linear device characteristics may complicate the approximation of nonlinear functionalities (Snider and Robinett 2005). Though diode and FET crossbars will do better with regard to the latter point, they carry significant disadvantages with existing fabrication technology (Snider and Robinett 2005): limited current density, forward-biased voltage drops, relatively high impedance of semiconductor nanowires, difficulty of configuration, etc. Resistor and diode crossbars, on the other hand, come with another disadvantage (Stan et al. 2003): being 2-terminal devices, they are passive, thus lacking signal gain – a major disadvantage as compared to 3-terminal devices like transistors – complicating the restoration of signals. Moreover, 2-terminal devices tend to have higher power consumption than devices like transistors, due to the inherently higher currents flowing to ground (Stan et al. 2003).

The lack of signal restoration is especially problematic when many levels of logic are required through which signals pass. Minimizing this number of levels, however, causes an inefficient use of hardware: it prevents the exploitation of the logic structure of a circuit design, since each level – such as an OR-plane or an AND-plane in a crossbar array – has a homogeneous structure. This is a well-known problem in Programmable Logic Arrays (PLAs) (DeHon 2005): an  $n$ -input XOR, for example, requires a number of product terms that is exponential in  $n$  when expressed by the 2-level logic of a single PLA. It is thus unsurprising that signal restoration in crossbar arrays has received significant attention.

Several proposals have been made to include signal restoration functionality in crossbar arrays. DeHon et al. (2003) realizes some of the cross points between two wires as FETs, such that the electric field of one wire controls the conductivity of the other wire. This requires doping of (parts of) a wire, but the problems related to that, such as alignment of doped parts of wires with crossing wires, may require further research. Kuekes et al. (2005c) propose to use a bistable-switch latch to restore signals, in which the logic value of a wire is controlled by two switches, one connecting to a

logic 0 and one to a logic 1, such that only one switch is closed at a time. Goldstein and Budiu (2001) also uses a latch for signal restoration, though in this case the latch is composed of a wire with two Negative Differential Resistance molecules at either end. There are also schemes in which the inherent poor voltage margins of resistor-based crossbar logic is improved by including error-correcting codes in the scheme (Kuekes et al. 2005b, 2006).

Within the class of crossbar arrays for memory applications, there is a pronounced difference in terms of storage capacity between crossbar arrays based on diodes and FETs on the one hand, and crossbar arrays based on resistors on the other hand. The use of diodes or FETs ensures that each cross point is independent of the other cross points, since electrical current can only flow in one direction in these designs. In resistor-based crossbar arrays, on the other hand, there will often be an indirect (parasitic) path through which a current flows when wires are connected at multiple cross points to perpendicular wires. This has a profound impact on the storage capacity of memories based on crossbar arrays. Whereas an  $N \times N$  crossbar array will have an  $O(N^2)$  storage capacity when based on diodes or FETs, it will have only  $O(N \log N)$  storage capacity when based on resistors (for more detailed estimates of storage capacities see Sotiriadis 2006).

Though a standalone crossbar array is insufficient as an architecture for nanocomputers, it can be applied at different places and at different levels in a hierarchy to form a more or less “complete” architecture.

Goldstein’s NanoFabrics (Goldstein and Budiu 2001), for example, consist of so-called nanoBlocks and switch-blocks organized in tiles called clusters that are connected to each other through long nanowires of varying lengths (e.g. 1, 2, 4, and 8 clusters long). These wires allow signals to traverse a few clusters without the need to go through switches. The nanoBlocks themselves consist of logic based on crossbar arrays, and the switch-blocks consist of interconnections based on crossbar arrays. Defect-tolerance is obtained by testing the circuits – like in the Teramac, mentioned in section “Fault-Tolerance” – but unlike

the Teramac, testing is conducted in incremental stages in which an increasing part of the hardware – once it has been configured around defects – is used in the testing of the remaining parts (Mishra and Goldstein 2003). Testing the initial part is conducted by a host.

DeHon’s Nanowire-Based Programmable Architecture (DeHon 2005) provides designs for Wired-OR logic, interconnection, and signal restoration implemented by atomic-scale nanowire crossbar arrays.

The CMOL architecture (Likharev and Strukov 2005; Strukov and Likharev 2005) is based on a reconfigurable crossbar array of nanodevices that is superimposed at a small angle on a grid of CMOS cells. It is used to implement memory, FPGA logic, or neural network inspired models (see section “[Neural Network-Based Architectures](#)”). Each CMOS cell consists of an inverter and two pass transistors serving two pins that connect to the crossbar array. Snider et al.’s Field-Programmable Nanowire Interconnect (FPNI) (Snider and Williams 2007) improves on this proposal by adjusting the crossbar array’s structure so that all CMOS-fabricated pins can be of the same height. Logic functionality in the architecture is restricted to the CMOS part and routing to the crossbar of nanowires.

Reconfigurability is one of the strong points of crossbar arrays, but it requires a time-intensive process of testing for defects to guarantee defect-free routes. In effect, manufacturing yield is traded for an expensive die per die multi-step functional test and programming process (Brillouët 2007). A combination with error-correcting codes, like in Kuekes et al. (2005a), is thought to be more economical by some authors (Bahar et al. 2007). It is unclear as to what extent reconfiguration around defects can be made self-contained in crossbar-array-based architectures.

## Neural Network-Based Architectures

A *Neural Network* is a collection of threshold elements, called neurons, of which the human brain contains about  $10^{11}$ , each connected to  $10^3$  to  $10^4$  other neurons. Neurons receive inputs from

each other through weighted connections. The strengths (weights) of the connections in a neural network are changed in accordance with the input to it; this process is called *learning*. Learning is usually accomplished through a *Hebbian learning rule*, which was first discovered by Hebb to be of fundamental importance in real brains. Hebbian learning can be shortly described as “neurons that fire together, wire together”. In other words, two neurons tend to strengthen the connection between them when their patterns of fired pulses correlate positively with each other. In artificial neural networks this rule is often implemented in terms of the correlation of the time-averaged analog values of neurons, rather than the actual pulses. Hebbian learning lies at the basis of an impressive self-organizing ability of neural networks. The model has much flexibility with regard to the connectivity of the network between the neurons, as a result of which little detailed information is required about the underlying network structure to conduct learning processes. This tends to result in a good tolerance to defects and faults, which is an important reason for the interest in neural networks for nano-computer architectures.

Neural networks tend to be especially useful to problems that involve pattern recognition, classification, associative memory, and control. Recognition of visual images, for example, is conducted by brains to an extent that is unsurpassed by traditional computers, in terms of speed (taking approximately 100 ms), as well as in terms of fidelity. Though individual neurons are slow – it takes about 10 ms for a neural signal to propagate from one neuron to the other – the system as a whole is very fast. Contrast that to the time it takes computers with clock speeds of many GHz to performing the same task, which tends to be in the order of minutes to hours.

Most studied applications of artificial neural networks have difficulty to go beyond the toy-problem level. A better knowledge about the brain may lead to better results, and there is also the expectation (Likharev and Strukov 2005) that the availability of huge amounts of hardware resources may be of help. It is important to realize, however, that the large-scale organization of the brain is directed not only by relatively easy-to-

characterize learning algorithms, but also by genetic factors, which involve a certain degree of “wetness”. The latter is not very compatible with the usual perceptions about computers. It is questionable whether the abundance of hardware resources alone will result in a breakthrough in neural networks-based architectures.

*Neuromorphic Engineering* is the term used to describe the interdisciplinary field that takes inspiration from neuroscience, computer science and engineering to design artificial neural systems. Though initially associated with VLSI systems engineering – due to Mead (1990) coining the term *neuromorphic* – it is increasingly attracting contributions related to nanocomputers. One of the early papers in this context is Roychowdhury et al. (1996), in which quantum dots arranged in a two-dimensional array and deposited on a resonant tunneling diode (RTD) are connected by nanowires, to form nodal points of nonlinear bistable elements in a conductive network. The dynamics of this system bears strong similarities to the equations representing associative memories.

Another proposal related to neuromorphic nanoarchitectures is the *Nanocell*. A Nanocell is an element containing randomly ordered metallic islands that are interlinked with molecules – acting as switches – between micrometer-sized metallic input/output leads (Tour et al. 2003). To impose certain functionalities on the nanocell, like an AND, a NOR, an Adder, etc., it is necessary for these switches to be set into certain configurations. Genetic Algorithms (GA) are used for this in Tour et al. (2002), but this method requires a detailed knowledge of the random structure inside the nanocell, so it is impractical, resulting in proposals to use neural networks, in which – ideally – the switches can be programmed through a learning process employing electrical pulses on input and output terminals of the nanocell (Husband et al. 2003). To create useful circuits from nanocells it is necessary to connect them – after programming – into certain circuits. Creating such inter-cell connectivity has not been researched in detail, but neuromorphic learning methods may be suitable in this context as well.

Boltzmann machine neural networks based on single electron tunneling devices are proposed in

Yamada et al. (2001). *Simulated annealing*, the processing in Boltzmann machines through which a minimum energy state is found in an optimization problem, is implemented through a digital oscillator based on the stochastic nature of the tunneling phenomenon. The architecture consists of a network in which outputs of neural elements are fed back to their inputs through weights in a interconnection network, which are implemented by capacitance values reflecting the particular problem to be optimized.

*CrossNets* (Türel et al. 2005) are neuromorphic architectures based on the CMOL architecture. Neurons in this model, being relatively sparse, are represented by CMOS logic, and the neural interconnections, being very dense, are represented by a crossbar array of nanometer scale wires, whereby a cross-connection between two wires represents a so-called *synapse*. The crossbar array offers a very flexible framework to connect neurons to each other: it allows connectivity ranging from hierarchical structures that are typically used in multi-layered perceptrons (Haykin 1998) to more homogeneous networks with quasi-local connections. CrossNets can be used for pattern classification in feedforward multilayered perceptrons as well as for pattern restoration in associative memories. The crossbar array, though very flexible, has also disadvantages in this context, because it limits interconnection weights to be binary, and it is hard to avoid interference between different switches when setting or updating weights. Some solutions to these problems may be reached by using multiple-wire cross connections per weight (Türel et al. 2005). The above models employ neural signals that have analogous values, as opposed to the spiking signals that are common in real brains. Spikes offer the opportunity for decreased power consumption, and, more importantly, their timing appears to encode important information in brains, and is thought to allow the semantic binding of different brain areas with each other. Some preliminary results on spiking neuromorphic architectures based on CMOL CrossNets and comparisons to non-spiking models are given in Gao and Hammerstrom (2007). A different neuromorphic architecture based on crossbar arrays is explored in Davis et al. (2004) for use in associative memories.

## Future Directions

In pondering the future directions nanocomputers may take, it is important to realize that CMOS technology is a formidable competitor. Detailed predictions on the pace at which this technology progresses have been made for decades, with varying degrees of success. When the emphasis is on technological limitations, the resulting predictions tend to be overly conservative, as they do not take into account the many ways often found around those limitations (Ho et al. 2001). When on the other hand trends are simply extrapolated from the past, predictions tend to be on the optimistic side, as they ignore hard physical limitations that might still be ahead (Ho et al. 2001). Notwithstanding the uncertainties of the life-time of CMOS technology, there seems to be a consensus that it can be extended to at least the year 2015. The exponential growth experienced over so many decades, however, may significantly slow down towards the end (Moore 2003). Beyond this, it is expected (Bourianoff 2003) that the benefits of increased integration densities will be extended for 30 more years through the use of emergent technologies. These technologies are likely to be used in combination with silicon technologies and gradually replace more of the latter.

Many of the limitations that are perceived in technology – and CMOS is no exception to this – are likely to be circumvented by reconsidering the underlying assumptions, as was observed by Feynman (1992). This extends to the framework of circuits and architectures as well. Reconsiderations of underlying assumptions has frequently resulted in paradigm shifts in computer science. A notable example form *Reduced Instruction Set Computers (RISC)* (Gimarc and Milutinovic 1987), which use a small set of rapidly executed instructions rather than a more expanded and slower set of instructions as was common until the 1980s in Complex Instruction Set Computers (CISC). The CISC approach was advantageous in the context of limited sizes of memories, necessitating a dense storage of instructions at the cost of increased processing in the processor itself. That changed with the increasing sizes and decreasing costs of memories, coupled with the realization

that only a small percentage of the instructions available in CISC processors were actually used in practice. In a similar way, a changing context will determine the direction in which architectures of nanocomputers develop. This chapter has identified a number of factors of importance in this respect, like the growing problems faced in the fabrication of arbitrary structures, the increasing occurrence of defects in those structures, the increasing occurrence of faults during computations, and the increasing problems with heat dissipation. These problems will have to be addressed one way or the other, and architectural solutions are likely to play an important role in them; regular or random structures in combination with (re)configuration techniques seem especially attractive in this context.

A gradual move away from the traditional von Neumann architecture toward the increasing use of parallelism is likely to occur. Initially it will be conventional processors that will be placed in parallel configurations – a trend that has already started in the first decade of the twenty-first century. As feature sizes decrease further, fabrication technology will be more and more bottom-up, resulting in parallelism becoming increasingly fine-grained, to the point that computers consist of a huge number of elements with extremely simple logic functionality that can be configured according to specifications by a programmer. In the end, architectures may look more like intelligent memory than like the computers known today. Programming methods, though apt to change as architectures change, may be relatively unaffected by the trend toward finely grained parallelism, witness the resemblance of conventional programming languages to languages for fine-grained hardware platforms like FPGAs, some of which have been designed with resemblance to the programming language C in mind.

Trends in technology are often highly influenced by breakthroughs, and it is entirely possible that future nanocomputers develop in different directions. The increasing research efforts into unconventional computing offers a clue to the wide range of what is possible in this regard. Computation models based on new media like reaction-diffusion systems, solitons, and liquid crystals

(Adamatzky 2002) are just a few of the fascinating examples that may some day work their way into practical computer architectures.

## Bibliography

- Adachi S, Peper F, Lee J (2004) Computation by asynchronously updating cellular automata. *J Stat Phys* 114(1/2):261–289
- Adamatzky A (2002) New media for collision-based computing. In: Collision-based computing. Springer, London, pp 411–442
- Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266(11):1021–1024
- Appenzeller J, Joselevich E, Hönlein W (2003) Carbon nanotubes for data processing. In: Nanoelectronics and information technology. Wiley, Berlin, pp 473–499
- Athas WC, Svensson LJ, Koller JG, Tzartzanis N, Chou EYC (1994) Low-power digital systems based on adiabatic-switching principles. *IEEE Trans Very Large Scale Integr Syst* 2(4):398–407
- Aviram A, Ratner MA (1974) Molecular rectifiers. *Chem Phys Lett* 29(2):277–283
- Bahar RI, Hammerstrom D, Harlow J, Joyner WH Jr, Lau C, Marculescu D, Orailoglu A, Pedram M (2007) Architectures for silicon nanoelectronics and beyond. *Computer* 40(1):25–33
- Ball P (2006) Champing at the bits. *Nature* 440(7083):398–401
- Banu M, Prodanov V (2007) Ultimate VLSI clocking using passive serial distribution. In: Future trends in microelectronics: up the nano creek. Wiley, Hoboken, pp 259–276
- Bashirullah R, Liu W (2002) Raised cosine approximation signalling technique for reduced simultaneous switching noise. *Electron Lett* 38(21):1256–1258
- Beckett P, Jennings A (2002) Towards nanocomputer architecture. In: Lai F, Morris J (eds) Proceedings of 7th Asia-Pacific computer systems architecture conference ACSAC'2002 (Conference on research and practice in information technology), vol 6. Australian Computer Society, Darlinghurst
- Benioff P (1980) The computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *J Stat Phys* 22(5):563–591
- Benioff P (1984) Comment on: dissipation in computation. *Phys Rev Lett* 53(12):1203
- Benjamin SC, Johnson NF (1997) A possible nanometer-scale computing device based on an adding cellular automaton. *Appl Phys Lett* 70(17):2321–2323
- Benjamin SC, Johnson NF (1999) Cellular structures for computation in the quantum regime. *Phys Rev A* 60(6):4334–4337
- Bennett CH (1973) Logical reversibility of computation. *IBM J Res Dev* 17(6):525–532
- Bennett CH (1982) The thermodynamics of computation – a review. *Int J Theor Phys* 21(12):905–940

- Bennett CH (1984) Thermodynamically reversible computation. *Phys Rev Lett* 53(12):1202
- Bennett CH (1988) Notes on the history of reversible computation. *IBM J Res Dev* 32(1):16–23
- Biafore M (1994) Cellular automata for nanometer-scale computation. *Physica D* 70:415–433
- Birge RR, Lawrence AF, Tallent JR (1991) Quantum effects, thermal statistics and reliability of nanoscale molecular and semiconductor devices. *Nanotechnology* 2(2):73–87
- Bohr MT, Chau RS, Ghani T, Mistry K (2007) The high  $\kappa$  solution. *IEEE Spectr* 44(10):23–29
- Bourianoff G (2003) The future of nanocomputing. *Computer* 36(8):44–53
- Brillouët M (2007) Physical limits of silicon CMOS: real showstopper or wrong problem? In: Future trends in microelectronics. Up the Nano Creek Wiley, Hoboken, pp 179–191
- Carmona J, Cortadella J, Takada Y, Peper F (2006) From molecular interactions to gates: a systematic approach. In: ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on computer-aided design, San Jose, 5–9 Nov 2006
- Carter FL (1983a) The chemistry in future molecular computers. In: Computer applications in chemistry, proceedings of 6th international conference on computers in chemical research and education. Elsevier, Amsterdam, pp 225–262
- Carter FL (1983b) Molecular level fabrication techniques and molecular electronic devices. *J Vac Sci Technol B* 1(4):959–968
- Carter FL (1984) The molecular device computer: point of departure for large scale cellular automata. *Physica D* 10(1–2):175–194
- Cavin RK, Zhirnov VV, Hutchby JA, Bourianoff GI (2005) Energy barriers, demons, and minimum energy operation of electronic devices. *Proc SPIE* 5844, pp 1–9
- Ceruzzi P (1998) A history of modern computing. MIT Press, Cambridge
- Chan SC, Shepard KL, Restle PJ (2005) Uniform-phase uniform-amplitude resonant-load global clock distributions. *IEEE J Solid-State Circuits* 40(1):102–109
- Chen Y, Jung GY, Ohlberg DAA, Li X, Steward DR, Jeppesen JO, Nielsen KA, Stoddard JF, Williams RS (2003) Nanoscale molecular-switch crossbar circuits. *Nanotechnology* 14(4):462–468
- Choi H, Mody C (2007) Molecular electronics in the *longue durée*: the microelectronics origins of nanotechnology. In: Joint Wharton-chemical heritage foundation symposium on the social studies of nanotechnology, Philadelphia, 7–8 Jun 2007
- Chou SY, Krauss PR, Renstrom PJ (1996) Imprint lithography with 25-nanometer resolution. *Science* 272(5258):85–87
- Chua LO, Yang L (1988) Cellular neural networks: theory. *Circuit Syst IEEE Trans* 35(10):1257–1272
- Collier CP, Wong EW, Belohradský M, Raymo FM, Stoddart JF, Kuekes PJ, Williams RS, Heath JR (1999) Electronically configurable molecular-based logic gates. *Science* 285(5426):391–394
- Collier CP, Mattersteig G, Wong EW, Luo Y, Beverly K, Sampaio J, Raymo FM, Stoddart JF, Heath JR (2000) A [2]Catenane-based solid state electronically reconfigurable switch. *Science* 289(5482):1172–1175
- Constantinescu C (2007) Impact of intermittent faults on nanocomputing devices. In: Workshop on dependable and secure nanocomputing, Edinburgh, 28 Jun 2007
- Cowburn RP, Welland ME (2000) Room temperature magnetic quantum cellular automata. *Science* 287(5457):1466–1468
- Cui Y, Lieber CM (2001) Functional nanoscale electronic devices assembled using silicon nanowire building blocks. *Science* 291(5505):851–853
- Cui Y, Lieber C, Lauhon L, Gudiksen M, Wang J (2001) Diameter-controlled synthesis of single crystal silicon nanowires. *Appl Phys Lett* 78(15):2214–2216
- Dasmahapatra S, Werner J, Zauner KP (2006) Noise as a computational resource. *Int J Unconv Comput* 2(4):305–319
- Davari B (1999) CMOS technology: present and future. In: Proceedings of IEEE symposium on VLSI circuits. Digest of technical papers, pp 5–9
- Davis A, Nowick SM (1997) An introduction to asynchronous circuit design. Tech Rep UUUCS-97-013, Computer Science Department, University of Utah
- Davis BA, Principe JC, Fortes JAB (2004) Design and performance analysis of a novel nanoscale associative memory. In: Proceedings of 4th IEEE conference on nanotechnology, pp 314–316
- Debray P, Raichev OE, Rahman M, Akis R, Mitchel WC (1999) Ballistic transport of electrons in T-shaped quantum waveguides. *Appl Phys Lett* 74(5):768–770
- DeHon A (2003) Array-based architecture for FET-based nanoscale electronics. *IEEE Trans Nanotechnol* 2(1):23–32
- DeHon A (2004) Law of large numbers system design. In: Nano, quantum and molecular computing: implications to high level design and validation. Kluwer, Norwell, pp 213–241
- DeHon A (2005) Nanowire-based programmable architectures. *ACM J Emerg Technol Comput Syst* 1(2):109–162
- DeHon A, Lincoln P, Savage JE (2003) Stochastic assembly of sublithographic nanoscale interfaces. *IEEE Trans Nanotechnol* 2(3):165–174
- Dennard RH, Gaensslen FH, Yu HN, Rideout VL, Bassous E, LeBlanc AR (1974) Design of ion-implanted mosfets with very small physical dimensions. *IEEE J Solid-State Circ* 9(5):256–268
- Depledge PG (1981) Fault-tolerant computer systems. *IEE Proc A* 128(4):257–272
- Diehl MR, Yaliraki SN, Beckman RA, Barahona M, Heath JR (2002) Self-assembled deterministic carbon nanotube wiring networks. *Angew Chem Int Ed* 41(2):353–356
- Dobrushin RL, Ortyukov SI (1977) Upper bound for the redundancy of self-correcting arrangements of unreliable functional elements. *Probl Inform Transm* 13(3):203–218
- Drexler KE (1986) Engines of creation. Anchor Books, New York

- Drexler KE (1992) Nanosystems: molecular machinery, manufacturing, and computation. Wiley, New York
- Durbeck LJK, Macias NJ (2001) The cell matrix: an architecture for nanocomputing. *Nanotechnology* 12(3):217–230
- Eigler DM, Lutz CP, Crommie MF, Mahoran HC, Heinrich AJ (1819) Gupta JA (2004) Information transport and computation in nanometer-scale structures. *Philos Trans R Soc Lond A* 362:1135–1147
- Feynman RP (1985) Quantum mechanical computers. *Optics News* 11:11–20
- Feynman RP (1992) There's plenty of room at the bottom (reprint of 1959 lecture). *J Microelectromech Syst* 1(1):60–66
- Feynman RP, Leighton R, Sands M (2006) Ratchet and pawl. In: The Feynman lectures on physics, vol 1. Addison Wesley, San Francisco, pp 1–9
- Fountain TJ, Duff MJB, Crawley DG, Tomlinson CD, Moffat CD (1998) The use of nanoelectronic devices in highly parallel computing systems. *IEEE Trans VLSI Syst* 6(1):31–38
- Frank MP (2005) Introduction to reversible computing: motivation, progress, and challenges. In: CF '05: Proceedings of the 2nd conference on computing frontiers. ACM Press, New York, pp 385–390
- Frazier G, Taddiken A, Seabaugh A, Randall J (1993) Nanoelectronic circuits using resonant tunneling transistors and diodes. In: Digest of technical papers. IEEE international solid-state circuits conference (ISSCC), San Francisco, 24–26 Feb 1993, pp 174–175
- Fredkin E, Toffoli T (1982) Conservative logic. *Int J Theor Phys* 21:219–253
- Fukš H (2002) Nondeterministic density classification with diffusive probabilistic cellular automata. *Phys Rev E* 66(6):066106
- Gács P (1986) Reliable computation with cellular automata. *J Comput Syst Sci* 32(1):15–78
- Gács P (1989) Self-correcting two-dimensional arrays. In: Micali S (ed) Randomness in computation, Advances in computing research (a scientific annual), vol 5. JAI Press, Greenwich, pp 223–326
- Gács P (1997) Reliable cellular automata with self-organization. In: IEEE symposium on foundations of computer science, pp 90–99
- Gács P, Reif X (1988) A simple three-dimensional real-time reliable cellular array. *J Comput Syst Sci* 36(2): 125–147
- Gao C, Hammerstrom D (2007) Cortical models onto CMOL and CMOS – architectures and performance/price. *IEEE Trans Circ Syst I: Regul Pap* 54(11): 2502–2515
- Gil D, de Andrés D, Ruiz JC, Gil P (2007) Identifying fault mechanisms and models of emerging nanoelectronic devices. In: Workshop on dependable and secure nanocomputing (DSN'07). Online proceedings [www.laas.fr/WDSN07/WDSN07\\_files/Texts/WDSN07-POST-01-Gil.pdf](http://www.laas.fr/WDSN07/WDSN07_files/Texts/WDSN07-POST-01-Gil.pdf). Accessed 5 Aug 2008
- Gimarc CE, Milutinovic VM (1987) A survey of RISC processors and computers of the mid-1980s. *Computer* 20(9):59–69
- Goldstein SC (2005) The impact of the nanoscale on computing systems. In: IEEE/ACM international conference on computer-aided design (ICCAD 2005), San Jose, pp 655–661. Online Proceedings [www.cs.cmu.edu/~seth/papers/goldstein-iccad05.pdf](http://www.cs.cmu.edu/~seth/papers/goldstein-iccad05.pdf). Accessed 5 Aug 2008
- Goldstein SC, Budiu M (2001) Nanofabrics: spatial computing using molecular electronics. In: Proceedings of the 28th annual international symposium on computer architecture, pp 178–191
- Graham P, Gokhale M (2004) Nanocomputing in the presence of defects and faults: a survey. In: Nano, quantum and molecular computing. Kluwer, Boston, pp 39–72
- Green JE, Choi JW, Boukai A, Bumimovich Y, Johnston-Halperin E, Delonno E, Luo Y, Sheriff BA, Xu K, Shin YS, Tseng HR, Stoddart JF, Heath JR (2007) A 160-kilobit molecular electronic memory patterned at  $10^{11}$  bits per square centimeter. *Nature* 445(7126): 414–417
- Han J, Jonker P (2003) A defect- and fault-tolerant architecture for nanocomputers. *Nanotechnology* 14(2): 224–230
- Han J, Gao J, Qi Y, Jonker P, Fortes JAB (2005) Toward hardware-redundant, fault-tolerant logic for nanoelectronics. *IEEE Des Test Comput* 22(4):328–339
- Harao M, Noguchi S (1975) Fault tolerant cellular automata. *J Comput Syst Sci* 11(2):171–185
- Hartmannis J (1995) On the weight of computations. *Bull Eur Assoc Theor Comput Sci* 55:136–138
- Hauck S (1995) Asynchronous design methodologies: an overview. *Proc IEEE* 83(1):69–93
- Haykin S (1998) Neural networks: a comprehensive foundation. Prentice Hall PTR, Upper Saddle River
- Heath JR, Kuekes PJ, Snider GS, Williams RS (1998) A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science* 280(5370): 1716–1721
- Heinrich AJ, Lutz CP, Gupta JA, Eigler DM (2002) Molecule cascades. *Science* 298(5597):1381–1387
- Ho R, Mai KW, Horowitz MA (2001) The future of wires. *Proc IEEE* 89:490–504
- Huang Y, Duan X, Wei Q, Lieber C (2001) Directed assembly of one-dimensional nanostructures into functional networks. *Science* 291(5504):630–633
- Husband CP, Husband SM, Daniels JS, Tour JM (2003) Logic and memory with nanocell circuits. *IEEE Trans Electron Dev* 50(9):1865–1875
- Hush NS (2003) An overview of the first half-century of molecular electronics. *Ann N Y Acad Sci* 1006:1–20
- Isokawa T, Abo F, Peper F, Kamiura N, Matsui N (2003) Defect-tolerant computing based on an asynchronous cellular automaton. In: Proceedings of SICE annual conference, Fukui, pp 1746–1749
- Isokawa T, Abo F, Peper F, Adachi S, Lee J, Matsui N, Mashiko S (2004) Fault-tolerant nanocomputers based on asynchronous cellular automata. *Int J Mod Phys C* 15(6):893–915
- Isokawa T, Kowada S, Peper F, Kamiura N, Matsui N (2006) Online marking of defective cells by random

- flies. In: Yacoubi SE, Chopard B, Bandini S (eds) Lecture notes in computer science, vol 4173. Springer, Berlin, pp 347–356
- Isokawa T, Kowada S, Takada Y, Peper F, Kamiura N, Matsui N (2007) Defect-tolerance in cellular nanocomputers. *New Gener Comput* 25(2):171–199
- International Roadmap Committee (2005a) International technology roadmap for semiconductors
- International Roadmap Committee (2005b) International technology roadmap for semiconductors, emerging research devices. [www.itrs.net/Links/2005ITRS/ERD2005.pdf](http://www.itrs.net/Links/2005ITRS/ERD2005.pdf). Accessed 5 Aug 2008
- International Roadmap Committee (2005c) International technology roadmap for semiconductors, interconnect. [www.itrs.net/Links/2005ITRS/ERD2005.pdf](http://www.itrs.net/Links/2005ITRS/ERD2005.pdf). Accessed 5 Aug 2008
- Iwai H (2004) CMOS scaling for sub-90 nm to sub-10 nm. In: VLSID ‘04: Proceedings of the 17th international conference on VLSI design. IEEE Computer Society, Washington, DC, p 30
- Jablonski DG (1990) A heat engine model of a reversible computation. *Proc IEEE* 78(5):817–825
- Jung GY, Johnston-Halperin E, Wu W, Yu Z, Wang SY, Tong WM, Li Z, Green JE, Sheriff BA, Boukai A, Bunimovich Y, Heath JR, Williams RS (2006) Circuit fabrication at 17 nm half-pitch by nanoimprint lithography. *Nano Lett* 6(3):351–354
- Kamins TI, Williams RS, Chen Y, Chang YL, Chang YA (2000) Chemical vapor deposition of Si nanowires nucleated by TiSi<sub>2</sub> islands on Si. *Appl Phys Lett* 76(5):562–564
- Kiehl RA (2006) Information processing in nanoscale arrays: DNA assembly, molecular devices, nano-array architectures. In: ICCAD ‘06: proceedings of the 2006 IEEE/ACM international conference on computer-aided design, San Jose, 5–9 Nov 2006
- Kish LB (2002) End of Moore’s law: thermal (noise) death of integration in micro and nano electronics. *Phys Lett A* 305(3–4):144–149
- Kish LB (2006) Thermal noise driven computing. *Appl Phys Lett* 89(14):144104
- Knap W, Deng Y, Rumenyantsev S, Lu JQ, Shur MS, Saylor CA, Brunel LC (2002) Resonant detection of sub-terahertz radiation by plasma waves in a submicron field-effect transistor. *Appl Phys Lett* 80(18):3433–3435
- Korkmaz P, Akgul BES, Palem KV, Chakrapani LN (2006) Advocating noise as an agent for ultra-low energy computing: probabilistic complementary metal-oxide-semiconductor devices and their characteristics. *Jpn J Appl Phys* 45(4B):3307–3316
- Kreup F, Graham AP, Liebau M, Duesberg GS, Seidel R, Unger E (2004) Carbon nanotubes for interconnect applications. In: Electron devices meeting, 2004. IEDM technical digest. IEEE International, pp 683–686
- Kuekes PJ, Williams RS, Heath JR (2000) Demultiplexer for a molecular wire crossbar network. US Patent 6 128 214
- Kuekes PJ, Robinett W, Seroussi G, Williams RS (2005a) Defect-tolerant interconnect to nanoelectronic circuits: internally redundant demultiplexers based on error-correcting codes. *Nanotechnology* 16(6):869–881
- Kuekes PJ, Robinett W, Williams RS (2005b) Improved voltage margins using linear error-correcting codes in resistor-logic demultiplexers for nanoelectronics. *Nanotechnology* 16(9):1419–1432
- Kuekes PJ, Steward DR, Williams RS (2005c) The crossbar latch: Logic value storage, restoration, and inversion in crossbar circuits. *J Appl Phys* 97(3):034301
- Kuekes PJ, Robinett W, Roth RM, Seroussi G, Snider GS, Williams RS (2006) Resistor-logic demultiplexers for nanoelectronics based on constant-weight codes. *Nanotechnology* 17(4):1052–1061
- Lala PK (2001) Self-checking and fault-tolerant digital design. Morgan Kaufmann, San Francisco
- Landauer R (1961) Irreversibility and heat generation in the computing process. *IBM J Res Dev* 5(3):183–191
- Landauer R (1984) Dissipation in computation. *Phys Rev Lett* 53(12):1205
- Landauer R (1992) Information is physical. In: PhysComp ’92: workshop on physics and computation, Dallas, 2–4 Oct 1992, pp 1–4
- Le J, Pinto Y, Seeman NC, Musier-Forsyth K, Taton TA, Kiehl RA (2004) DNA-templated self-assembly of metallic nanocomponent arrays on a surface. *Nano Lett* 4(12):2343–2347
- Lee J, Adachi S, Peper F, Morita K (2003) Embedding universal delay-insensitive circuits in asynchronous cellular spaces. *Fundamenta Informaticae* 58(3/4): 295–320
- Lee J, Adachi S, Peper F, Mashiko S (2004) On reversible computation in asynchronous systems. In: Quantum information and complexity. World Scientific, Singapore, pp 296–320
- Lee J, Adachi S, Peper F, Mashiko S (2005) Delay-insensitive computation in asynchronous cellular automata. *J Comput Syst Sci* 70:201–220
- Lee J, Peper F, Adachi S (2006) Reversible logic elements operating in asynchronous mode. US Patent 6 987 402
- Lent CS, Tougaw PD, Porod W, Bernstein GH (1993) Quantum cellular automata. *Nanotechnology* 4(1): 49–57
- Li C, Fan W, Lei B, Zhang D, Han S, Tang T, Liu X, Liu Z, Asano S, Meyyappan M, Han J, Zhou C (2004) Multi-level memory based on molecular devices. *Appl Phys Lett* 84(11):1949–1951
- Liebmann LW (2003) Layout impact of resolution enhancement techniques: impediment or opportunity? In: Proceedings of 2003 international symposium on physical design (ISPD’03). ACM Press, New York, pp 110–117
- Likharev KK, Semenov VK (1991) RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Trans Appl Supercond* 1(1):3–28
- Likharev KK, Strukov DB (2005) Introduction to Molecular Electronics. In: Cuniberti G et al (eds) CMOL: devices, circuits, and architectures. Springer, Berlin, pp 447–477

- Lloyd S (1993) A potentially realizable quantum computer. *Science* 261(5128):1569–1571
- Lloyd S (2000) Ultimate physical limits to computation. *Nature* 406(6799):1047–1054
- Madou MJ (2002) Lithography. In: Fundamentals of microfabrication. The science of miniaturization. CRC Press, Florida, pp 1–76
- Maezawa K, Förster A (2003) Quantum transport devices based on resonant tunneling. In: Nanoelectronics and information technology. Wiley-VCH, Weinheim, pp 407–424
- Manohar R, Martin AJ (1995) Quasi-delay-insensitive circuits are Turing-complete. Tech. Rep. CaltechCSTR: 1995.cs-tr-95-11, California Institute of Technology, Pasadena
- Margolus NH (1984) Physics-like models of computation. *Physica D* 10(1/2):81–95
- Margolus NH (1999) Crystalline computation. In: Feynman and computation: exploring the limits of computers. Perseus books, Cambridge, pp 267–305
- Martin AJ (1990) Programming in VLSI: from communicating processes to delay-insensitive circuits. In: Hoare CAR (ed) Developments in concurrency and communication. Addison-Wesley, Reading, pp 1–64
- Mayor M, Weber HB, Waser R (2003) Molecular electronics. In: Nanoelectronics and information technology. Wiley, Berlin, pp 501–525
- Mead C (1990) Neuromorphic electronic systems. *Proc IEEE* 78(10):1629–1636
- Mead C, Conway L (1980) Introduction to VLSI systems. Addison-Wesley, Boston
- Meindl JD (1995) Low power microelectronics: retrospect and prospect. *Proc IEEE* 83(4):619–635
- Meindl JD, Chen Q, Davis JA (2001) Limits on silicon nanoelectronics for terascale integration. *Science* 293(5537):2044–2049
- Miller DAB (2000) Rationale and challenges for optical interconnects to electronic chips. *Proc IEEE* 88(6):728–749
- Mishra M, Goldstein SC (2003) Defect tolerance at the end of the roadmap. In: Proceedings of the IEEE international test conference (ITC), vol 1, pp 1201–1210
- Mizuno M, Anjo K, Surni Y, Wakabayashi H, Mogami T, Horiuchi T, Yamashina M (2000) On-chip multi-ghz clocking with transmission lines. In: 2000 I.E. international solid-state circuits conference (ISSCC). Digest of technical papers, pp 366–367
- Montemerlo MS, Love JC, Opitbeck GJ, Goldhaber-Gordon DJ, Ellenbogen JC (1996) Technologies and designs for electronic nanocomputers. Technical report 96W0000044, MITRE
- Moore GE (2003) No exponential is forever: but “forever” can be delayed! In: Solid-state circuits conference. Digest of technical papers. ISSCC. IEEE international solid-state circuits conference (ISSCC), vol 1, pp 20–23
- Morales A, Lieber C (2001) A laser ablation method for the synthesis of crystalline semiconductor nanowires. *Science* 291(5348):208–211
- Morita K (2003) A simple universal logic element and cellular automata for reversible computing. *Lect Notes Comput Sci* 2055:102–113
- Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, New York
- Muller DE, Bartky WS (1959) A theory of asynchronous circuits. In: Proceedings of an international symposium on the theory of switching. Harvard University Press, Cambridge, MA, pp 204–243
- Nikolic K, Forshaw M (2003) The current status of nano-electronic devices. *Int J Nanosci* 2(1/2):7–29
- Nikolic K, Sadek A, Forshaw M (2002) Fault-tolerant techniques for nanocomputers. *Nanotechnology* 13(3):357–362
- Nishio H, Kobuchi Y (1975) Fault tolerant cellular spaces. *J Comput Syst Sci* 11(2):150–170
- O KK, Kim K, Floyd B, Mehta J, Yoon H, Hung CM, Bravo D, Dickson T, Guo X, Li R, Trichy N, Caserta J, Bomstad W, Branch J, Yang DJ, Bohorquez J, Gao L, Sugavanan A, Lin JJ, Chen J, Martin F, Brewer J (2003) Wireless communications using integrated antennas. In: Proceedings of 2003 I.E. international interconnect technology conference, San Francisco, 2–4 June 2003, pp 111–113
- O’Mahony F, Yue CP, Horowitz MA, Wong SS (2003a) A 10-GHz global clock distribution using coupled standing-wave oscillators. *IEEE J Solid-State Circ* 38(11):1813–1820
- O’Mahony F, Yue CP, Horowitz M, Wong SS (2003b) 10 GHz clock distribution using coupled standing-wave oscillators. In: Solid-state circuits conference. Digest of technical papers. IEEE international solid-state circuits conference (ISSCC), vol 1, pp 428–504
- Ono Y, Fujiwara A, Nishiguchi K, Inokawa H, Takahashi Y (2005) Manipulation and detection of single electrons for future information processing. *J Appl Phys* 97:031101
- Palem KV (2005) Energy aware computing through probabilistic switching: a study of limits. *IEEE Trans Comput* 54(9):1123–1137
- Parviz BA, Ryan D, Whitesides GM (2003) Using self-assembly for the fabrication of nano-scale electronic and photonic devices. *IEEE Trans Adv Packag* 26(3):233–241
- Peper F, Lee J, Adachi S, Mashiko S (2003) Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers? *Nanotechnology* 14(4):469–485
- Peper F, Lee J, Abo F, Isokawa T, Adachi S, Matsui N, Mashiko S (2004) Fault-tolerance in nanocomputers: a cellular array approach. *IEEE Trans Nanotechnol* 3(1):187–201
- Petty M (2007) Molecular electronics, from principles to practice. Wiley, West Sussex
- Pinto YY, Le JD, Seeman NC, Musier-Forsyth K, Taton TA, Kiehl RA (2005) Sequence-encoded self-assembly of multiple-nanocomponent arrays by 2D DNA scaffolding. *Nano Lett* 5(12):2399–2402
- Pippenger N (1985) On networks of noisy gates. In: 26th annual symposium on foundations of computer science, 21–23 October 1985, Portland. IEEE, Washington, DC, pp 30–38
- Pippenger N (1989) Invariance of complexity measures for networks with unreliable gates. *J ACM* 36(3):531–539

- Pippenger N (1990) Developments in: "The synthesis of reliable organisms from unreliable components". In: Proceedings of symposia in pure mathematics, vol 50, pp 311–324
- Porod W (1998) Quantum-dot cellular automata devices and architectures. *Int J High-Speed Electron Syst* 9(1):37–63
- Porod W, Grondin RO, Ferry DK (1984) Dissipation in computation. *Phys Rev Lett* 52(3):232–235
- Rahman A, Reif R (2000) System-level performance evaluation of three-dimensional integrated circuits. *IEEE Trans Very Large Scale Integr Syst* 8(6):671–678
- Robert RW, Keyes W (1985) What makes a good computer device? *Science* 230(4722):138–144
- Robinson AL (1984) Computing without dissipating energy. *Science* 223(4641):1164–1166
- Rothenmund PW, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biol* 2(12):2041–2053
- Roychowdhury VP, Janes DB, Bandyopadhyay S, Wang X (1996) Collective computational activity in self-assembled arrays of quantum dots: a novel neuromorphic architecture for nanoelectronics. *IEEE Trans Electron Dev* 43(10):1688–1699
- Rueckes T, Kim K, Joselevich E, Tseng G, Cheung C, Lieber C (2000) Carbon nanotube based nonvolatile random access memory for molecular computing. *Science* 289(5476):94–97
- Sadek AS, Nikolic K, Forshaw M (2004) Parallel information and computation with restitution for noise-tolerant nanoscale logic networks. *Nanotechnology* 15(1):192–210
- Sathe V, Chueh JY, Kim J, Ziesler CH, Kim S, Papaefthymiou M (2005) Fast, efficient, recovering, and irreversible. In: CF '05: Proceedings of the 2nd conference on computing frontiers. ACM, New York, pp 407–413
- Seitz CL (1980) System timing. In: Mead CA, Conway LA (eds) *Introduction to VLSI Systems*. Addison-Wesley, Boston
- Sherman WB, Seeman NC (2004) A precisely controlled DNA biped walking device. *Nano Lett* 4(7): 1203–1207
- Shor PW (2004) Progress in quantum algorithms. *Quantum Inf Process* 3(1–5):5–13
- Smith PA, Nordquist CD, Jackson TN, Mayer TS, Martin BR, Mbundo J, Mallouk TE (2000) Electric-field assisted assembly and alignment of metallic nanowires. *Appl Phys Lett* 77(9):1399–1401
- van de Snepscheut JLA (1985) Trace theory and VLSI design. In: Lecture notes in computer science, vol 200. Springer, Berlin
- Snider GS, Kuekes PJ (2003) Molecular-junction-nanowire-crossbar-based associative array. US Patent 6 898 098
- Snider GS, Robinett W (2005) Crossbar demultiplexers for nanoelectronics based on n-hot codes. *IEEE Trans Nanotechnol* 4(2):249–254
- Snider GS, Williams RS (2007) Nano/CMOS architectures using a field-programmable nanowire interconnect. *Nanotechnology* 18(3):1–11
- Snider GS, Kuekes PJ, Williams RS (2004) CMOS-like logic in defective, nanoscale crossbars. *Nanotechnol* 15(8):881–891
- Snider GS, Kuekes PJ, Hogg T, Williams RS (2005) Nanoelectronic architectures. *Appl Phys A* 80(6): 1183–1195
- Soh C, Quate C, Morpurgo C, Marcus C, Kong C, Dai C (1999) Integrated nanotube circuits: controlled growth and ohmic contacting of single-walled carbon nanotubes. *Appl Phys Lett* 75(5):627–629
- Sotiriadis PP (2006) Information capacity of nanowire crossbar switching networks. *IEEE Trans Inf Theory* 52(7):3019–3032
- Spagocci S, Fountain T (1999) Fault rates in nanochip devices. *Proc Electrochem Soc* 98–19:582–596
- Spielman DA (1996) Highly fault-tolerant parallel computation. In: Proceedings of the 37th IEEE symposium on foundations of computer science (FOCS), Burlington, 14–16 Oct 1996, pp 154–163
- Srivastava N, Banerjee K (2004) Interconnect challenges for nanoscale electronic circuits. *TMS J Mater (JOM)* 56(10):30–31
- Stan MR, Franzon PD, Goldstein SC, Lach JC, Ziegler MM (2003) Molecular electronics: from devices and interconnect to circuits and architecture. *Proc IEEE* 91(11):1940–1957
- Strukov DB, Likharev KK (2005) CMOL FPGA: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology* 16(6):888–900
- Taubin A, Cortadella J, Lavagno L, Kondratyev A, Peeters A (2007) Design automation of real life asynchronous devices and systems. *Found Trends Electron Des Autom* 2(1):1–133
- Theis TN (2000) The future of interconnection technology. *IBM J Res Dev* 44(3):379–390
- Toffoli T (1984) Comment on: dissipation in computation. *Phys Rev Lett* 53(12):1204
- Tougaw PD, Lent CS (1994) Logical devices implemented using quantum cellular-automata. *J Appl Phys* 75:1818–1825
- Tour JM, Van Zandt L, Husband CP, Husband SM, Wilson LS, Franzon PD, Nackashi DP (2002) Nanocell logic gates for molecular computing. *IEEE Trans Nanotechnol* 1(2):100–109
- Tour JM, Cheng L, Nackashi DP, Yao Y, Flatt AK, St Angelo SK, Mallouk TE, Franzon PD (2003) Nanocell electronic memories. *J Am Chem Soc* 125(43):13279–13283
- Türel Ö, Lee JH, Ma X, Likharev K (2005) Architectures for nanoelectronic implementation of artificial neural networks: new results. *Neurocomputing* 64: 271–283
- Uchida K (2003) Single-electron devices for logic applications. In: *Nanoelectronics and information technology*. Wiley, Berlin, pp 425–443
- Unger SH (1969) Asynchronous sequential switching circuits. Wiley, New York
- von Hippel AR (1956) Molecular engineering. *Science* 123(3191):315–317
- von Neumann J (1956) Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: *Automata studies*. Princeton University Press, Princeton, pp 43–98

- Waingold E, Taylor M, Srikrishna D, Sarkar V, Lee W, Lee V, Kim J, Frank M, Finch P, Barua R, Babb J, Amarasinghe S, Agarwal A (1997) Baring it all to software: Raw machines. *Computer* 30(9):86–93
- Wang KL, Khitun A, Flood AH (2005) Interconnects for nanoelectronics. In: Proceedings of 2005 I.-E. international interconnect technology conference, San Francisco, 6–8 June 2005, pp 231–233
- Wang W (1990) An asynchronous two-dimensional self-correcting cellular automaton. Ph.D thesis, Boston University, Boston, MA 02215, short version: In: Proceedings of 32nd IEEE symposium on the foundations of computer science, San Juan, 1–4 Oct 1990. IEEE Press, pp 188–192, 1991
- Weeber JC, González MU, Baudrion AL, Dereux A (2005) Surface plasmon routing along right angle bent metal strips. *Appl Phys Lett* 87(22):221101
- Whitesides GM, Grzybowski B (2002) Self-assembly at all scales. *Science* 295(5564):2418–2421
- Mac Williams FJ, Sloane NJA (1978) The theory of error-correcting codes. North-Holland, Amsterdam
- Williams RS, Kuekes PJ (2001) Demultiplexer for a molecular wire crossbar network. US Patent 6 256 767
- Winfree E, Liu F, Wenzler LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. *Nature* 394(6693):539–544
- Wolf SA, Awschalom DD, Buhrman RA, Daughton JM, von Molnar S, Roukes ML, Chitchevkanova AY, Treger DM (2001) Spintronics: a spin-based electronics vision for the future. *Science* 294(5546):1488–1495
- Wong HSP, Frank DJ, Solomon PM, Wann CHJ, Wesler JJ (1999) Nanoscale CMOS. *Proc IEEE* 87(4):537–570
- Wood J, Edwards TC, Lipa S (Nov 2001) Rotary traveling-wave oscillator arrays: a new clock technology. *IEEE J Solid-State Circ* 36(11):1654–1665
- Worschech L, Beuscher F, Forchel A (1999) Quantized conductance in up to 20 μm long shallow etched GaAs/AlGaAs quantum wires. *Appl Phys Lett* 75(4):578–580
- Wu W, Jung GY, Olynick DL, Straznický J, Li Z, Li X, Ohlberg DAA, Chen Y, Wang SY, Liddle JA, Tong WM, Williams RS (2005) One-kilobit cross-bar molecular memory circuits at 30-nm half-pitch fabricated by nanoimprint lithography. *Appl Phys A* 80(6): 1173–1178
- Yamada T, Akazawa M, Asai T, Amemiya Y (2001) Boltzmann machine neural network devices using single-electron tunneling. *Nanotechnology* 12(1):60–67
- Yanagida T, Ueda M, Murata T, Esaki S, Ishii Y (2007) Brownian motion, fluctuation and life. *Biosystems* 88(3):228–242
- Yang T, Kiehl R, Chua L (2001) Tunneling phase logic cellular nonlinear networks. *Int J Bifurc Chaos* 11(12):2895–2911
- Zhirnov VV, Cavin RK, Hutchby JA, Bourianoff GI (2003) Limits to binary logic switch scaling – a gedanken model. *Proc IEEE* 91(11):1934–1939
- Zhong Z, Wang D, Cui Y, Bockrath MW, Lieber CM (2003) Nanowire crossbar arrays as address decoders for integrated nanosystems. *Science* 302(5649):1377–1379



## Cellular Computing

Christof Teuscher

Portland State University, Portland, OR, USA

### Article Outline

Glossary

Definition of the Subject

Introduction

Computation and Computability

The Machine as a Cell

Cellular Computing Hardware

The Cell as a Machine

Future Directions

Bibliography

### Glossary

**Cell** The biological cell is the smallest self-contained, self-maintaining, and self-reproducing unit of all living organisms. Various computing paradigms were inspired by the biological cell.

**Computation** Synonymous with information processing or also algorithm. Computations can, for example, be performed by abstract machines, real computer hardware, or biological systems. The abstract concept of the Turing machine separates the class of computable from the class of non-computable functions.

**Computing** The science that deals with the manipulation of symbols. Also refers to the processes carried out by real or abstract computers.

**Molecular computing** A subfield of cellular computing, where the molecules instead of the cell play a central functional role.

**Parallel computing** Parallel computing involves the execution of a task on multiple processors

with the goal to speed up the execution process by dividing up the task into smaller sub-tasks that can be executed simultaneously.

### Definition of the Subject

The field of *cellular computing* (abbreviated as CC) defines both a general computing framework and a discipline concerned with the analysis, modeling, and engineering of real cellular processes for the purpose of computation. The biological cell, discovered and coined by R. Hooke in 1665, is the smallest self-contained, self-maintaining, and self-reproducing unit of all living organisms. Its understanding and modeling is crucial both for the understanding of life and for the ability to use, control, and modify its complex biochemical processes to perform specific functions for the purpose of *in vivo* or *in vitro* computation. The cellular metaphor has inspired and influenced numerous both abstract computing models and *in silico* implementations, such as *cellular automata* (abbreviated as CA), *membrane systems* (or P systems), or *Field Programmable Gate Arrays* (abbreviated as FPGAs), with the main purpose to solve algorithmic problems in alternative ways. The cellular computing approach is appealing because the cell provides a convenient level of abstraction and functionality, is reasonably simple with enough abstractions, and can be used as a building block to compose more complex systems. On the other hand, a wide range of computational approaches are used to model and understand the functioning of the inter- and intracellular processes of biological cells on its various levels of complexity. The cell's biochemical processes can either directly be interpreted as computations or be modified for the specific purposes of computations through bioengineering methods. The hope in using biological cells as computing devices is (1) to ultimately go beyond complexity and fabrication limits currently encountered with traditional sili-

con circuits, (2) to embed computing capabilities into living organisms for autonomous therapeutic and diagnostic applications, and (3) to realize integrated biological sensing applications that use living organisms, such as plants and bacteria, as intelligent sensors.

## Introduction

The English naturalist Robert Hooke is commonly accredited for the discovery of the cell (Alberts et al. 1994) and the coining of its term, which he borrowed from the Latin word *cella*, that designates a small chamber. Hooke's work was published in 1665 in his book *Micrographia* (Hooke 1665) which contained observations he made with a very rudimentary microscope of dead cork cells. Since then, tremendous discoveries have been made around the cell, ranging from the pioneering work of German anatomist Walther Flemming (1880; Paweletz 2001) on cell division (mitosis) to the discovery of the DNA double helix structure by Watson and Crick in 1953 (Watson and Crick 1953). Nowadays, a sheer endless amount of knowledge and data on cells is available, originating from genetics, biochemistry, and molecular dynamics research. Nevertheless, the computational modeling at the molecular level of complete cells has been notoriously difficult because of the enormous complexity of the real cell, a lack of qualitative understanding, and the large number of unknowns in the models.

With the advent of modern computer science in the 1940s under pioneering fathers such as John von Neumann and Alan M. Turing, biology and computer science started to grow closer together. Biologists increasingly used computers to analyze experimental data and to model real biological organisms, while computer scientists began to draw inspiration from biology with the essential goal to engineer better and more "lifelike" machines. Using computer science, mathematics, statistics, machine learning, and artificial intelligence as tools to solve real problems in biology is commonly known under the term *bioinformatics* (Baldi and Brunak 2001), while *computational biology* is

more concerned with the exploration and discovery of new knowledge, for example, by testing specific hypotheses by means of computer models. *Systems biology* (Kitano 2001), on the other hand, focuses on the system level and how the participating biological components interact. This is typically done in a combined approach, which involves theory, computational modeling, and experiments.

Living organisms are complex systems exhibiting a range of desirable characteristics, such as evolution, adaptation, and fault tolerance, that have proved difficult to realize using traditional engineering methodologies. Drawing inspiration from biology to design better and both more "lifelike" machines and algorithms is generally known under the term *biologically inspired computer science* (Forbes 2004; Mange and Tomassini 1998; Sipper 2002) (also *nature-inspired*, *biomimetic*, or *biologically motivated*). Typically, this approach involves simulations, algorithms, and classical silicon-based circuits, but not living matter. The goal is not to copy nature, but to solely draw inspiration from it, while the field of *artificial life* (Langton 1995) is more concerned with understanding life by building it. Figure 1 illustrates the various possibilities to implement cellular models that range from realistic to abstract.

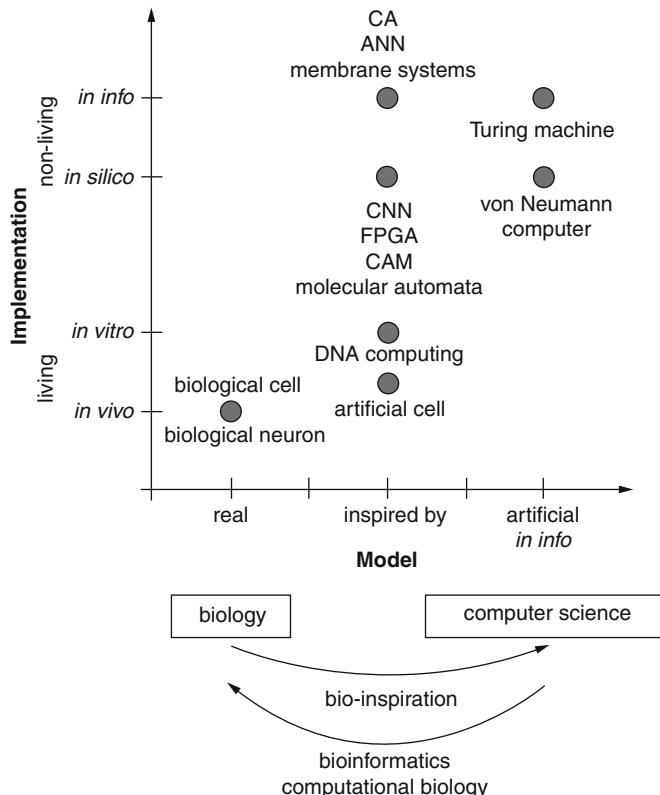
The broad and very interdisciplinary field of *cellular computing*, in its natural and artificial dimensions, defines both a:

- General computing framework (Sipper 1999)
- Discipline concerned with the analysis, modeling, and engineering of real cellular processes for the purpose of computation (Amos 2004)

As opposed to pure bioinspired computing, cellular computing thus also includes the use of real biological cells for information processing. Cellular computing encompasses abstract theory, simulations, models, and experiments and includes both the paradigms of "the cell as a machine" and "the machine as a cell." Sometimes, cellular computing is used interchangeably with *molecular computing* (Sienko et al. 2003); however, while molecular computing is concerned with the

### Cellular Computing,

**Fig. 1** Illustration of the various possibilities to implement cellular models that range from realistic to abstract. Drawing inspiration from biology to build better computing machines is known under the term *bioinspiration*, while using machines to address problems in biology is known as *bioinformatics* and *computational biology*



information processing in which molecules play a central functional role, cellular computing looks at the cell as the functional building block for more complex systems. Naturally, molecules are involved in cellular information processing as well, so we therefore consider molecular computing as a subset of cellular computing.

The birth hour of *connectionism* in the 1940s can be considered as the first occurrence of looking at cells as automata and modeling them by digital systems. The well-known McCulloch-Pitts neuron (McCulloch and Pitts 1943) simulated a wealth of research in neural information processing and modeling that continues with increasing activity until today (Arbib 1995). In 1948 (but first published in 1969, reprinted in Turing (1992)), Alan Turing proposed his own connectionist ideas (Teuscher 2002; Turing 1969), which were highly influenced by the digital systems paradigms. His neurons were simple NAND (not AND) logical

gates, which would thus allow in principle to compute any logical function, provided enough neurons are available and that they can be interconnected in arbitrary ways.

The concept of molecular automata and of cells seen as information processing devices appeared around the 1960s. In their studies of the control of gene expression and the synthesis of proteins, Jacob and Monod (1961; Monod and Jacob 1961), Davis (1961), and others emphasized deterministic operations at the level of DNA, RNA, and ribosomes. Early work on protein synthesis, for example, by Warner et al. (1962), suggested that ribosomes move along the messenger RNA, not unlike the head reading the tape of a Turing machine. In 1961, Pattee (1961) used the concept of the Turing machine computation to explain the generation of growing macromolecular sequences and rhetorically asked in the title of a section of his paper “Can Molecules Compute?”

Stahl and Goheen (1963) further pioneered the idea of seeing the cell as an information processor and Turing machine. They considered enzymes as computational primitives whose operations are simulated by a Turing machine and are therefore computable in the formal sense of the term (Davis 1958). At about the same time, Sugita considered some aspects of molecular systems by using a logical circuit equivalent (Sugita 1961, 1963; Sugita and Fukuda 1963). He states:

The mechanism of life shall be clarified by using electronic analogues, because the chemical system composing the living organism is nothing but a complicated network of rate processes from the chemical or physical point of view. Simulation of such a network may be done either by using an analogue computer or a digital differential analyzer, as has been done at the laboratory[...]. A digital system composed of some chemical reactions may play the role of a controlling computer of the living chemical plant, and the reactions having analogue behavior may be controlled by that computer as well as by analog control systems which can be seen in primitive control. (Sugita 1961).

Schmitt (1962) discussed the biological memory of macromolecular automata in his 1962 book, while Feynman (1960) elaborated in his famous talk “There’s Plenty of Room at the Bottom” on the possibility to compute with atoms, molecules, and cells. Feynman was more focused on the miniaturization of the computer by storing and processing information at atomic levels, but his ideas were pursued by many others later (e.g., Sienko et al. 2003; Bennett 1982; Conrad and Liberman 1982) and ultimately led to what we know today as *nano- and molecular electronics* (Cuniberti et al. 2005) but also greatly stimulated the earlier cellular computing research.

Michael Conrad, a pioneer in biologically inspired, molecular, and cellular computing, as well as many others, observed that living organisms process information in a very different way than digital computers (Conrad 1989). Organisms typically exploit the inherent physical properties of the matter they are made of, while digital computers are based on multiple levels of abstractions, which are typically far away from the physical substrate and do not directly exploit any of its

specific physical characteristics for the purpose of more efficient computation. Cellular computing, both in its natural and artificial dimensions, tries to address these issues by drawing inspiration from real cells and by using them for the purpose of computation.

The principal reason for the increasing interest in cellular computing paradigms seems twofold. Firstly, Moore’s Law (1965) has dominated the progress of electronic circuits in the past decades, but fundamental limits of silicon-based technology now begin to become serious showstoppers (Kish 2002). The hope is that cellular computing will help to go beyond such limits by using alternative computing paradigms inspired by the cell or by using real cells, molecules, and atoms as information processing devices. Secondly, despite lots of progress, Turing’s hope that “machines will eventually compete with men in all purely intellectual fields” (Turing 1950) is far from accomplished. Biological systems outperform their silicon counterparts in various areas, such as with their ability to adapt, to recognize objects, or to self-repair. Again, by using cellular computing paradigms, the hope is that such bioinspired approaches will ultimately allow to go beyond existing limits.

## Computation and Computability

Both from the perspective of computer science and biological organisms, a central question is what kind of operations a given system can perform, what kind of problems it can solve, and how efficiently it can do this. Despite their ubiquitousness and seemingly endless power, there are clear and fundamental limitations – which are too often ignored unfortunately – to what computers can do (Harel 2003; Lloyd 2000). Two fields of computer science are relevant in this context: *computability theory* (Davis 1958; Minsky 1967) deals with what problems can be solved, while *computational complexity theory* (Papadimitriou 1994) deals with how efficient they can be solved.

Since Alan M. Turing’s seminal paper on the abstract concept of the *Turing machine* (abbreviated as TM) (Turing 1937), *computability*

is well defined, and we generally know what classes of problems can be solved by which classes of machines. The *Chomsky hierarchy* (Sipser 2006), for example, partitions the formal languages into classes of different expressive powers. Each class can be associated a class of abstract automaton, which is able to generate and recognize the corresponding language. At the top of this hierarchy sits the Turing machine, which is able to generate the most expressive class of languages. Along with other abstract formalisms and machines, a Turing machine is able to carry out any effective computation, i.e., it can simulate any other machine capable of performing a well-defined computational procedure. In this context, “effective” is synonymous with “mechanical” or “algorithmic.” In its original form, the *Church-Turing thesis* states that there is an equivalence between Church’s  $\lambda$ -calculus and Turing machines. Since then, numerous more general and stronger forms of the thesis have emerged. For example, David Deutsch wrote: “Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means” (Deutsch 1985, p. 99). Despite claims from “hyper-computationalists” (e.g., see Davis (2004), Teuscher and Sipper (2002) for an overview of this debate), the fact is that no physically realizable device, which thus necessarily operates on finite resources and time, was ever able to compute functions (in the formal sense of the definition) that a Turing machine could not compute, which means that the Church-Turing thesis has not been disproved as of today.

It is obviously an interesting question to ask how and what biological systems “compute,” what their limits are, and how efficient they can solve problems. Biological organisms naturally process some information in some way, but how – if at all – can these processes be described in an algorithmic manner? What kind of abstract automata are useful to describe the many computational mechanisms of a biological cell? Paton describes four general types of computations in Paton (1993) under which the information processing capabilities of a cell can be seen:

- *Sequential*. The most common form of machine, which usually has a central processor and a global memory. The instructions are carried out serially, such as by a Turing machine.
- *Parallel*. Several processors jointly execute a task. A wide range of parallel computing paradigms exist.
- *Distributed*. Similar to parallel computation, but the processors are usually located at larger distances.
- *Emergent*. The global behavior of such a computation emerges from the local interactions between the processors. This form is typical of self-organizing systems, cellular automata, and neural networks.

As he states, “[c]omputational models of the cell can utilize any of these views”; however, most processes are best described as a parallel distributed computing paradigm with emergent properties. Paton also points out the risk of trying to force biological systems to “fit” into a certain computational model. Important details might be ignored and one is forced to think within the existing computing paradigms. We can conclude that abstract and formal models of computation have been very useful in describing the information processing capabilities of biological systems; however, not every formalism is well suited for such descriptions, even if his expressiveness matches that of the corresponding biological system. For example, modeling the highly parallel processes of a biological cell by a Turing machine is possible since parallel machines are not more expressive than sequential machines; however, the sequential paradigm does not offer a convenient and useful model in this context.

## The Machine as a Cell

In 1999, Moshe Sipper, used the term “cellular computing” in a broad sense to introduce a general computing philosophy and framework. He hypothesized that

[t]his philosophy promises to provide new means for doing computation more efficiently – in terms of speed, cost, power dissipation, information storage, and solution quality. Simultaneously, cellular computing offers the potential to addressing much larger problem instances than previously possible, at least for some application domains. (Sipper 1999, p. 18)

Cellular computing consists of three essential and admittedly very general principles:

- Simplicity
- Vast parallelism
- Locality

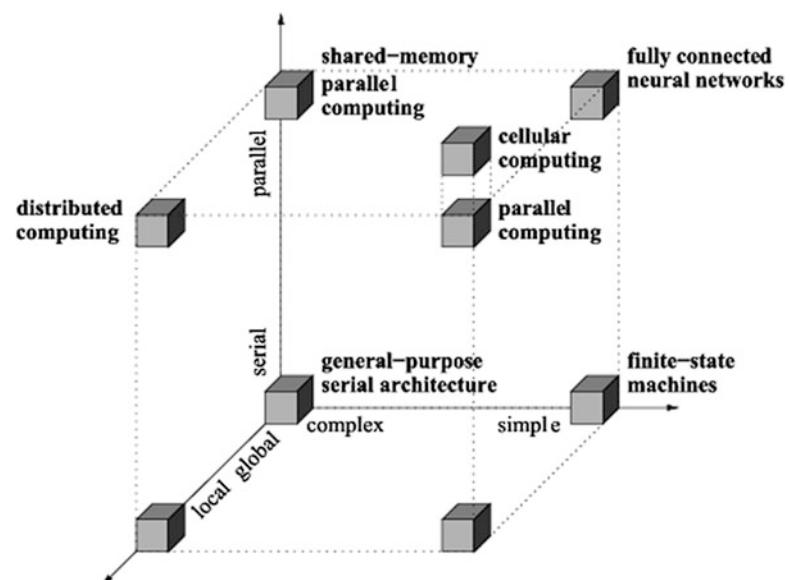
Figure 2 illustrates these three principles. While most general-purpose processors are universal machines in the sense of a Turing machine, the concept of cellular computation expects the complexity of the basic unit, the cell, to be significantly lower, which is characterized by *simplicity*. While “simplicity” is not well or formally defined in this context, Sipper provides the example of an AND gate as being simple, while a Pentium processor is not. Most of today’s computers are based on a multi-core processor, which offers several cores that allow to process information in parallel. This trend – generally considered as an elegant way to keep up with Moore’s law – is expected to continue in the next few years. As of November 2006, the Top500 supercomputing list ([Top500 supercomputing sites](#)) is

headed by a Blue Gene/L server with 131,072 processors. Cellular computing wants to be orders of magnitude above this level of complexity in terms of the number of cells (i.e., the processors) involved. Sipper uses the term *vast parallelism* to characterize systems where the number of cells is measured by the exponential notation  $10^x$ . For example, an *Avogadro-scale system* would involve an Avogadro number of cells, i.e.,  $10^{23}$ . As we scale down electronic circuits to molecular and atomic dimensions and make use of self-assembling techniques, assemblies of this complexity become increasingly feasible. It is unlikely that systems of such complexity will ever be built on macroscopic levels. The last principle concerns the interactions among the cells: cellular computing is based on *local interactions* only, where no cell has a global view or control over the entire system. Obviously, the three principles are related and the cell’s simplicity, for example, helps to achieve vast parallelism. Similarly, the vast parallelism makes it hard to implement non-local interactions. These interactions are illustrated in Fig. 2 by the “computing cube.” Changing a single term in the “equation” simple + vastly parallel + local results in a different computing paradigm.

Compared to traditional parallel computing, which typically makes use of a rather small number of complex nodes, the cellular computing framework is based on a vast number of simple

### **Cellular Computing,**

**Fig. 2** The “computing cube.” Illustration of the cellular computing paradigm: simple + vastly parallel + local (Redrawn from Sipper (1999) and extended). Cellular computing has been placed further along the parallelism axis to emphasize the “vastness” aspect

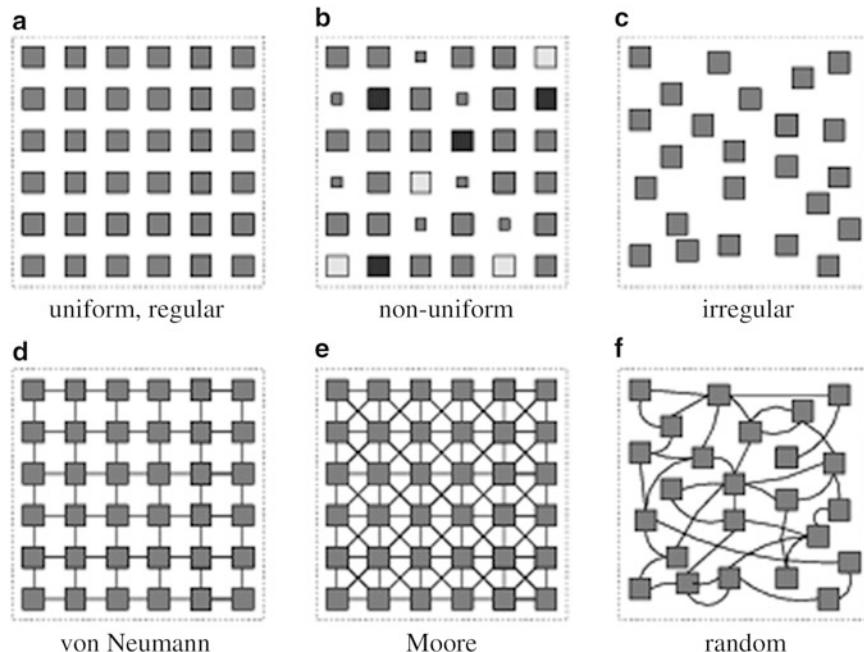


nodes. Also, parallel computers often have non-local interconnect topologies or even special nodes, which have some sort of global control over all other nodes, which cellular computing does not allow. Clearly, while it is already very challenging to find enough parallelism in applications to exploit the parallelism offered by traditional parallel computers, this problem will only be aggravated – and further moved to the programmer and the software – for fine-grained cellular computers. It is commonly accepted that such machines will likely only be more efficient for very specific problem domains which already offer plenty of inherent parallelism, such as image processing. Obviously, one can always implement some form of serial computing paradigm on a parallel machine (e.g., implementing a Turing machine on the *Game of Life* (Rendell 2002)), but that completely defeats the purpose of such a machine.

This very general framework naturally allows for considerable flexibility in the models. Besides the three basic principles, a number of other

important properties play a role, such as the detailed local interconnect topology, the cell's arrangement in space, the mobility, the uniformity, and the cell's behavior (Sipper 1999):

- *Arrangement.* While a majority of abstract cellular models do not consider explicit physical locations and spatial dimensions (e.g., the cell dimension of a cellular automata is not generally considered), more biologically and physically plausible models do. Regular grids in 2D (Fig. 3a) or 3D with rectangular or other geometries are most common, but irregular (Fig. 3c) or even nonrigid grids are possible. In addition, one needs to specify the boundary conditions of the grid (e.g., fixed or wrapped around) and whether the grid has a finite or infinite dimension (open boundary).
- *Interconnect topology.* Cellular computing is based on local interactions only, but there are many possibilities to interconnect neighboring cells in a non-global way. In case of a regular rectangular grid, purely local four- (von



**Cellular Computing, Fig. 3** Illustration of different cell arrangements and interconnect topologies. (a) Regular and uniform; (b) regular and nonuniform, different shadings and sizes indicate different cell programs; (c) irregular and

uniform; (d) regular, uniform, von Neumann neighborhood; (e) regular, uniform, Moore neighborhood; (f) irregular, uniform, random neighborhood

Neumann; see Fig. 3d) and eight-cell neighborhoods (Moore; see Fig. 3e) are most common (Toffoli and Margolus 1987). However, interconnect topologies that are completely random or that have the small-world (Watts and Strogatz 1998) or power-law (Albert and Barabasi 2002) property are also possible since no cell has a global view over the whole system.

- *Mobility.* Cells can be either physically mobile (e.g., for ant colony optimization (Dorigo and Stützle 2004)) or immobile. Alternatively, their “program” can also move from one cell to another (e.g., for self-replicating loops (Mange et al. 2004)) without a physical relocation of the cell.
- *Uniformity.* Both the cell type (i.e., its state and behavior) and its interconnections with the neighboring cells can be uniform (Fig. 3a) or nonuniform (Fig. 3b) to some degree. Most cellular paradigms use uniform cells and interconnection schemes.
- *State and behavior.* Cells can have internal states that are either discrete or analog. Based on its internal state and the internal states read from the neighboring cells, the cell changes its own state according to some algorithm. This can be as simple as a lookup table (abbreviated as LUT) or a mathematical function or as complicated as a small program. The cells are usually updated in discrete time steps, in either a synchronous or asynchronous way. A continuous behavior is also possible, for example, if the cell’s behavior is guided by differential equations or analog electronics. Finally, both the behavior and the cell’s updating can be deterministic or nondeterministic.

Cellular computing does not claim to be a faithful model of biological cells; however, the framework is general and flexible enough to allow for biologically plausible models as well. As Toffoli and Margolus point out in their book (Toffoli and Margolus 1987) with regard to cellular automata, the generality and flexibility of cellular approaches also come with a cost. Instead of only a few variables as typical in traditional computing models, the cellular computing paradigm

uses at least one variable (or “program”) per cell. With the vast number of cells involved, the challenging task is thus to find the right rule(s) which allows to solve a given task. As already stated above, the cellular computing paradigm is ideally suited for tasks that already contain a high degree of parallelism and require local interactions only.

A large number of computing models have drawn inspiration from real cells. While a very large part of the work is concerned with modeling neural cells, a number of researchers were also interested in developmental and other aspects. For example, Astor and Adami (2001) used a developmental model for the growth of neural networks. Each neuron is an autonomous entity, which behavior is only determined by its genetic information contained within the cell. An artificial chemistry is also used to model chemicals and substrates. Fleischer and Barr (1994) introduced a model of multicellular development that combines elements of the chemical, cell lineage, and mechanical models of morphogenesis pioneered by Turing (1952), Lindenmayer (1968), and Odell (1981). Cell migration is an important aspect of development, especially of the neural development. Compared to previous models, Fleischer’s approach allows cells to move freely within the environment. Other pioneers of morphogenetic, developmental, and growing neural networks models, which necessarily involve cells in various forms, were, for example, Dellaert and Beer (1994), Rust (1998), Eggenberger (1997), and Gruau (1994). More abstract models that deal with the self-maintenance, the self-reproduction, and the self-replication came, for example, from Ono and Ikegami (2000), Varela et al. (1974), Langton (1984), Mange et al. (2004), and various others.

In the following, we will give three more detailed examples of cellular computing models.

### **Example: Cellular Automata**

*Cellular automata* (abbreviated as CA) were originally conceived by Ulam and von Neumann in the 1940s to provide a formal framework for investigating the behavior of complex, extended systems (von Neumann 1966). CAs are dynamical systems in which space and time are discrete.

In its basic version, a cellular automaton (Toffoli and Margolus 1987; Wolfram 1984) consists of a regular grid of cells, each of which can be in one of a finite number of  $k$  possible states, updated synchronously (or asynchronously) in discrete time steps according to a local, usually identical (uniform, otherwise nonuniform) interaction rule. The state of a given cell is determined by the previous states of a specified neighborhood of cells. Figure 3d and e shows two typical local CA neighborhoods, the von Neumann and the Moore neighborhood.

In CAs and many other cellular computing systems, there is a confound between the data and the computing devices, i.e., the structure, which process it. The abstract Turing machine and the von Neumann computer architecture make a clear distinction between these two. The Turing machine stores data on the tape; however, a CA cell can both store and process data and the machine can even extend or modify itself. Although it has been shown that elementary CAs are universal (Cook 2004), this is more of theoretical than practical interest since most CAs that solve some real problem cannot – and do not need to – perform any universal computation.

One of the main challenges with cellular automata is to find the cell's rule(s) which allows to obtain a global behavior from the local interactions only. For example, given a specific task, say, detecting the contours of an object in an image, what are the individual cell rules (either uniform or nonuniform) that allow to successfully solve that task for a wide range of inputs? The large – or even vast – number of cells involved and the local interactions make this problem very challenging. Therefore, most cellular automata are either programmed by hand - which becomes infeasible for larger systems and complex problems, or the rules are, for example, found by means of evolutionary algorithms (Capcarrere 2002; Sipper 1996, 1997; Sipper and Ruppin 1997).

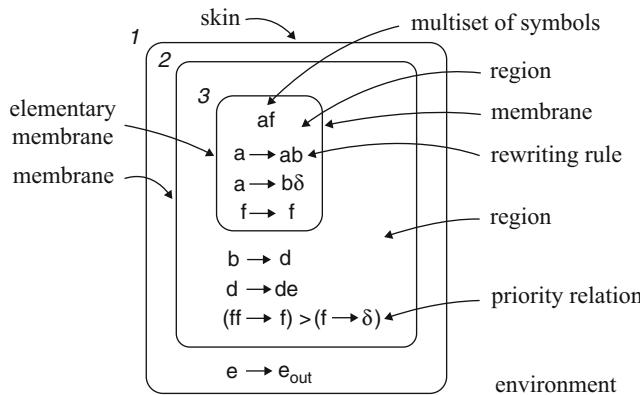
### Example: Membrane Computing

*Artificial chemistries* (Dittrich et al. 2001) are man-made systems that are a very general formulation of abstract systems of *objects* that follow

arbitrary *rules of interaction*. More formally speaking, an artificial chemistry essentially consists of a set of *molecules*  $S$ , a set of *rules*  $R$ , and a definition of the *reactor algorithm A* that describes how the set of rules is applied to the set of molecules. This very broad paradigm, inspired by biochemical systems, allows to describe many complex natural and artificial systems by means of simple rules of decentralized, local, and parallel interactions. The chemical paradigm thus also fits in the bigger framework of cellular computing.

In 1998, George Paun initiated *membrane computing* (also *P systems*) (Paun 2000; 2002) as an abstract computational model afar inspired by biochemistry and by some of the basic features of biological membranes. Membrane computing makes use of a hierarchical membrane structure that is similar to the structure used in the chemical abstract machine as proposed by Berry and Boudol (1992). The evolution rules that transform the multisets are inspired by *Gamma systems* as proposed by Banâtre et al. (1988).

A typical membrane system consists of cell-like membranes placed inside a unique “skin” membrane. Multisets of *objects* – usually strings of symbols – and a set of *evolution rules* (or *reaction rules*) are then placed inside the regions delimited by the membranes. As an example, a simple membrane system is depicted in Fig. 4. The evolution between system configurations is done nondeterministically but synchronously by applying the rules in parallel for all objects able to evolve. A sequence of transitions in a membrane system is called a *computation*. A computation *halts* when a halting configuration is reached, i.e., when no rule can be applied in any region. In classical membrane systems, a computation is considered successful if and only if it halts, but other interpretations of inputs and outputs are possible. A great feature of membrane systems is their inherent parallelism, which allows to trade space for time, for example, to solve computationally hard problems. Using membrane division, it has been shown that NP-complete problems can be solved in polynomial and even linear time (Paun 2002). Most membrane systems are also Turing universal.



**Cellular Computing, Fig. 4** An example of a membrane system as given in Paun (2002). The system generates  $n^2$ , where  $n \geq 1$ , where  $n$  is the number of steps before the first application of the rule  $a \rightarrow b\delta$ . The rules are applied

A wide variety of different membrane system flavors exist (see [The P systems web page](#) for the latest publications), which have been applied to applications ranging from modeling spiking neural systems to economics.

### Example: Amorphous Computing

In a 1996 white paper, Abelson et al. first described the *amorphous computing* (abbreviated as AC) framework (Abelson et al. 2000). Amorphous computing is the development of organizational principles and programming languages for obtaining coherent global behavior from the local cooperation of a myriad of unreliable parts that are interconnected in unknown, irregular, and time-varying ways. In biology, this question has been recognized as fundamental in the context of animals (such as ants and bees) that cooperate and form organizations.

Amorphous computing asks this question in the field of computer science and engineering. Using the biological metaphor, the cells cooperate to form a multicellular organism (also called *programmable multitude*) under the direction of a genetic program shared by all members of the colony. Again, the cellular computing philosophy applies particularly well to the amorphous computing framework. The properties of an amorphous computer can be summarized under the following assumptions, where processor is used synonymous with cell:

synchronously and in a maximum parallel manner, i.e., all rules that can be applied have to be applied. If multiple rules can be applied, one is picked nondeterministically

- Individual processors are identical and mass produced.
- Processors possess no a priori knowledge of their location, orientation, or neighbors' identities.
- Processors operate asynchronously although they have similar clock speeds.
- Processors are distributed densely and randomly.
- Processors are unreliable.
- Processors communicate only locally and do not have a precision interconnect.
- The processors are arranged on a 2D surface or in 3D space.
- It is assumed that the number of particles is very large.
- The communication model assumes that all processors have a circular broadcast of approximately the same fixed radius (large compared to the fixed size of a processor) and share a single channel.

The biggest challenge in amorphous computing – as with the majority of cellular computing paradigms – is how to program the individual cells in order to obtain a consistent and robust global behavior. The amorphous computing research community came up with several automated approaches, which allow to compile a global system description into local interactions. Examples are Nagpal's *origami shape language* (Nagpal 2001), Coore's *growing point language* (Coore 1999), and Butera's *process fragments* (Butera 2002).

## Cellular Computing Hardware

Cells have inspired not only abstract computing models but also real hardware, which is commonly called *biologically inspired hardware* (Sipper et al. 1997a). On the other hand, both specialized and nonspecialized computers have been used for the acceleration and the faithful simulation of many biological processes related to the cell, particularly within the field of *bioinformatics* (Baldi and Brunak 2001). A biological cell is a massively parallel system, where thousands of highly complex processes run concurrently. Simulating these processes on a sequential von Neumann computer will obviously not be very efficient and will seriously limit the performance. Specialized hardware, which offers a high level of inherent parallelism, has thus attracted considerable attention in this field of cellular computing.

*Cellular automata machines* (abbreviated as CAMs) are specialized machines with the goal to very efficiently simulate cellular automata. For example, Hillis' connection machine (Hillis 1985) was the first milestone in that endeavor in the early 1980s. The processors were extremely simple and there was a strong emphasis on the interconnectivity, but not particularly on local topologies because the machines were targeted for supercomputer applications. Toffoli and Margolus' CAM machines (Toffoli and Margolus 1987) represent other examples of highly specialized machines. Nowadays, because of their inherent fine-grained parallelism, reconfigurable circuits, such as *Field Programmable Gate Arrays* (abbreviated as FPGAs) (Gokhale and Graham 2005; Villasenor and Mangione-Smith 1997), are ideal candidates to simulate cellular systems. For example, Petreska and Teuscher (2004) proposed a very first hardware realization of membrane systems using reconfigurable circuits, which was based on a universal and minimal membrane hardware component that allowed to very efficiently evolve membrane systems in hardware. Another example of cellular hardware are *cellular neural network* (abbreviated as CNN) (Chua and Roska 2002) chips, which are massively parallel analog processor arrays. The CNN approach is extremely

powerful for specific applications, such as real-time image and video processing.

In the following, we will provide three more detailed examples of typical and biologically inspired hardware, which is based on cellular paradigms and metaphors.

### Example: Embryonics

Biological systems grow, live, adapt, and reproduce characteristics that are not truly encompassed by any existing computing system. Sipper et al. (1997a) proposed the *POE model* of bioinspired hardware systems, which stands for *phylogeny* (abbreviated as P), *ontogeny* (abbreviated as O), and *epigenesis* (abbreviated as E). This model allows to partition the hardware space along three axes, which, in very simplified terms, correspond to evolution (P), development (O), and learning (E). Each axis thus represents a different form of adaption and organization on a different time scale. The ultimate goal is machines that combine all three forms of adaption, so-called POEtic machines (Tyrrell et al. 2003).

The *embryonics project* (Mange and Tomassini 1998; Mange et al. 2000) (embryonics stands for embryonic electronics) is an attempt to design highly robust integrated circuits, endowed with the properties usually associated with the living world: self-repair (cicatrization) and self-replication. In this context, self-replication allows for a complete reconstruction of an organism by making a copy of itself. The approach draws inspiration from the basic processes of molecular biology and from the embryonic development of living beings, which is represented by the ontogenetic axis in the POE model. An embryonic circuit is based on a finite but arbitrarily large two-dimensional surface of silicon. This surface is divided into rows and columns, whose intersections define the cells. Since such cells (i.e., a small processor and its memory) have an identical physical structure (i.e., an identical set of logic operators and connections), the cellular array is homogeneous. Embryonics largely draws inspiration from the following biological features, which the majority of living beings (at the exception of unicellular organisms) share: (1) multicellular organization, (2) cellular division, and (3) cellular

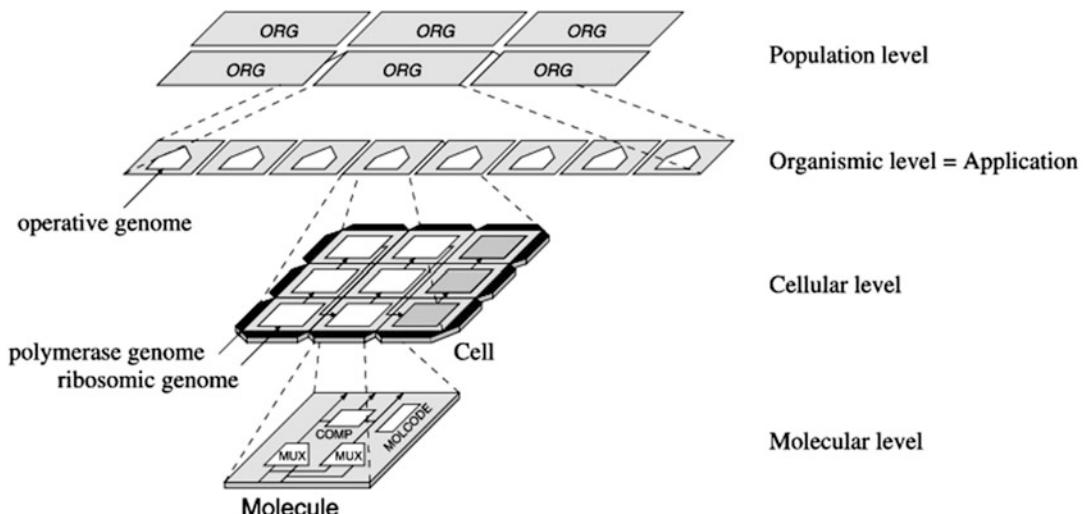
differentiation. Similar to biological cells, each embryonic cell is “universal” in the sense that it contains the whole of the organism’s genetic material, the genome. This feature allows to implement self-repair in an elegant way since each cell is potentially able to replace every other cell within an embryonic organism. The final embryonics architecture makes use of four hierarchical levels of organization:

- *Molecule*. The basic primitive of the system, which essentially consists of a multiplexer associated with a programmable connection network. The multiplexer is duplicated to allow the detection of faults. The logic function of each molecule is defined by its molecular code, the *MOLCODE*.
- *Cell*. A finite set of molecules makes up a cell, which is essentially a processor with an associated memory.
- *Organism*. A finite set of cells makes up an organism (synonymous with application), an application-specific multiprocessor system.
- *Population*. The organism can itself self-replicate, giving rise to a population of identical organisms.

Figure 5 illustrates these four hierarchical levels. As a showcase of bioinspired systems and hardware, a complete embryonic machine and application has been implemented in real hardware by using *Field Programmable Gate Arrays* (abbreviated as FPGAs) (Gokhale and Graham 2005) to implement the basic molecule (Tempesti et al. 2002). The main application, which used several thousand molecules, was a fault-tolerant electronic watch, the BioWatch (Stauffer et al. 2001). The project illustrates how biological organisms, and cells in particular, can help to build alternative computing machines with properties not truly encompassed by any existing architecture.

#### Example: Field Programmable Gate Arrays

A *Field Programmable Gate Array* (abbreviated as FPGA) (Gokhale and Graham 2005) is a reconfigurable device (Villasenor and Mangione-Smith 1997) which is based on a set of *configurable logic blocks* (abbreviated as CLBs) that are interconnected by a programmable interconnect fabric. The logic blocks of most FPGAs are rather simple, consisting typically of some combinational logic (e.g., a 4-input lookup table)



**Cellular Computing, Fig. 5** The four-level hierarchy of the embryonics landscape. Each level is configured by a part of the genome, which in parallel into all cells once their cell membranes have been constructed by a

mechanism similar to cellular division. After this process, the cells differentiate according to their spatial location in the organism (Mange et al. 2000)

and some sequential logic (e.g., a flip-flop). Some CLBs offer additional memory too. The interconnect fabric is typically organized in a hierarchical manner with mostly local interconnections between CLBs but with the possibility to establish limited long(er)-distance connections as well. Sophisticated design tools, compilers, and hardware description languages allow to map essentially any digital circuit onto such a reconfigurable chip, provided enough CLBs are available. In accordance with the cellular computing framework, FPGAs are based on simple basic building blocks (the CLBs) and are locally interconnected (in the sense that no cell controls the entire system), and today's high-end circuits contain hundreds of thousands of CLBs. FPGAs tend to be slower than full-custom VLSI circuits but offer more flexibility at a lower price. The application of FPGA is obviously most interesting where their massive parallelism can be fully exploited, for example, for image processing, neural networks (Ormondi and Rajapakse 2006), DNA sequence matching, or cellular automata. Sipper et al., for example, used FPGAs for their firefly CA machine (Sipper et al. 1997b), which were also used to implement online evolution algorithms to automatically find solutions to the synchronization task for CAs (Sipper 1997; Das et al. 1995).

### Example: The Cell Matrix

The *Cell Matrix* ([Cell matrix Corporation](#); Durbeck and Macias 2001; Macias 1999) is an extremely fine-grained and universal hardware architecture, not unlike the structure of an FPGA. The reconfigurable device consists of a homogeneous collection of cells that are interconnected in a nearest-neighbor scheme (e.g., von Neumann or other neighborhoods). Each cell performs very basic operations only that are implemented by a simple lookup table. Unlike traditional reconfigurable devices that are controlled by external systems, the Cell Matrix is self-configurable, i.e., each cell within the matrix can reconfigure other cells. In order to accomplish this, each cell can operate independently in one of two modes: (1) the D mode or (2) the C mode. In the D mode (processing mode), incoming data is processed by the cell's internal lookup table and used to generate

the output. In the C mode (configuration mode), on the other hand, incoming data is used to rewrite the cell's lookup table. A cell can therefore modify one of its neighbors by placing it in the C mode and by writing the data into the lookup table. Since a cell's lookup table governs the ability to control a neighboring cell's mode, a cell X, for example, can configure a neighboring cell Y in such a way that Y will subsequently configure a third cell Z. By combining cells to so-called supercells, hierarchical designs of almost any complexity can be realized. The small groups of cells then interact with nearby groups to achieve higher functionalities, which are used to perform more complex functions and so on. The Cell Matrix architecture was extended in 2004 by Petraglio in his Ph.D. thesis (Petricaglio 2004).

### The Cell as a Machine

“The brain computes! This is accepted as a truism by the majority of neuroscientists engaged in discovering the principles employed in the design and operation of nervous systems. What is meant here is that any brain takes the incoming sensory data, encodes them into various biophysical variables, such as the membrane potential or neural firing rates, and subsequently performs a very large number of ill-specified operations, frequently termed computations, on these variables to extract relevant features from the input” (Koch 1999, p. 1). Further on, Koch continues by elaborating that any physical process, which transforms variables, can in principle be thought of as computation, as long as it can be described by some mathematical formulation. As also noted by Conrad (1989), living organisms process information in a very different way than digital computers and typically exploit the inherent physical properties of the matter they are made of. The direct, efficient, and sophisticated ways to process information in cells, molecules, and atoms are the results of billions of years of evolution.

In general, computation in living systems is more a question of what level is being analyzed and through which glasses the analyzer looks. A cell has thousands of information processing

mechanisms; some are easy to map to existing computing paradigms, while others are far away from any existing model and can therefore only be captured by new paradigms. If we want to use a biological cell as a computing device, we essentially have two possibilities:

- *Interpretation.* Interpret existing biochemical cellular mechanisms as computation.
- *Modification.* With the advances in biotechnology and bioengineering in the last decade, biochemical processes at all organization levels of a cell can increasingly be modified for the specific purposes of computation.

For example, DNA can be described in computational terms rather easily, and the DNA strand interpreted as data, which is being read by ribosomes, is a fairly straightforward natural analog to the Turing machine (Warner et al. 1962). But this undertaking becomes more interesting if the analogy can be reversed and the DNA is used to realize Boolean *in vivo* or *in vitro* logic gates in a well-controlled manner.

As stated in the introduction, the concept of molecular automata and cells seen as information processing devices appeared around the 1960s, and the field has seen impressive developments since then. The field is vast and we shall only give a very shallow taste here. The books by Amos (2004), Calude and Paun (2000), Paton et al. (2004), and Sienko et al. (2003) provide excellent overviews from the “cell as a machine” perspective on more recent work, while Paton (1993), for example, deals with some of the earlier concepts.

Since Adelman’s seminal work on *in vitro* DNA computation (Adelman 1994), this unconventional computing paradigm has played a very important role in cellular computation in both experiment and theory (Paun et al. 1998). DNA computation clearly outperforms traditional silicon-based electronics in terms of speed, energy efficiency, and information density. However, while it is rather easy to explore huge search spaces, it is often nontrivial to find the correct answer within all the generated solutions, which has somehow limited the practical applications. Other work has focused on RNA editing as a computational process, which offers another

interesting cellular paradigm for “biological software” (Landweber and Kari 1999). Later, Winfree, Weiss, and various others pioneered DNA-based and other *in vivo* logical circuits (Benenson et al. 2001; Seelig et al. 2006; Shapiro and Gil 2007; Weiss 2001). Based on deoxyribozymes, Stefanovic’s group has developed molecular automata and logic (Lederman et al. 2006; Stojanovic and Stefanovic 2003). Their circuits include a full adder, three-input logic gates, and an automaton that plays a version of the game tic-tac-toe. While such circuits serve as examples and proof of concepts, the group’s long-term goal is to use engineered molecules to control autonomous therapeutic and diagnostic devices. These research areas are fast moving and lots of novel results are to be expected in the next few years.

## Future Directions

“The simplest living cell is so complex that supercomputer models may never simulate its behavior perfectly. But even imperfect models could shake the foundations of biology” (Gibbs 2001). Gibbs further emphasizes that most attempts to create artificial life or to faithfully model biological systems suffered from a tremendous number of degrees of freedom. The high number of unconstrained system parameters can then lead to almost any desired behavior. Also, the models are often so complicated that they have a very limited ability to predict anything at all. Tomita believes that the study of the cell will never be complete unless its dynamic behavior on all levels is fully understood (Tomita 2001). He suggests that the complex behavior can only be understood by means of sophisticated computer models that allow to undertake whole cell simulations. Endy and Brent (2001) emphasize that “[p]ast efforts to model behavior of molecular and cellular systems over absolute time typically were qualitatively incomplete or oversimplified compared to available knowledge, and qualitatively incomplete in the sense that key numbers were unknown.” It is estimated that complete *E. coli* simulations could run, perhaps, on a single processor system by 2020.

But the future not only lies in more faithful cellular models and in the ability to tweak cells for the purpose of computation. The construction of realistic living cells in silico, thought intractable for a long time, has now become within reach, and so does the possibility to build artificial living cells. Several efforts with this specific goal in mind are currently under way in both Europe and the United States (see, e.g., Pohorille and Deamer 2002; Rasmussen et al. 2004; Szostak et al. 2001). The question is how and under which conditions simple life forms, which are able to self-maintain and self-replicate, can be synthesized artificially? Steen Rasmussen's team has taken one of the simpler and more artificial routes (Rasmussen et al. 2004; 2003), but there is still a long way to go toward artificial cells that will be able to truly self-replicate and to hopefully perform some specific user-defined functions.

Cellular computing is still a very young field and lots of progress is to be expected in the next few decades, particularly in the areas of biotechnology, nanotechnology, modeling, and computing. The main challenges in this interdisciplinary field are mostly related to the remaining wide-open gap between biological systems and machines (Conrad 1989). On one hand, a much better understanding of cellular systems is required; on the other hand, we need to be able to apply this vast and growing body of knowledge to build better and more "lifelike" computing machines either by drawing inspiration from them or by using real cells and organisms for the purpose of computation.

The following research directions and challenges need to be addressed in the next few years:

1. Faithful qualitative and quantitative cellular models require a better understanding of the biological cell and its biochemical mechanism. For example, better noninvasive brain imaging techniques are required for a better understanding of neurons, populations of neurons, and brain regions.
2. Large-scale molecular simulations of cellular processes and large populations of cells. For example, by using specialized supercomputers, large-scale brain simulations only become possible now.
3. Novel computing paradigms are required to model and to understand biological systems. Boolean logic is appropriate for digital systems, where 0s and 1s can easily be represented by the presence and absence of electrical current, but alternative representations may be more appropriate for biological systems.
4. Novel programming paradigms for cellular systems made up from a vast number of cells. Traditional top-down design approaches do not typically scale up to such complexities. Nature-inspired bottom-up design approaches are required.

## Bibliography

### Primary Literature

- Abelson H, Allen D, Coore D, Hanson C, Rauch E, Sussman GJ, Weiss R (2000) Amorphous computing. *Commun ACM* 43(5):74–82
- Adelman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266:1021–1024
- Albert R, Barabasi AL (2002) Statistical mechanics of complex networks. *Rev Mod Phys* 74:47–97
- Alberts B, Bray D, Lewis J, Raff M, Roberts K, Watson JD (eds) (1994) *Molecular biology of the cell*, 3rd edn. Garland Publishing, New York
- Amos M (ed) (2004) *Cellular computing*. Oxford University Press, New York
- Arbib MA (ed) (1995) *The handbook of brain theory and neural networks*. MIT Press, Cambridge
- Astor JC, Adami C (2001) A developmental model for the evolution of artificial neural networks. *Artif Life* 6(3):189–218
- Baldi P, Brunak S (2001) *Bioinformatics: the machine learning approach*, 2nd edn. MIT Press, Cambridge
- Banâtre JP, Coutant A, Le Metayer D (1988) A parallel machine for multiset transformation and its programming style. *Future Gener Comput Syst* 4:133–144
- Benenson Y, Paz-Elizur T, Adar R, Keinan E, Leivneh Z, Shapiro E (2001) Programmable and autonomous computing machine made of biomolecules. *Nature* 414(6862):430–434
- Bennett CH (1982) The thermodynamics of computation a review. *Int J Theor Phys* 21(12):905–940
- Berry G, Boudol G (1992) The chemical abstract machine. *Theor Comput Sci* 96:217–248
- Butera WJ (2002) Programming a paintable computer. PhD thesis, MIT Media Lab, Cambridge
- Calude CS, Paun G (2000) *Computing with cells and atoms: an introduction to quantum and membrane computing*. Taylor & Francis, New York
- Capcarrere MS (2002) *Cellular automata and other cellular Systems: design & evolution*. PhD thesis, Swiss Federal Institute of Technology, Lausanne

- Cell Matrix Corporation. <http://www.cellmatrix.com>. Accessed 12 July 2008
- Chua LO, Roska T (2002) Cellular neural networks & visual computing. Cambridge University Press, Cambridge
- Conrad M (1989) The brain-machine disanalogy. *BioSystems* 22(3):197–213
- Conrad M, Liberman EA (1982) Molecular computing as a link between biological and physical theory. *J Theor Biol* 98(2):239–252
- Cook M (2004) Universality in elementary cellular automata. *Complex Syst* 15(1):1–40
- Coore D (1999) Botanical computing: a developmental approach to generating interconnect topologies on an amorphous computer. PhD thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge
- Cuniberti G, Fagas G, Richter K (eds) (2005) Introducing molecular electronics. Lecture notes in physics, vol 680. Springer, Berlin
- Das R, Crutchfield JP, Mitchell M, Hanson JE (1995) Evolving globally synchronized cellular automata. In: Eshelman LJ (ed) Proceedings of the sixth international conference on genetic algorithms. Morgan Kaufmann, San Francisco, pp 336–343
- Davis M (1958) Computability and unsolvability. McGraw-Hill, New York
- Davis BD (1961) The teleonomic significance of biosynthetic control mechanisms. *Cold Spring Harb Symp Quant Biol* 26:1–10
- Davis M (2004) The myth of hypercomputation. In: Teuscher C (ed) Alan Turing: life and legacy of a great thinker. Springer, Berlin, pp 195–212 (reprinted with color images in 2005)
- Dellaert F, Beer RD (1994) Toward an evolvable model of development for autonomous agent synthesis. In: Brooks RA, Maes P (eds) Artificial life IV. Proceedings of the fourth international workshop on the synthesis and simulation of living systems. A Bradford Book, MIT Press, Cambridge, pp 246–257
- Deutsch D (1985) Quantum theory, the Church-Turing principle of the universal quantum computer. *Proc R Soc Lond A* 400:97–117
- Dittrich P, Ziegler J, Banzhaf W (2001) Artificial chemistries a review. *Artif Life* 7(3):225–275
- Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge
- Durbeck LJK, Macias NJ (2001) The cell matrix: an architecture for nanocomputing. *Nanotechnology* 12:217–230
- Eggenberger P (1997) Creation of neural networks based on developmental and evolutionary principles. In: Gerstner W, Germond A, Hasler M, Nicod JD (eds) Proceedings of the international conference on artificial neural networks. ICANN'97, Lausanne, Switzerland. Lecture notes in computer science, vol 1327. Springer, Berlin, pp 337–342
- Endy D, Brent R (2001) Modelling cellular behavior. *Nature* 409(6818):391–395
- Feynman RP (1960) There's plenty of room at the bottom: an invitation to enter a new field of physics. *Caltech's Eng Sci* 23:22–36
- Fleischer KW, Barr AH (1994) A simulation testbed for the study of multicellular development: the multiple mechanisms of morphogenesis. In: Langton CG (ed) Artificial life III. SFI studies in the science of complexity, vol XVII. Addison-Wesley, Redwood City, pp 389–416
- Flemming W (1880) Beiträge zur Kenntnis der Zelle und ihrer Lebenserscheinungen. *Arch Mikrosk Anat* 18:151–289
- Forbes N (2004) Imitation of life: how biology is inspiring computing. MIT Press, Cambridge
- Gibbs WW (2001) Cybernetic cells. *Sci Am* 285(2):43–47
- Gokhale M, Graham PS (2005) Reconfigurable computing: accelerating computation with field programmable gate arrays. Springer, Berlin
- Gruau F (1994) Neural network synthesis using cellular encoding and the genetic algorithm. PhD thesis, Ecole Normale Supérieure de Lyon
- Harel D (2003) Computers Ltd.: what they really can't do. Oxford University Press, New York
- Hillis DW (1985) The connection machine. MIT Press, Cambridge
- Hooke R (1665) Micrographia: or, some physiological descriptions of minute bodies made by magnifying glasses. J. Martyn and J. Allestry, London
- Jacob F, Monod J (1961) Genetic regulatory mechanisms in the synthesis of proteins. *J Mol Biol* 3:318–356
- Kish LB (2002) End of Moore's law: thermal (noise) death of integration in micro and nano electronics. *Phys Lett A* 305:144–149
- Kitano H (ed) (2001) Foundations of systems biology. MIT Press, Cambridge
- Koch C (1999) Biophysics of computation. Oxford University Press, New York
- Landweber LF, Kari L (1999) The evolution of cellular computing: nature's solution of a computational problem. *BioSystems* 52:3–13
- Langton CG (1984) Self-reproduction in cellular automata. *Physica D* 10(1–2):135–144
- Langton CG (ed) (1995) Artificial life: an overview. MIT Press, Cambridge
- Lederman H, Macdonald J, Stefanovic D, Stojanovic MN (2006) Deoxyribozyme-based three-input logic gates and construction of a molecular full adder. *Biochemistry* 45(4):1194–1199
- Lindenmayer A (1968) Mathematical models for cellular interaction in development, parts I and II. *J Theor Biol* 18:280–315
- Lloyd S (2000) Ultimate physical limits to computation. *Nature* 406:1047–1054
- Macias NJ (1999) The PIG paradigm: the design and use of a massively parallel fine grained self-reconfigurable infinitely scalable architecture. In: Stoica A, Keymeulen D, Lohn J (eds) Proceedings of the First NASA/DOD Workshop on Evolvable Hardware. IEEE Computer Society, Los Alamitos, pp 175–180

- Mange D, Tomassini M (eds) (1998) Bio-inspired computing machines: towards novel computational architectures. Presses Polytechniques et Universitaires Romandes, Lausanne
- Mange D, Sipper M, Stauffer A, Tempesti G (2000) Toward robust integrated circuits: the embryonics approach. *Proc IEEE* 88(4):516–540
- Mange D, Stauffer A, Peparolo L, Tempesti G (2004) A macroscopic view of self-replication. *Proc IEEE* 92(12):1929–1945
- McCulloch WS, Pitts WH (1943) A logical calculus of the ideas immanent in neural nets. *Bull Math Biophys* 5:115–133
- Minsky ML (1967) Computation: finite and infinite machines. Prentice-Hall, Englewood Cliffs
- Monod J, Jacob F (1961) Teleonomic mechanisms in cellular metabolism, growth, and differentiation. *Cold Spring Harb Symp Quant Biol* 26:389–401
- Moore GE (1965) Cramming more components onto integrated circuits. *Electronics* 38(8):114–117
- Nagpal R (2001) Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics. MIT Department of Electrical Engineering and Computer Science, Cambridge, Ph D thesis
- Odell GM, Oster G, Albrech P, Burnside B (1981) The mechanical basis of morphogenesis. In: Epithelial folding and invagination. *Dev Biol* 85:446–462
- Ono N, Ikegami T (2000) Self-maintenance and self-reproduction in an abstract cell model. *J Theor Biol* 206(2):243–253
- Ormondi AR, Rajapakse JC (eds) (2006) FPGA implementations of neural networks. Springer, Dordrecht
- Papadimitriou CH (1994) Computational complexity. Addison-Wesley, Reading
- Paton RC (1993) Some computational models at the cellular level. *Biosystems* 29:63–75
- Paton R, Bolouri H, Holcombe M, Parish JH, Tateson R (eds) (2004) Computation in cells and tissues. Springer, Berlin
- Pattee HH (1961) On the origin of macromolecular sequences. *Biophys J* 1(8):683–710
- Paun G (2000) Computing with membranes. *J Comput Syst Sci* 61(1):108–143, first published in a TUCS Research Report, No 208, Nov 1998, <http://www.tucs.fi>
- Paun G (2002) Membrane computing. Springer, Berlin
- Paun G, Rozenberg G, Salomaa A (1998) DNA computing: new computing paradigms. Springer, Heidelberg
- Pawelez N (2001) Walther Flemming: pioneer of mitosis research. *Nat Rev Mol Cell Biol* 2:72–75
- Petraglio E (2004) Fault tolerant self-replicating systems. Swiss Federal Institute of Technology (EPFL), Lausanne, Ph D thesis
- Petreska B, Teuscher C (2004) A reconfigurable hardware membrane system. In: Martin-Vide C, Mauri G, Paun G, Rozenberg G, Salomaa A (eds) Membrane computing. Lecture notes in computer science, vol 2933. Springer, Berlin, pp 269–285
- Pohorille A, Deamer D (2002) Artificial cells: prospects for biotechnology. *Trends Biotechnol* 20(3):123–128
- Rasmussen S, Chen L, Nilsson M, Abe S (2003) Bridging nonliving and living matter. *Artif Life* 9(3):269–316
- Rasmussen S, Chen L, Deamer D, Krakauer DC, Packard NH, Stadler PF, Bedau MA (2004) Transitions from nonliving to living matter. *Science* 303:963–965
- Rendell P (2002) Turing universality of the game of life. In: Adamatzky A (ed) Collision-based computing. Springer, London, pp 513–539
- Rust AG (1998) Developmental self-organisation in artificial neural networks. PhD thesis, University of Hertfordshire
- Schmitt FO (1962) Macromolecular specificity and biological memory. MIT Press, Cambridge
- Seelig G, Soloveichik D, Zhang DY, Winfree E (2006) Enzyme-free nucleic acid logic circuits. *Science* 314:1585–1588
- Shapiro E, Gil B (2007) Logic goes in vitro. *Nat Nanotechnol* 2:84–85
- Sienko T, Adamatzky A, Rambidi NG, Conrad M (eds) (2003) Molecular computing. MIT Press, Cambridge
- Sipper M (1996) Co-evolving non-uniform cellular automata to perform computations. *Physica D* 92:193–208
- Sipper M (1997) Evolution of parallel cellular machines: the cellular programming approach. Springer, Heidelberg
- Sipper M (1999) The emergence of cellular computing. *IEEE Comput* 32(7):18–26
- Sipper M (2002) Machine nature: the coming age of bio-inspired computing. McGraw-Hill, New York
- Sipper M, Ruppin E (1997) Co-evolving architectures for cellular machines. *Physica D* 99:428–441
- Sipper M, Sanchez E, Mange D, Tomassini M, Pérez-Uribe A, Stauffer A (1997a) A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Trans Evol Comput* 1(1):83–97
- Sipper M, Goeke M, Mange D, Stauffer A, Sanchez E, Tomassini M (1997) The firefly machine: online evolware. In: Proceedings of the 1997 I.E. international conference on evolutionary computation (ICEC'97), Piscataway. IEEE Press, Piscataway, pp 181–186
- Sipser M (2006) Introduction to the theory of computation, 2nd edn. Thomson, Boston
- Stahl WR, Goheen HE (1963) Molecular algorithms. *J Theor Biol* 5(2):266–287
- Stauffer A, Mange D, Tempesti G, Teuscher C (2001) Bio watch: a giant electronic bio-inspired watch. In: Keymeulen D, Stoica A, Lohn J, Zebulum RS (eds) Proceedings of the third NASA/DoD workshop on evolvable hardware, EH-2001. IEEE Computer Society, Los Alamitos, pp 185–192
- Stojanovic MN, Stefanovic D (2003) A deoxyribozyme-based molecular automaton. *Nat Biotechnol* 21:1069–1074
- Sugita M (1961) Functional analysis of chemical systems in vivo using a logical circuit equivalent. *J Theor Biol* 1(2):415–430
- Sugita M (1963) Functional analysis of chemical systems in vivo using a logical circuit equivalent. In: the idea of a molecular automaton. *J Theor Biol* 4(2):179–192

- Sugita M, Fukuda N (1963) Functional analysis of chemical systems in vivo using a logical circuit equivalent. III: analysis using a digital circuit combined with an analogue computer. *J Theor Biol* 5(3):412–425
- Szostak JW, Bartel DP, Luisi PL (2001) Synthesizing life. *Nature* 409:387–390
- Tempesti G, Mange D, Stauffer A, Teuscher C (2002) The biowall: an electronic tissue for prototyping bio-inspired systems. In: Stoica A, Lohn J, Katz R, Keymeulen D, Zebulum RS (eds) *Proceedings of the 2002 NASA/DoD conference on evolvable hardware*. IEEE Computer Society, Los Alamitos, pp 221–230
- Teuscher C (2002) Turing's connectionism. An investigation of neural network architectures. Springer, London
- Teuscher C, Sipper M (2002) Hypercomputation: hype or computation? *Commun ACM* 45(8):23–24
- The P systems web page. <http://ppage.psystems.eu>. Accessed 12 July 2008
- Toffoli T, Margolus N (1987) Cellular automata machines. MIT Press, Cambridge
- Tomita M (2001) Whole-cell simulation: a grand challenge of the 21st century. *Trends Biotechnol* 19(6):205–210
- Top500 supercomputing sites. <http://www.top500.org>. Accessed 12 July 2008
- Turing AM (1937) On computable numbers, with an application to the Entscheidungsproblem. *Proc Lond Math Soc* 42:230–265; Corrections. *Proc Lond Math Soc* 43:544–546
- Turing AM (1950) Computing machinery and intelligence. *Mind* 59(236):433–460
- Turing AM (1952) The chemical basis of morphogenesis. *Philos Trans R Soc Lond B* 237:37–72
- Turing AM (1969) Intelligent machinery. In: Meltzer B, Michie D (eds) *Machine intelligence*, vol 5. Edinburgh University Press, Edinburgh, pp 3–23
- Turing AM (1992) Intelligent machinery. In: Ince DC (ed) *Collected works of A.M. Turing: mechanical intelligence*. North-Holland, Amsterdam, pp 107–127
- Tyrrell A, Sanchez E, Floreano D, Tempesti G, Mange D, Moreno JM, Rosenberg J, Alessandro Villa EP (2003) Poetic tissue: an integrated architecture for bio-inspired hardware. In: Tyrrell AM, Haddow PC, Torresen J (eds) *Evolvable systems: from biology to hardware*. *Proceedings of the 5th international conference (ICES2003)*. Lecture notes in computer science, vol 2606. Springer, Berlin, pp 129–140
- Varela F, Maturana H, Uribe R (1974) Autopoiesis: the organization of living systems, its characterization and a model. *BioSystems* 5:187–196
- Villasenor J, Mangione-Smith WH (1997) Configurable computing. *Sci Am* 276(6):54–59
- von Neumann J (1966) *Theory of self-reproducing automata*. University of Illinois Press, Urbana
- Warner JR, Rich A, Hall CE (1962) Electron microscope studies of ribosomal clusters synthesizing hemoglobin. *Science* 138(3548):1299–1403
- Watson JD, Crick FHC (1953) A structure for deoxyribose nucleic acid. *Nature* 171:737–738
- Watts DJ, Strogatz SH (1998) Collective dynamics of small-world networks. *Nature* 393:440–442
- Weiss R (2001) *Cellular computation and communications using engineered genetic regulatory networks*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge
- Wolfram S (1984) Cellular automata as models of complexity. *Nature* 311:419–424

## Books and Reviews

- Adamatzky A (ed) (2002) *Collision-based computing*. Springer, London
- Adami C (1998) *Introduction to artificial life*. Springer, New York
- Amos M (2005) *Theoretical and experimental DNA computation*. Springer, Berlin
- Bentley BJ (2001) *Digital biology: how nature is transforming our technology*. Headline Book Publishing, London
- Ehrenfeucht A, Harju T, Petre I, Prescott DM, Rozenberg G (2004) *Computation in living cells: gene assembly in ciliates*. Springer, Berlin
- Hey AJG (ed) (1998) *Feynman and computation: exploring the limits of computers*. Westview, Boulder
- Landweber LF, Winfree E (eds) (2002) *Evolution as computation*. Springer, Berlin
- Paun G, Rozenberg G (2002) A guide to membrane computing. *J Theory Comput Sci* 287(1):73–100
- Petty MC, Bryce MR, Bloor D (eds) (1995) *An introduction to molecular electronics*. Oxford University Press, New York
- Pozrikidis C (ed) (2003) *Modeling and simulation of capsules and biological cells*. Chapman and Hall/CRC, Boca Raton
- Trimberger SM (1994) *Field-programmable gate array technology*. Kluwer, Boston
- Wolfram S (2002) *A new kind of science*. Wolfram Media, Champaign
- Zauner KP (2005) Molecular information technology. *Crit Rev Solid State Mater Sci* 30(1):33–69



---

## Neuromorphic Computing Based on Organic Memristive Systems

Victor Erokhin

Institute of Materials for Electronics and Magnetism, Italian National Council of Research (CNR-IMEM), Parma, Italy

### Article Outline

Glossary  
Definition of the Subject  
Introduction  
Architecture and Properties of Organic Memristive Device  
Biomimicking Circuits  
Organic Memristive Device Based Hardware  
Artificial Neuron Networks  
Conclusions and Future Directions  
Bibliography

### Glossary

**Artificial Neuron Networks (ANN)** Computing systems inspired by the biological neural networks.

**Logic with memory** Logic gates with integrated memory: output depends not only on the current configurations of inputs but also on the history of the gate function.

**Memristor** Hypothetical element, introduced by L. Chua in 1971; originally, the resistance of the memristor must be a function of the passed charge.

**Memristive device** Modern understanding of ideal Chua's memristor, including devices, varying their resistance, capacity, or inductance.

**Mnemotrix** Hypothetical element introduced by V. Braitenberg in 1984 for explaining learning capabilities.

**Perceptron** Perceptron is an algorithm or device for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not).

**Polyaniline (PANI)** Conducting polymer whose conductivity depends strongly on the doping level and redox state.

**Polyethylene oxide (PEO)** One of the most used organic matrix for solid electrolytes (usually, doped with lithium salts).

**Synapse** A synapse is a neural junction used for communication between neurons.

### Definition of the Subject

Neuromorphic computing is bioinspired and biomimicking system, where information storage and treatment is done by same elements, what allows learning at a hardware level.

Organic memristive devices (can be considered also as device realization of the Braitenberg's mnemotrix) have several properties, allowing considering them as electronic analogs of biological synapses.

We consider here some circuits, mimicking the function of the parts of nervous system and networks, allowing learning, classification, and decision making.

### Introduction

Significant difference between modern computers and nervous system in general, and brain in particular, is the architecture: in computers, memory and processor are different devices. Thus, information is playing a passive role: it can be recorded, accessed, and eliminated. However, it makes no influence on the processor properties. In the brain, instead, the same elements are used for the memorizing and processing of the

information. This feature allows learning at the “hardware” level. Accepted information is not simply memorized but it varies mutual connections between elements of the nervous system (neurons). Synapses (connections of the axons of one neuron with dendrites of the other one) play very important role in such learning. One of the main paradigms (even if not the only one) is a rule of Hebb, stating (Hebb 1961):

When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased.

Thus, the weight function of the junction (synapse) for the signal transfer between two neuronal cells increases with the frequency of its involvement into the formation of signal pathways.

Similar approach is used for the formation of artificial neuron networks: nonlinear threshold elements are connected with each other by links that can vary their conductivity. At early stages of the computer development efforts were dedicated to the hardware realization of artificial neuron networks (Rosenblatt 1958). However, great success of the abovementioned architecture (separation of the memory and processor) resulted in the fact that the most of works on the artificial neuron networks now are done at the software level.

Nevertheless, hardware realization of artificial neuron networks is still an actual problem. One reason is that it will allow really parallel information processing. Even if now there are multinuclear computers, where each nucleus can perform task execution independently, the concept remains the same: each nucleus performs a single task in a given moment. Of course, when the number of these nuclei will be very large, we can expect qualitative breakthrough. However, currently it is not a case. Second reason is connected to the fact that new computer architecture will be more similar to the nervous system and can be very useful for the understanding and modelling of the brain function. In addition, it can provide new approaches for the realization of effective brain-computer interface. Some steps in this direction were already done (Gupta et al. 2016).

New phase in the activity of hardware realization of artificial neuron networks started in 2008, when HP group published a paper, claiming the realization of memristor (Strukov et al. 2008).

Memristor is a hypothetical element, introduced in 1971 by Chua as a missed passive electronic compound (Chua 1971). Important property of the element is the dependence of the conductance on the quantity of the charge that has passed through it. As it can be recognized, this property marks the element as a synapse analog working according to the Hebbian rule: passed charge corresponds to the frequency (or duration) of the element involvement into the signal pathway formation.

In the mentioned HP work (Strukov et al. 2008), the element was a two terminal device, using nanoscale titanium oxide between two metal electrodes. The variation of the conductivity was attributed to the migration of oxygen vacancies in the applied electric field.

Resistance switching phenomena were observed and studied even before this work (Asamitsu et al. 1997; Waser and Aono 2007); however, HP group for the first time connected such properties with the concept of memristor.

The work resulted in a huge resonance: the number of published papers and specially dedicated conferences are continuously increasing. The most of research groups use metal oxides as basic material. However, other materials are also used for the realization of such elements (Kim et al. 2009; Pershin and Di Ventra 2008).

As it was shown now, ideal memristors do not exist (Pershin and Di Ventra 2011). However, the concept of such devices was enlarged for including other elements with memory (Corinto et al. 2015).

Polyaniline-(PANI-) based organic memristive device (Erokhin et al. 2005; Erokhin 2007, 2015) was developed starting not from the Chua memristor definition, but from a mental experiment of Valentino Braitenberg, where he introduced a mnemotrix concept (Braitenberg 1984):

we buy a role of special wire, called Mnemotrix, which has the following interesting property: its resistance is at first very high and stays high unless

the two components that it connects are at the same time traversed by an electric current. When this happens, the resistance of Mnemotrix decreases and remains low for a while

Thus, the task was to make a device, which must have several specific properties, allowing adaptations of the system, namely: anisotropy of the conductivity and dependence of the device resistance on the history of its previous involvement into the signal transfer process.

This chapter will describe the construction of this element, its main properties, several particular examples of the computational systems, based on these elements, and some future developments.

## Architecture and Properties of Organic Memristive Device

Scheme of the organic memristive device is shown in Fig. 1.

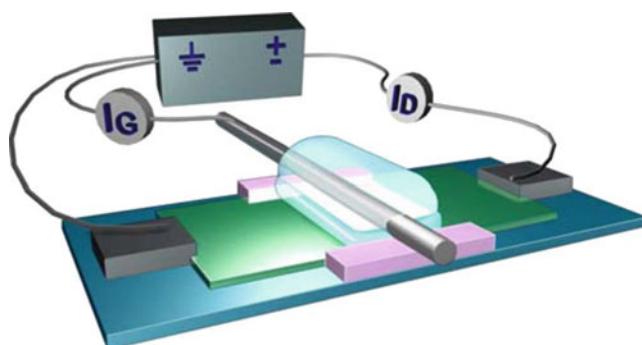
Working principle of the device function is based on the significant difference of the PANI conductivity when the material is in a reduced or oxidized state (Kang et al. 1998). This property determines the device architecture (Erokhin et al. 2005). Active channel of the device is made from thin PANI layer to which two metal electrodes, called according to field effect transistors terminology as Source and Drain, are connected. In the middle part, the PANI layer is covered with a stripe of solid electrolyte (usually, polyethylene

oxide (PEO) doped with lithium salts). Contact of PANI and PEO is an active zone where all redox reactions occur, varying, therefore, the device conductivity. In order to control the PANI redox state in the active zone, a silver wire (Gate electrode) is attached to PEO and connected to one of terminal electrodes, maintained at the ground potential level. Therefore, PANI potential in the active zone will be determined by the voltage, applied to the second electrode and by the conductivity profile along the PANI channel length.

As it is clear from the figure, two currents can flow in this structure: electronic current in the PANI channel and ionic current in the PEO electrolyte. Even if the device is a two terminal one with respect to the connection to the external circuit, it is better to analyze separately electrical properties regarding these two contributions into the total current. As it is shown in the Fig. 1, we have performed measurement of the total device current ( $I_D$ ) and ionic current ( $I_G$ ). Electronic current can be obtained by a simple subtraction of the second contribution from the first one.

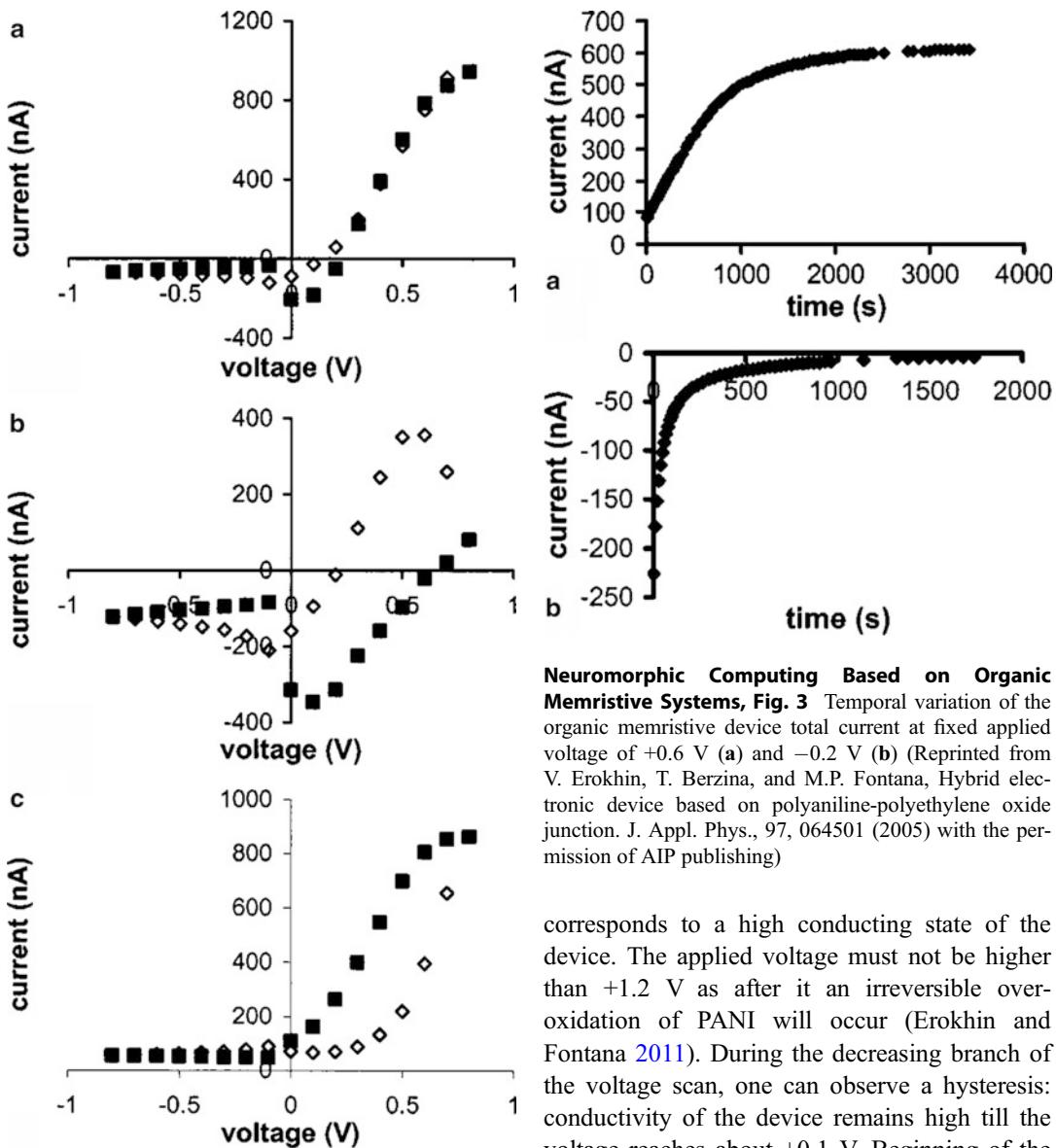
Cyclic voltage-current characteristics of the device are shown in Fig. 2.

The voltage scan starts at 0 V in the direction of positive bias. During the initial voltage increase, we can see very small values of both currents (PANI in the active zone is in reduced insulating state). After a certain value (about +0.5 V), we can see the increase of the electronic conductivity and appearance of the positive maximum in the ionic



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 1** Organic memristive device (Reprinted from T. Berzina, A. Smerieri, M. Bernabo', A. Pucci, G. Ruggeri, V. Erokhin, and M.P. Fontana,

Optimization of an organic memristor as an adaptive memory element. J. Appl. Phys., 105, 124515 (2009) with the permission of AIP publishing)



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 2** Cyclic voltage current characteristics of the organic memristive device for total (a), ionic (b), and electronic (c) currents. Empty rhombs represent the increasing trend of the applied voltage; filled squares represent the decreasing trend of the applied voltage (Reprinted from V. Erokhin, T. Berzina, and M.P. Fontana, Hybrid electronic device based on polyaniline-polyethylene oxide junction. J. Appl. Phys., 97, 064501 (2005) with the permission of AIP publishing)

current, what corresponds to the oxidation of PANI and its transfer into a conducting state. Further increase of the applied voltage

**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 3** Temporal variation of the organic memristive device total current at fixed applied voltage of +0.6 V (a) and -0.2 V (b) (Reprinted from V. Erokhin, T. Berzina, and M.P. Fontana, Hybrid electronic device based on polyaniline-polyethylene oxide junction. J. Appl. Phys., 97, 064501 (2005) with the permission of AIP publishing)

corresponds to a high conducting state of the device. The applied voltage must not be higher than +1.2 V as after it an irreversible overoxidation of PANI will occur (Erokhin and Fontana 2011). During the decreasing branch of the voltage scan, one can observe a hysteresis: conductivity of the device remains high till the voltage reaches about +0.1 V. Beginning of the decrease of the electronic current corresponds to the negative maximum of the ionic current: PANI in the active zone is reduced and the device is transferred into the low conducting state. Whole negative branch is in this state.

For the realization of adaptive networks it seems even more important to study the variation of the conductivity at fixed applied voltage, shown in Fig. 3.

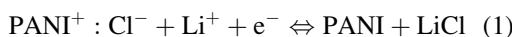
The dependence shown in Fig. 3a corresponds well to the Hebbian rule and can be considered as a basis for unsupervised learning. In fact,

connections in the network, composed by numerous organic memristive devices, will be potentiated according to the frequency and/or duration of their involvement into the formation of signal preferential pathways. The dependence in Fig. 3b, instead, can establish a basis for the supervised learning. In fact, it will be enough to invert the polarity between certain input-output pair to eliminate the spontaneously formed connections between them, established during the unsupervised learning.

In the first organic memristive devices, the ON/OFF ratio of the conductivity was 100. After optimization of the device composition and architecture, this value was increased and now the best devices have  $10^5$  ON/OFF ratio (Erokhin et al. 2008).

Mechanism of the device functioning was studied by microRaman spectroscopy (Berzina et al. 2007) and X-ray fluorescence using synchrotron radiation for the excitation (Berzina et al. 2009). In both cases, we have observed a motion of  $\text{Li}^+$  ions between PANI and PEO layers in the active zone.

Reaction, responsible for the conductivity variation, is shown in formula (1):



Conducting form of PANI has a lack of electron in the chain.  $\text{Cl}^-$  ion is located near the chain for the electrical neutrality. When the reduction potential is applied, PANI receives the electron, while  $\text{Cl}^-$  is coupled with  $\text{Li}^+$  ion that enters PANI

film from the PEO side. Such mechanism explains why we must use lithium (due to the sizes) and thin PANI layer (due to diffusion control of the conductivity).

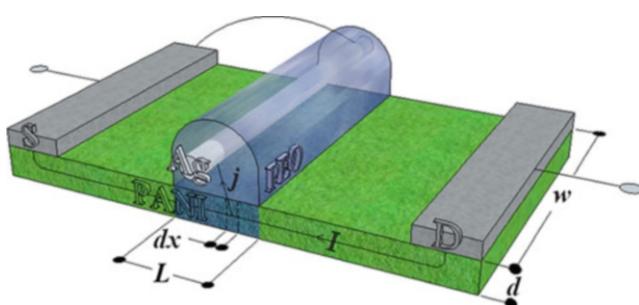
Even if we present here data on DC behavior of organic memristive device, similar results were obtained also in the pulse (Smerieri et al. 2008a).

The model, describing the characteristics of the device, is based on the splitting of the active zone to narrow stripes, supposing the processes occurring homogeneously in each of them, as it is shown in Fig. 4 (Demin et al. 2014)

Electrical equivalent circuit is shown in Fig. 5.

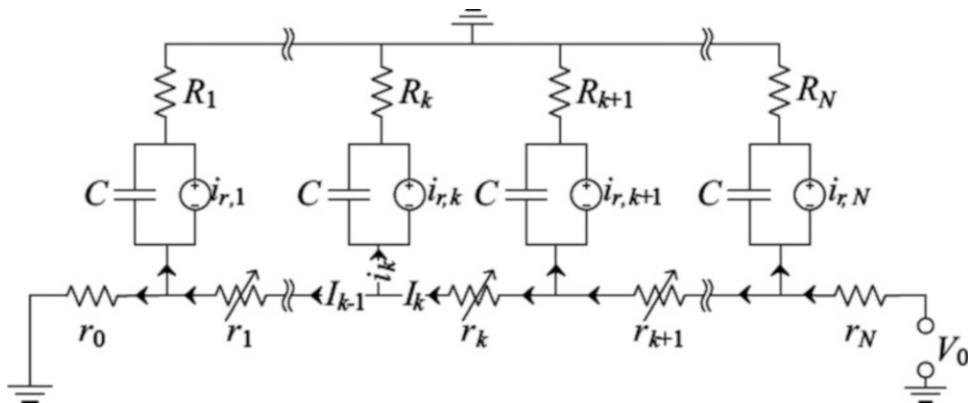
Each stripe of PANI is represented by variable resistor  $r_k$ . Gate circuit is composed of the resistor  $R_k$  serially connected to parallel connection of the capacitor  $C$  and current generator  $i_{r,k}$ .  $R_k$  and  $C$  are passive components of the electrolyte, while  $i_{r,k}$  is an active generator of the redox current. These generators are switched on when the corresponding stripe has a potential higher than the oxidation one or lower than the reduction one. Within the model, the program calculates a voltage distribution profile for each time moment. If the stripe passed reduction or oxidation potential, the generator was activated for providing ion motion, necessary for the redox reactions.

The developed model did not require additional suggestions and the results were in a good agreement with experimental data. In addition, the model explained also the observed difference in the kinetics of the transfer of the device from insulating to conducting states and vice versa.



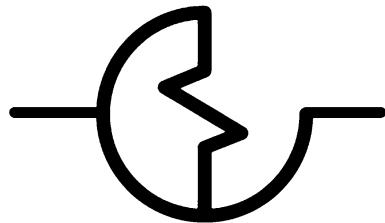
**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 4** Scheme of the organic memristive device (Reprinted from V.A. Demin, V. Erokhin, P.K. Kashkarov, and M.V. Kovalchuk,

Electrochemical model of the polyaniline based organic memristive device. J. Appl. Phys., 116, 064507 (2014) with the permission of AIP publishing)



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 5** Equivalent circuit of the organic memristive device (Reprinted from V.A. Demin, V. Erokhin, P.K. Kashkarov, and M.V. Kovalchuk,

Electrochemical model of the polyaniline based organic memristive device. J. Appl. Phys., 116, 064507 (2014) with the permission of AIP publishing)



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 6** Symbol of organic memristive device

In the next part of the chapter we will consider circuits and networks, based on organic memristive devices. The symbol that we will use for these circuits and networks is shown in Fig. 6 (Erokhin et al. 2007a).

## Biomimicking Circuits

### Oscillator

Presence of rhythmic variations of internal parameters at fixed environmental conditions is an essential feature of living beings (Schrodinger 1944). Considering bioinspired or biomimicking computing, this feature is important for the realization of clock generators.

It turned out that small modification of the memristor can transfer it into an oscillator (Erokhin et al. 2007b; Smerieri et al. 2008b). It

was enough only to insert a capacitor into the circuit of the gate electrode. Temporal dependences of the gate and drain currents of the organic memristive devices, where an external capacitor of  $1.0 \mu\text{F}$  was connected between gate electrode and the ground at fixed applied voltage of  $1.0 \text{ V}$  revealed the appearance of auto-oscillations.

It is interesting to note that the oscillations of gate and drain currents are shifted for about  $100 \text{ s}$  – characteristic time of the conductivity variation.

It is possible to transfer organic memristive device into the oscillator even without connection of the external element (capacitor). It was reached by the use of appropriate material for the gate electrode realization. When silver wire was substituted with a stripe of highly oriented graphite, similar characteristics were acquired. In this case, graphite role was to serve as an accumulator of lithium ions (supercapacitor). In fact, it is well known that  $\text{Li}^+$  can be intercalated between graphite lattice layers.

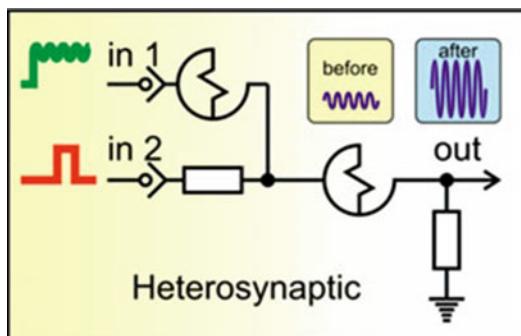
### Mimicking Parts of Nervous System

As it was mentioned before, organic memristive device (mnemotrix (Braitenberg 1984)) was developed for the mimicking synapse properties. Therefore, starting from the beginning we tried to realize circuits, mimicking associative

properties. The very first circuit was based on one memristive device only (Smerieri et al. 2008c). Later, similar work was attributed to the analog of Pavlov's dog learning (Ziegler et al. 2012).

Direct evidence of the synapse mimicking properties in electronic circuits was done by reproducing architecture and properties of the pond snail nervous system part, responsible for learning of this animal during feeding (Erokhin et al. 2012a). In this case we had a model, developed after analysis of the data, obtained from the experiments with implanted microelectrode arrays. Learning of the snail implies the association of the initially neutral input (lip touching) with the presence of food (Straub and Benjamin 2001; Klemenes et al. 2006): after touching lips with sugar, the animal associate touching with the presence of food and begins to open mouth and start digestion process after lips touching even without sugar.

The important feature of the developed model is the fact that we need only two synapses for the execution of this very simple learning. The artificial circuit, mimicking the architecture and properties of this feature is shown in Fig. 7.



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 7** Electric scheme, mimicking part of the pond snail nervous system, responsible for learning during feeding: position of the memristive devices corresponds directly to the position of synapses; input 1 corresponds to the mechanical stimulus; input 2 corresponds to the food presence; the value of the output (after passing a threshold) can open mouth and start digestion process (Reprinted from V. Erokhin, T. Berzina, P. Camorani, A. Smerieri, D. Vavoulis, J. Feng, and M.P. Fontana, Material memristive device circuits with synaptic plasticity: learning and memory. *BioNanoScience*, 1, 24–30 (2011) with the permission of Springer publishing)

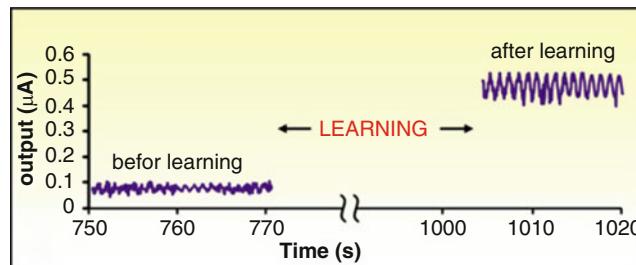
The electronic circuit reproduces directly the model of the nervous system part. Organic memristive devices are in the position of synapses. The circuit has two inputs: one corresponds to the initially neutral stimulus (mechanical touching of lips) and the other one corresponds to the food presence. The system has one output: it can be connected to the motor, responsible for the mouth opening. This motor must have a threshold value: when the output is higher of this value, the motor starts operating.

It is to note that this scheme can be considered also as Pavlov's dog. However, dog is more complicated system comparing with a snail, and other mechanisms can be involved into the association of the bell ring with the food presence. Here, instead, the animal is rather simple and the nervous system model is available. Thus, we can mimic not only the function, but also the nervous system architecture, realizing this function.

Experimental results of the learning of the circuit, shown in Fig. 7 are presented in Fig. 8.

Initially, only an input, corresponding to the mechanical stimulus, is applied (oscillating voltage). The corresponding output is rather small. During learning phase, both stimuli are applied (analog of the mechanical touching and presence of food). When learning phase is finished, the analog of the neutral stimulus (touching without food) along is applied once more. As it is clear from Fig. 8, oscillation amplitude and DC offset were increased for about five times. If the motor with threshold, mentioned above, will be connected to the output, the snail robot will begin to open mouth in the presence of a mechanical stimulus only (after appropriate learning).

The working principle of the suggested circuit is the following one. When the signal, corresponding to the mechanical stimulus is applied, the voltage is distributed between two memristive devices (synapses) and its value is such that it is not enough for transferring each of them into conducting state. In particular, +0.6 V was applied: +0.3 V on each element that must not change their conductivity state. Instead, when the signal, corresponding to the presence of food, is applied, it will result in switching of the second memristive device into a conducting state.



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 8** Experimental results, demonstrating learning capabilities. Initially, application of the signal, corresponding to the mechanical stimulus, results in the rather low output signal value. During the learning phase, both stimuli (mechanical one and the presence of food) are applied simultaneously. After it, again the mechanical stimulus only is applied and the corresponding

output turned out to be five times increased. “Electronic snail” learned to associate touching with the presence of food (Reprinted from V. Erokhin, T. Berzina, P. Camorani, A. Smerieri, D. Vavoulis, J. Feng, and M.P. Fontana, Material memristive device circuits with synaptic plasticity: learning and memory. *BioNanoScience*, 1, 24–30 (2011) with the permission of Springer publishing)

Thus, this element will be much more conducting and the signal (voltage), corresponding to the mechanical stimulus will be located at the first memristive device. Its value is sufficient for the resistance switching. Thus, after this procedure, the output for the same input will be significantly increased.

These results have demonstrated that the organic memristive device can be really used as a synapse analog in electronic circuits. It is to note that other memristor was used for making an electronic circuit, simulating Pavlov’s dog learning (Ziegler et al. 2012). However, as a dog is much more complicated system with respect to the pond snail, the reported results (Ziegler et al. 2012) mimic only properties (generally speaking, the same as in the case of pond snail: association of initially neutral stimulus with the presence of food), but not the architecture of the responsible part of the nervous system.

## Organic Memristive Device Based Hardware Artificial Neuron Networks

### Logic with Memory

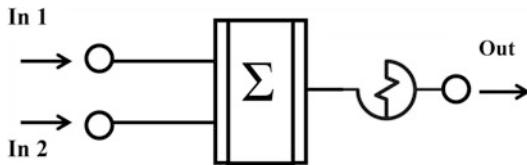
Logic gates are important compounds of modern computers. It seems not useful to reproduce these compounds with memristors, as they can be easily produced with standard CMOS technology. Instead, memristors can be key elements for new

type of logic gates – logic with memory (Erokhin et al. 2012b; Baldi et al. 2014).

Nervous system, in general, and brain, in particular, do not use Boolean logic in its basic form. Let us consider an operation AND. This operation, in the simplest case, can be considered as an identification of an object when two important characteristics are present. However, living beings do not make this association automatically: it requires learning. More frequently the coincidence of the characteristics occurs, more probable will be the association. It requires also a feedback confirming or not the correctness. Thus, the output will be not a binary code but an analog signal between 0 and 1, whose value indicates the probability of the identification of the object by two characteristics.

Scheme of the organic memristive device based AND gate with memory is shown in Fig. 9.

The gate has two inputs and one output. Input signals (voltages) are connected to a summator, providing a sum of both input voltages. The scheme includes also a divider (by two). In principle, this element is not required if the gate will work along. However, if it will be connected to a circuit with other logic gates with memory, it is better to use this element in order to have standard values of input voltages. Values of the applied voltages are such that only when both of them are applied to inputs, the resultant potential will be enough for transferring the organic memristive device into a more conducting state.



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 9** Scheme of the AND gate with memory

The study has demonstrated that individual applications of the voltage to input 1 or 2 results in only small constant increase of the output current (according to the Ohm Law). Instead, when both signals are applied, we see gradual increase of the output current in time. In the case of DC, the output value will be determined as:

$$S = I_{\text{out}}(t)/I_{\text{out}}(\infty)$$

where  $t$  is a time, when both inputs were applied.

In the pulse mode, the  $S$  value will depend on the number of coincided pulses on both inputs.

Similarly, OR and NOT gates with memory were realized and studied (Erokhin et al. 2012b).

Such elements will allow the realization of systems capable for decision making in a human-like mode: the answer will depend not only on a configuration of external stimuli, but also on the past experience of the system during its functioning.

## Perceptron

Perceptron is an artificial neural network capable of supervised learning in a wide variety of tasks. These tasks can be relatively easily solved by humans (in some cases also by animals) but are quite difficult for resolving by conventional computers.

Perceptron was developed by Rosenblatt in 1958 (Rosenblatt 1958) as a model of brain perception. Historically, single-layer perceptron was represented by three layers of neurons: sensory, associative, and responsive ones. Wherein, only the series of links from the associative to responsive neurons could be trained (varied in the process of the network learning), while the first layer of weights from sensory neurons to associative ones was considered accidentally or deterministically

given and remained unchanged in the learning process.

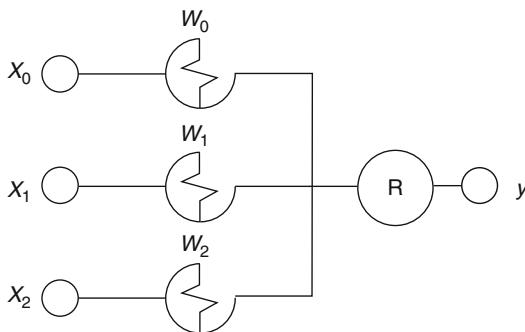
Later on, single-layer perceptron (e.g., in the terminology of Wasserman (1989)) became known as a network in which each sensory neuron actually was directly connected to a single neuron of associative layer. Thus, the need to assign a set of sensory neurons disappeared, and the only layer of variable links between associative and responsive neurons was held. However, works on the hardware implementation of perceptron were blocked for some years mainly due to the critical book of Minsky and Papert (1969), there were some misconceptions concerning the limitations of the classical perceptron.

In this respect, the realization of memristors, and organic memristive devices, in particular, has resulted in the revival of the experimental activities in this field. It is expected that characteristics of memristor-based artificial neuron networks (ANN) architectures would be strongly improved, compared to ANN emulated by software on actual computers (Wang et al. 2013).

The first report on the realization of the elementary perceptron for the images classification was done on the  $\text{TiO}_2$  crossbar memristors array (Prezioso et al. 2015). It was demonstrated the possibility of using this approach for the pattern recognition. As inorganic memristors are beyond the scope of this chapter (it will require a separate chapter or even more than one), we address readers to the above mentioned reference for details.

Scheme of the realized elementary perceptron based on organic memristive devices is shown in Fig. 10 (Demin et al. 2015).

The input layer of the perceptron (Fig. 10) consists of two functional inputs ( $x_1, x_2$ ), providing the information that will be used for the classification, and one permanently biased input ( $x_0 = "1"$ ), that is necessary for the realization of the offset, required for the implementation of some functions for separation of different classes of objects. Of course, such system can separate only two classes of objects: if the output value will overcome a threshold, the object will be classified as belonging to the class 1, otherwise, it will belong to the class 2. The number of classes can be easily increased by adding more outputs



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 10** Scheme of elementary perceptron based on organic memristive devices (Reprinted from V.A. Demin, V. Erokhin, A.V. Emelyanov, S. Battistoni, G. Baldi, S. Iannotta, P.K. Kashkarov, and M.V. Kovalchuk, Hardware elementary perceptron based on polyaniline memristive devices. *Organic Electronics*, 25, 16–20 (2015) Copyright (2011) with the permission from Elsevier)

connected to the previous network by variable link (memristive device).

In our case, all input signals were voltages, while the output was a current. Therefore, during the test phase, it is impossible to realize situations with negative weights: positive voltages will result always in the positive currents. However, we can do such actions at the learning phase: lowering of the weight function will be performed by the application of the negative potential to the appropriate input, while the potentiation will be done by the application of the positive potential higher than a threshold value (oxidation potential, higher than the voltage, corresponding to “1” used for the classification). In principle, the mentioned problem can be resolved by the connection of two memristive devices in opposite orientation for the each input channel. However, in the published work the first approach was realized.

We have applied the simplest algorithm of learning, suggested by Rosenblatt. It is a method of error correction (Rosenblatt 1961), wherein the weights of all active connections are changed in the same way in each iteration step depending only on the error sign but not on its magnitude.

For testing this perceptron, we decided to realize two simple logic functions: NAND and NOR. The reason of this choice was that the

classifications with these functions allow linear separation of the obtained results: the only classifications allowed with elementary perceptron.

Voltage values for the logic “0” and “1” were chosen, considering that their application should not vary the conductivity state of constituting memristive devices. As each chain contained one device only and taking into consideration that its switching into more conductive state occurs at a potential higher than about +0.5 V and into less conducting state at a potential lower than about +0.005 V, we took +0.2 V as a logic “0” and +0.4 V as logic “1.” For the potentiation of the weights we used +0.8 V, while for its inhibition we used -0.2 V. Duration interval for the potentiation and inhibition was different due to the different kinetics of the memristive device transfer from the conducting state to the insulating one and vice versa, what is due to the temporal redistribution of the applied voltage on the active zone length during the device function. In particular, analyzing properties of available devices we have chosen 300 s for the reinforcement and 30 s for the inhibition. Later, these values were varied, but principle results were the same.

The other important parameter is the value of the output current that can be classified as logic “0” or logic “1.” In order to determine this value we must take into account that during the classification phase we always have some positive current at the output (both input voltages are positive). Therefore, it was determined a threshold value of this current ( $I_t$ ), separating the state of the memristive device conductivity. In addition, we have determined a value of demarcation current ( $I_d$ ) – uncertainty of the attribution of the memristor device conductivity state to a conducting and insulating one. Values of these currents were determined experimentally analyzing electrical properties of memristive devices, constituting the perceptron. In particular, the following values were taken for the perceptron realization:  $I_t = 3.0 \mu\text{A}$ ,  $I_d = 0.5 \mu\text{A}$ .

Initial state of all memristive devices in the circuit was random. The training of the perceptron was done in the following way. The correctness table of the connection of the combination of the input values with the output one was calculated

for the chosen logic function (In1, In2: Out). For the NAND function, these values are: (0, 0: 1), (0, 1: 1), (1, 0: 1), (1, 1: 0).

The training was done according to the following algorithm:

- A. Application of the first combination of input signals
  1. Analysis of the output value; comparison of this value to the theoretical one (only the sign of Error =  $\text{Out}_{\text{exp}} - \text{Out}_{\text{theor}}$ ).
  2. If the error is “0,” no action is performed.
  3. If the error is “−1,” reinforcing training is applied.
  4. If the error is “+1,” inhibiting training is applied; repetition.
- B. Application of the second combination of input signals;  
Steps 1–4
- C. Application of the third combination of input signals;  
Steps 1–4
- D. Application of the fourth combination of input signals;  
Steps 1–4
- E. Repetition of steps A–D

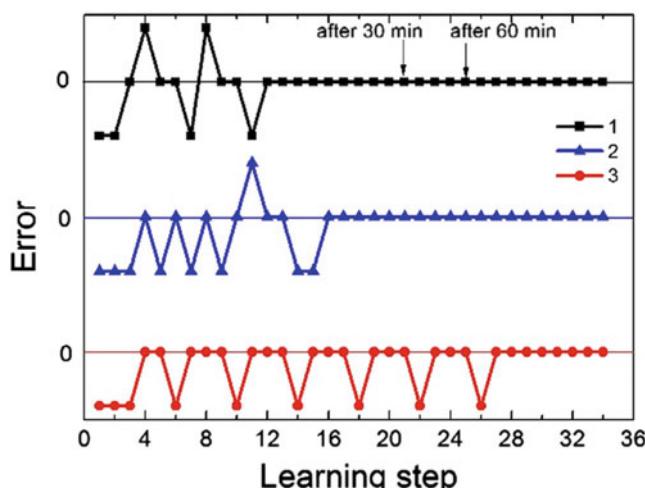
The training was stopped when all four successive steps A–D resulted in “0” error values twice.

Application of the described algorithm has demonstrated the possibility to train the organic memristive device based perceptron to perform classification according the logic function NAND. It is to note that the number of required training epochs depends on the duration of training phases. This dependence is shown in Fig. 11 for three different training times.

Learning step in Fig. 11 means the action after each A–D step (reinforcing, inhibition, or no action)

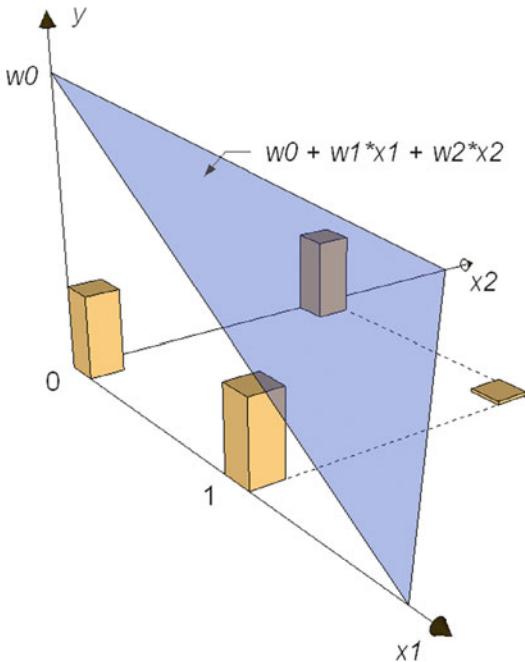
After the NAND function successful implementation, we decided to retrain the perceptron in order to realize the logic NOR function. In this case, only two steps with potentiation were required to retrain the perceptron. These steps were the (0, 1) and (1, 0) combinations on functional inputs in which the output of NOR differs from that of NAND. After these steps, the output error was zero for all incoming input combinations of NOR function.

As it was mentioned above, the suggested scheme of the perceptron can be applied for the classification of linearly separable objects. It is



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 11** Dependence of the error sign during the perceptron learning to perform NAND function on the learning step for the duration of potentiating/depressing pulses of 1–600/30 s, 2–200/20 s, and 3–150/15 s (Reprinted from V.A. Demin, V. Erokhin,

A.V. Emelyanov, S. Battistoni, G. Baldi, S. Iannotta, P.K. Kashkarov, and M.V. Kovalchuk, Hardware elementary perceptron based on polyaniline memristive devices. *Organic Electronics*, 25, 16–20 (2015) Copyright (2011) with the permission from Elsevier)

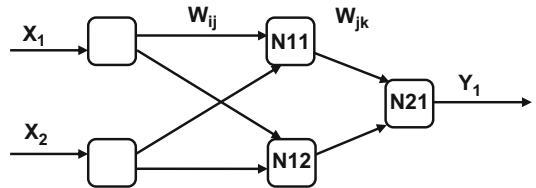


**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 12** Geometrical representation of the perceptron output (the separating plane) when it has been learned to the NAND function (Reprinted from V.A. Demin, V. Erokhin, A.V. Emelyanov, S. Battistoni, G. Baldi, S. Iannotta, P.K. Kashkarov, and M.V. Kovalchuk, Hardware elementary perceptron based on polyaniline memristive devices. *Organic Electronics*, 25, 16–20 (2015) Copyright (2011) with the permission from Elsevier)

illustrated in Fig. 12 for the objects that can be classified according NAND logic function.

As it is clear from the Fig. 12, there is a linear plain that separates one class of objects from the other one. In the case of not linearly separable objects, elementary single layer perceptron cannot be applied and we need to realize at least a double layer perceptron.

The simplest logic function that does not allow linear separation of results is XOR function. In fact, it requires the following Inputs – Output combinations \$(0, 0: 0), (0, 1: 1), (1, 0: 1), (1, 1: 0)\$. Obviously, there is no linear plain that can separate these objects. Therefore, a double layer perceptron is required for separating them after appropriate training. The simplified scheme of such perceptron is shown in Fig. 13.



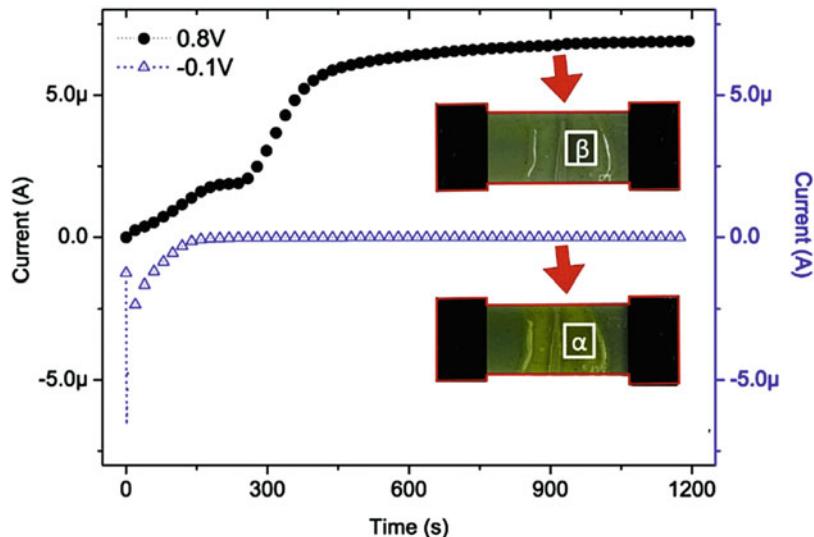
**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 13** Scheme of the implemented neural network with two inputs, two hidden, and one output neurons

The scheme in Fig. 13 differs from that in Fig. 10 by the presence of hidden layer, where weight functions can be also varied during learning (Emelyanov et al. 2016).

The realization of this scheme demands the presence of threshold elements in the middle layer. Moreover, the existing learning algorithm demands the possibility of testing the conductivity state of each connection (memristive device) after each learning epoch. Therefore, the realized circuit was rather complicated and it allowed a partial success of the performed experiment. It was experimentally demonstrated the possibility to form connections in the perceptron, allowing to perform classification according to the XOR function. Despite only partial success, this work was the very first step toward the realization of memristor-based double layer perceptron.

It is to note that polyaniline-based memristive devices have one other important advantage for the realization of multilayer perceptron, namely, the difference in color corresponding to the different conductivity state. It can allow avoiding special electronics, disconnecting all elements after each learning epoch for the testing of their conductivity. Variation of the active zone current, together with the kinetics of the conductivity switching, is shown in Fig. 14 (Battistoni et al. 2016).

The presented results give the possibility of checking the conductivity of all elements of the circuit in a contactless nondisturbing manner. Therefore, during the perceptron learning, it is possible to follow their conductivity variation in a real time. It is to note that this approach was successfully applied for the monitoring of the conductivity variation of the polyaniline thin



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 14** Kinetics of the current variation upon the application of 0.8 V (black curve) and  $-0.1$  V (blue curve), respectively. The insets are optical microscope photographs of the sample taken at the end of

each trend and revealing the active zone color variation (Reprinted from S. Battistoni, A. Dimonte, and V. Erokhin, Spectrophotometric characterization of organic memristive devices. *Organic Electronics*, 38, 79–83 (2016) Copyright (2016) with the permission from Elsevier)

layer resulted from the growth of the slime mold *Physarum polyciphala* on it (Dimonte et al. 2015).

### Stochastic Networks

Probably, the most important advantage of organic materials is the possibility of self-assembling into complex functional systems. Any living being is more complicated than any manmade device. Regarding memristive systems, this property is also very important, and we will illustrate it in this part.

As we have considered above, deterministic circuits allow to reproduce functional parts of nervous system. In particular, simple learning capabilities were reproduced for simple animals when the number of synapses is low enough. However, human brain contains about  $10^{11}$  neurons and about  $10^{15}$  synapses. Moreover, the brain is organized in a 3D system with the presence of connections not only between neighboring neurons but also between rather distant ones. Despite the great progress in electronic technology, it is still impossible to realize such architectures with inorganic materials.

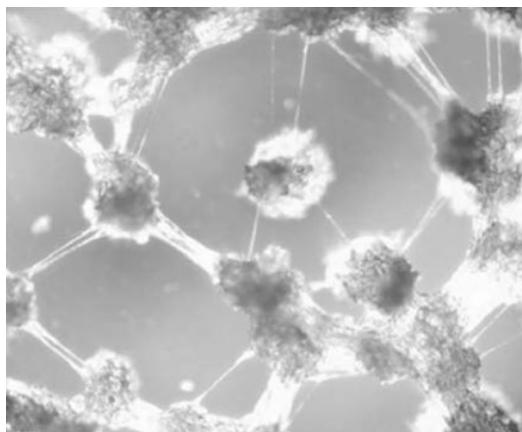
Historically, the first attempt for the realization of networks with stochastic architecture was done

on fibrillar networks, containing PEO and PANI parts (Erokhin et al. 2006). First, gel solution of PEO with already dissolved lithium perchloride was exposed to vacuum by placing samples into the vacuum chamber and its successive pumping with a mechanical pump for 15–20 min until a pressure of  $10^{-2}$  Torr. After it, PANI solution was dropped onto this sample and vacuum treatment was repeated. An optical image of the resultant sample is shown in Fig. 15.

These networks were formed on solid insulating supports with already evaporated electrodes. Reference silver wire was placed into PEO gel before the vacuum treatment.

In this very first experiment, we have demonstrated that such structure can have essential characteristics of organic memristive device, namely, hysteresis and rectification, due to the stochastically formed heterojunctions between PEO and PANI fibers. However, such structure was very unstable, heating of the free standing wires resulted in their deformation and even damage. Therefore, within 1 h the system was completely insulating one.

Better results were obtained when the active medium was formed on substrates with



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 15** Optical microscopy images of PEO–PANI fibrillar networks (image sizes are  $0.6 \times 0.5$  mm) (Reproducer from V. Erokhin, T. Berzina, P. Camorani and M.P. Fontana, Conducting polymer – solid electrolyte fibrillar composite material for adaptive networks. *Soft Matter*, 2, 870–874 (2006) with permission from Royal Society of Chemistry)

developed surface, in particular, on porous supports (Erokhina et al. 2015). First, it was demonstrated that such samples have organic memristive devices cyclic voltage-current characteristics; second, it was shown the possibility of training with improved stability.

The best results were obtained on networks, formed by the phase separation of specially synthesized block copolymers (Erokhin et al. 2012c).

A new block copolymer poly(styrene sulfonic acid)-b-poly(ethylene oxide)-b-poly(styrene sulfonic acid) (PSS-b-PEO-b-PSS) was prepared following the protocol for a block of PEO obtaining higher molecular weights to ensure greater morphological stability of the 3D matrix assembly. It is to note that casting of such copolymer resulted in the formation of the network of separated phases of preferential PEO content (electrolyte) and PSS content (insulator). PANI layers with distributed gold nanoparticles were deposited onto this matrix. The reason for the gold nanoparticles addition was the following one: work function of gold differs significantly from that of the PANI. Therefore, at a certain bias, we can expect charge trapping in them, what was also independently observed on other

organic materials (Son et al. 2011). Thus, gold nanoparticles can play a role of threshold elements: further signal propagation can occur only after overcoming a barrier, created by the trapped charges. It is to note also, that PSS was chosen as a part of copolymer, because it is an insulator, on the one hand, but, due to its acid nature, it can provide additional doping of contacting PANI, allowing better stability of its electrical properties.

Microscopic characterization (optical and SEM) of the formed systems revealed the formation of phase separated structures at the macroscopic and microscopic levels.

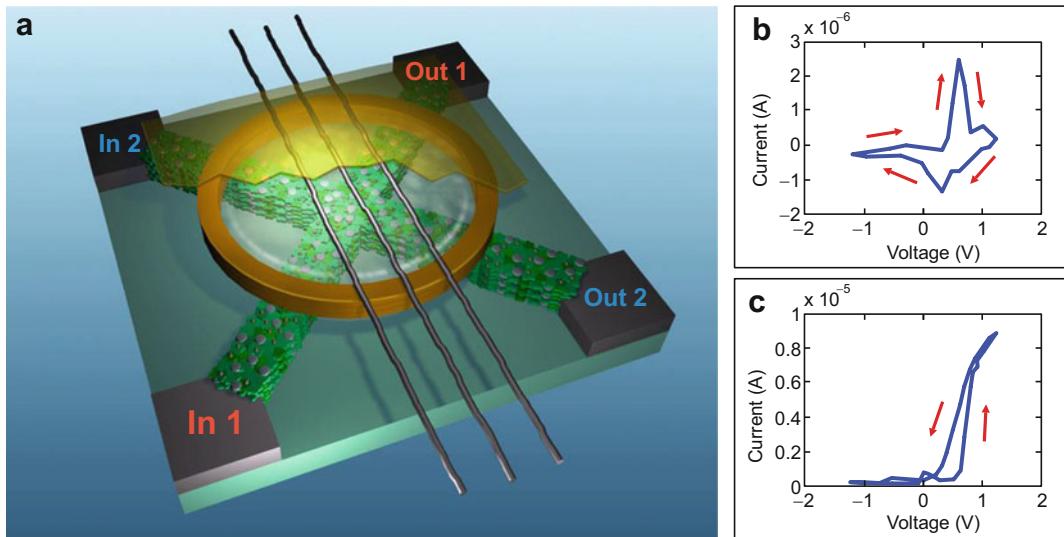
The scheme of the formed sample is shown in Fig. 16a.

The substrate was a glass support with four evaporated metal electrodes. The stochastic layer was deposited onto these substrates. The layer was patterned in order to form cross configuration between electrodes, as it is shown in Fig. 16a. An O-ring was placed in the central part of the sample, and additional PEO gel with lithium ions was placed on it and contacted with three silver wires, used as reference electrodes. The area inside the O-ring is an active zone of these samples. It was covered by the Kapton film for improving the stability. Important feature of this configuration is the absence of the contact of the metal electrodes with the active zone: it is necessary for preventing additional electrochemical processes on the metal-electrolyte interface.

Before experiments on the network learning, we have done standard cyclic V-I characteristics, using all three silver wires as reference electrodes. Characterization was done between all possible pairs of electrodes. Typical characteristics are shown in Fig. 16b for the ionic current and in Fig. 16c for the total one.

As it is clear from the figure, characteristics are very similar to those of the deterministic devices, what indicates that desirable PEO-PANI heterojunctions were really formed in the network with stochastic architecture.

Next step was learning of the network. As a target, the final state of the network must provide a high conductivity between one pair of electrodes (In 1–Out 1) and a low conductivity between the other one (In 2–Out 2).



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 16** Scheme of the sample with stochastic networks of active area (a); typical cyclic voltage-current characteristics for the ionic (b); and total (c) currents between any pair of the electrodes (arrows indicate the applied voltage variation) (Reprinted from

V. Erokhin, T. Berzina, K. Gorshkov, P. Camorani, A. Pucci, L. Ricci, G. Ruggeri, R. Sigala, and A. Schuz, Stochastic hybrid 3D matrix: learning and adaptation of electrical properties. *J. Mater. Chem.*, 22, 22881 (2012) with the permission from Royal Society of Chemistry)

Testing of the conductivity between pairs of electrodes was done by the application of +0.4 V between them. During training, the potentiation of the conductivity was done by the application of +0.8 V, while its depression was done by the application of -0.2 V.

Training of the network was done in two different ways: simultaneous and sequential protocols.

In the case of sequential training, the potentiating potential was applied between In 1 and Out 1 pair and was stopped when the current between these electrodes reached the saturation level. After it, the depressing potential was applied between In 2 and Out 2 pair of electrodes and was maintained till the current reached the saturation level.

After such training, test measurements revealed that the conductivity ratio between potentiated pairs of electrodes (In 1–Out 1) and the depressed one (In 2–Out 2) was about two orders of magnitude.

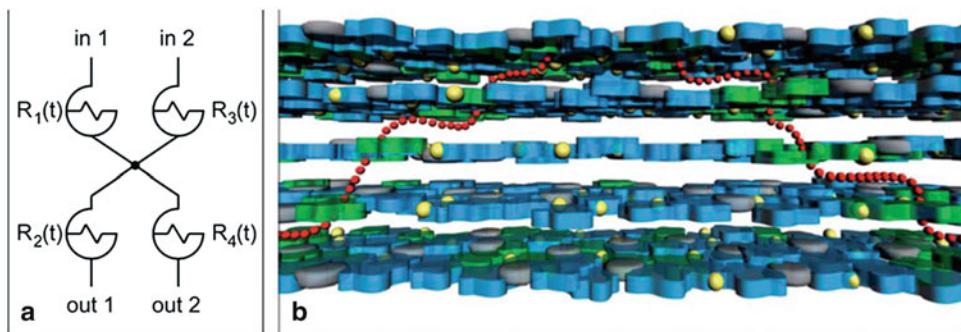
After this experiment, the effort to retrain the network was performed: potentiating voltage was applied between In 2 and Out 2 pair of electrodes, and the depressing one was applied between In

1 and Out 1 pair of electrodes. Duration of the training procedures was the same as in the case of the first training. However, it turned out to be impossible to invert the conductivity states: the training resulted only in a small variation of the conductivity between both pairs of electrodes.

Test measurements were performed immediately after training and also after a delay of 1 and 2 days (no manipulation of the sample was performed before testing). It is interesting to note that in this case the conductivity between the In 1 and –Out 1 pair of the electrodes increases again over time without any external influence on the system. Moreover, after 2 days its conductivity returned to the value that was observed before the second training procedure was started.

The results indicate that the sequential training allows to form stable signal pathways that is very difficult to destroy during successive training: the conductivity states are practically not influenced by the applied voltage variations.

Next set of experiments was done by the application of similar training algorithms but simultaneously to both pairs of electrodes. The results of



**Neuromorphic Computing Based on Organic Memristive Systems, Fig. 17** Simplified representation of the active zone in 2D (a) and formation of 3D pathways (red dots) in a self-organized matrix (b) (Reprinted from V. Erokhin, T. Berzina, K. Gorshkov, P. Camorani,

A. Pucci, L. Ricci, G. Ruggeri, R. Sigala, and A. Schuz, Stochastic hybrid 3D matrix: learning and adaptation of electrical properties. *J. Mater. Chem.*, 22, 22881 (2012) with the permission from Royal Society of Chemistry)

the training were significantly different with respect to the previous case. First, the conductivity ratio between the potentiated pathway and the depressed one was found to be one order of magnitude (one order of magnitude less with respect to the previous case). Second, the learning was found to be reversible: application of the opposite training algorithm allowed to invert the conductivity states of pathways between electrode pairs.

In order to explain the obtained results we must consider that in the case of the sequential training, learning results in a formation of a single pathway at a time, what involves the formation of stable configuration of the ion distribution and potential maps (due to the charge trapping). These potentials can act against the newly applied voltages decreasing their effect on the network.

In the case of simultaneous training, the situation is different. The application of different voltages between pairs of electrodes induces instability: we can suppose that there is a dynamic process of formation and destruction of temporal areas of low and high conductivity in the network. Thus, stable maps of potentials are not formed and the network properties can be varied by the application of the successive training process.

Such behavior allows making some analogies with the brain learning (Erokhin 2013). In the case of simultaneous training, the situation can be compared with our every day life: we have several stimuli simultaneously, resulting in the establishing of some temporal associations. However,

the system is in a dynamic equilibrium with a short-term memory effect. It can be easily varied as a result of the application of different external stimuli. It is rather similar to the “adult” learning. We can make associations according to external stimuli and memorize them, but the variation of the external stimuli can result in new associations.

In the case of sequential training we have one task only in a rather long time interval. When the first associations are established, we proceed to the successive associations. It is similar to “baby” learning (imprinting): learning one thing at a time. Thus, the system is not in a dynamic equilibrium: there are stable signal pathways, formed by multiple conducting polymer chains and potential distribution maps due to charges trapped on gold nanoparticle. Such learning results in long-term memory and associations that can be hardly varied when the external stimuli are changed.

Finally, it was necessary to demonstrate that this network with stochastic architecture allows really the formation of 3D signal pathways: previously shown cases can be done also in 2D space. For this reason, the following experiment was done (see Fig. 17).

In this case, the training algorithm (similar to the described above) was applied for having high conductivity between lateral pairs of electrodes (In 1–Out 1 and In 2–Out 2), while having a low conductivity between crossed pairs of electrodes (In 1–Out 2 and In 2–Out 1). As it is clear from the Fig. 17a, 2D organization of the network will

not allow the formation of such pathways. However, experimental results, obtained after training, have demonstrated that learning can allow at least a double increase of the conductivity between lateral pairs of electrodes with respect to the crossed ones (and vice versa). Therefore, this test has demonstrated that the system has 3D organization and can perform, after learning, associations, impossible in 2D networks.

Results, obtained on the networks with stochastic architecture, were modeled using approaches of the brain activity analysis (Sigala et al. 2013). The model was in a good agreement with the structure and properties of the realized networks.

Concluding, the shown network can be considered as a multilayer perceptron. Works on the demonstration of all advantages of such architecture are in progress.

## Conclusions and Future Directions

In this chapter, we have considered several important properties of organic memristive devices. In the conclusion, let us consider weak and strong points of these devices and try to suggest possible directions where these devices can be used.

Standard applications are, probably, the weakest point of the devices. In fact, without sharp breakthroughs, nobody will substitute modern Si-based technology. Therefore, in order to be considered by main chip manufacturers, the suggested devices must be well-compatible with the existing technological lines. In this respect, organic memristive devices do not fit these requirements. However, it is to note that even inorganic memristors, including currently developed Hf oxide memristors, the most compatible with CMOS technology, have a lot of limitations. Therefore, the integration into the ongoing technology is a weak point of all types of memristive devices.

Another weak point is the stability. Comparing to inorganic materials, it is much less stable. However, decreased stability implies an increased plasticity. Therefore, we can expect important features not on discrete devices, but on complex networks, where the instability of individual devices can be compensated by their common functioning, when

degradation of some components will be covered with the formation of alternative pathways, formed by “healthy” devices.

Regarding strong points, organic (polyaniline-based) memristive devices have several obvious advantages, such as:

- Low cost
- Low-power consumption
- Low weight
- High ON/OFF ratio ( $10^5$ , among the best available memristive devices)
- Possibility of flexible realizations
- Low-operational voltages, allowing direct interfaces with biological samples
- Electrochromism, allowing a contactless control of the conductivity state of network elements and groups of elements

In addition, organic materials have one very important advantage – they are able to form self-assembled structures. In this respect, inorganic technologies, even the most advanced ones, do not provide 3D organization of networks. Even if there are works, reporting three-layer organization of inorganic structures, it is very far from the architecture of the nervous system and brain and it does not allow long distance connections between functional elements. Organic world, instead, gives all these possibilities: we can organize networks with stochastic architectures allowing, after appropriate learning, performing the execution of specific tasks.

Very likely, the future development of the works on organic memristive devices can provide significant results in the following fields:

- Mimicking of learning process of animals
- Hardware modeling of individual and group behaviors
- Interfacing with live elements of nervous system

## Bibliography

- Asamitsu A, Tomioka Y, Kuwahara H, Tokura Y (1997) Current switching of resistive states in magnetoresistive manganites. *Nature* 388:50–52

- Baldi G, Battistoni S, Attolini G, Bosi M, Collini C, Iannotta S, Lorenzelli L, Mosca R, Ponraj JS, Verucci R, Erokhin V (2014) Logic with memory: AND gates made of organic and inorganic memristive devices. *Semicond Sci Technol* 29:104009
- Battistoni S, Dimonte A, Erokhin V (2016) Spectrophotometric characterization of organic memristive devices. *Org Electron* 38:79–83
- Berzina T, Erokhin V, Fontana MP (2007) Spectroscopic investigation of an electrochemically controlled conducting polymer – solid electrolyte junction. *J Appl Phys* 101:024501
- Berzina T, Erokhina S, Camorani P, Konovalov O, Erokhin V, Fontana MP (2009) Electrochemical control of the conductivity in an organic memristor: a time-resolved X-ray fluorescence study of ionic drift as a function of the applied voltage. *ACS Appl Mater Interfaces* 1:2115–2118
- Braitenberg V (1984) Vehicles. Experiments in synthetic psychology. MIT Press, Cambridge, MA
- Chua LO (1971) Memristor – the missing circuit element. *IEEE Trans Circuit Theory* 18:507–519
- Corinto F, Civalleri PP, Chua LO (2015) A theoretical approach to memristor devices. *IEEE J Emerg Sel Top Circuits Syst* 5:123–132
- Demin VA, Erokhin VV, Kashkarov PK, Kovalchuk MV (2014) Electrochemical model of the polyaniline based organic memristive device. *J Appl Phys* 116:064507
- Demin VA, Erokhin V, Emelyanov AV, Battistoni S, Baldi G, Iannotta S, Kashkarov PK, Kovalchuk MV (2015) Hardware elementary perceptron based on polyaniline memristive devices. *Org Electron* 25:16–20
- Dimonte A, Fermi F, Berzina T, Erokhin V (2015) Spectral imaging method for studying *Physarum polycephalum* growth on polyaniline surface. *Mater Sci Eng C* 53:11–14
- Emelyanov AV, Lapkin DA, Demin VA, Erokhin VV, Battistoni S, Baldi G, Dimonte A, Korovin AN, Iannotta S, Kashkarov PK, Kovalchuk MV (2016) First step towards the realization of a double layer perceptron based on organic memristive device. *AIP Adv* 6:111301
- Erokhin V (2007) Polymer-based adaptive networks. In: Erokhin V, Ram MK, Yavuz O (eds) The new frontiers of organic and composite nanotechnologies. Elsevier, Oxford, Amsterdam, pp 287–353
- Erokhin V (2013) On the learning of stochastic networks of organic memristive devices. *Int J Unconv Comput* 9:303–310
- Erokhin V (2015) Bioelectronics brain using memristive polymer statistical systems. In: Carrara S, Iniewski K (eds) Handbook of bioelectronics. Cambridge University Press, Cambridge. pp 256–265
- Erokhin V, Fontana MP (2011) Thin film electrochemical memristive systems for bio-inspired computation. *J Comput Theor Nanosci* 8:313–330
- Erokhin V, Berzina T, Fontana MP (2005) Hybrid electronic device based on polyaniline-polyethylene oxide junction. *J Appl Phys* 97:064501
- Erokhin V, Berzina T, Camorani P, Fontana MP (2006) Conducting polymer – solid electrolyte fibrillar composite material for adaptive networks. *Soft Matter* 2:870–874
- Erokhin V, Berzina T, Fontana MP (2007a) Polymeric elements for adaptive networks. *Crystallogr Rep* 52:159–166
- Erokhin V, Berzina T, Camorani P, Fontana MP (2007b) Non-equilibrium electrical behavior of polymeric electrochemical junctions. *J Phys Condens Matter* 19:205111
- Erokhin V, Berzina T, Camorani P, Fontana MP (2008) On the stability of polymeric electrochemical elements for adaptive networks. *Colloids Surf A* 321:218–221
- Erokhin V, Berzina T, Camorani P, Smerieri A, Vavoulis D, Feng J, Fontana MP (2012a) Material memristive device circuit with synaptic plasticity: learning and memory. *BioNanoScience* 1:24–30
- Erokhin V, Howard GD, Adamatsky A (2012b) Organic memristor devices for logic elements with memory. *Int J Bifurcat Chaos* 22:1250283
- Erokhin V, Berzina T, Gorshkov K, Camorani P, Pucci A, Ricci L, Ruggeri G, Sigala R, Schuz A (2012c) Stochastic hybrid 3D matrix: learning and adaptation of electrical properties. *J Mater Chem* 22:22881–22887
- Erokhina S, Sorokin V, Erokhin V (2015) Skeleton-supported stochastic networks of organic memristive devices: adaptation and learning. *AIP Adv* 5:027129
- Gupta I, Serb A, Khiat A, Zeitler R, Vassanelli S, Prodromakis T (2016) Real-time encoding and compression of neuronal spikes by metal-oxide memristors. *Nat Commun* 7:12805
- Hebb DO (1961) The organization of behavior: a neuro-psychological theory, 2nd edn. Wiley, New York
- Kang ET, Neoh KG, Tan KL (1998) Polyaniline: a polymer with many interesting intrinsic redox states. *Prog Polym Sci* 23:277–324
- Kim TH, Jang EY, Lee NJ, Choi DJ, Lee KJ, Jang J, Choi J, Moon SH, Cheong J (2009) Nanoparticle assemblies as memristors. *Nano Lett* 9:2229–2233
- Klemenes I, Straub VA, Nikitin ES, Staras K, O’Shea M, Klemenes G, Benjamin PR (2006) Role of delayed nonsynaptic neuronal plasticity in long-term associative memory. *Curr Biol* 16:1269–1279
- Minsky M, Papert S (1969) Perceptrons. MIT Press, Cambridge
- Pershin YV, Di Ventra M (2008) Spin memristive systems: spin memory effects in semiconductor spintronics. *Phys Rev V* 78:11309
- Pershin YV, Di Ventra M (2011) Memory effects in complex materials and nanoscale systems. *Adv Phys* 60:145–227
- Prezioso M, Merrikh Bayat F, Hoskins BD, Adam GC, Likharev KK, Strukov DB (2015) Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521:61–64
- Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol Rev* 65:386–408

- Rosenblatt F (1961) Principles of neurodynamics: perceptions and the theory of brain mechanism. Spartan Books, Washington, DC
- Schrodinger E (1944) What is life? Physical aspects of the living cells. Cambridge University Press, Cambridge
- Sigala R, Smerieri A, Schuz A, Camorani P, Erokhin V (2013) Modeling and simulating the adaptive electrical properties of stochastic polymeric 3D networks. *Model Simul Mater Sci Eng* 21:075007
- Smerieri A, Berzina T, Erokhin V, Fontana MP (2008a) Polymeric electrochemical element for adaptive networks: pulse mode. *J Appl Phys* 104:114513
- Smerieri A, Erokhin V, Fontana MP (2008b) Origin of current oscillations in a polymeric electrochemically controlled element. *J Appl Phys* 103:094517
- Smerieri A, Berzina T, Erokhin V, Fontana MP (2008c) A functional polymeric material based on hybrid electrochemically controlled junctions. *Mater Sci Eng C* 28:18–22
- Son DI, Park DH, Kim JB, Choi JW, Kim TW, Angadi B, Yi Y, Choi WK (2011) Bistable organic memory device with gold nanoparticles embedded in a conducting poly(*N*-vinylcarbazole) colloids hybrid. *J Phys Chem C* 115:2341–2348
- Straub VA, Benjamin PR (2001) Extrinsic modulation and motor pattern generation in a feeding network: a cellular study. *J Neurosci* 21:1767–1778
- Strukov DB, Snider GS, Steward DR, Williams RS (2008) The missing memristor found. *Nature* 453:80–83
- Wang L, Duan M, Duan S (2013) Memristive perceptron for combinational logic classification. *Math Probl Eng* 2013:625790
- Waser R, Aono M (2007) Nanoionic-based resistive switching memories. *Nat Mater* 6:833–840
- Wasserman PD (1989) Neural computing: theory and practice. Van Nostrand Reinhold, New York
- Ziegler M, Soni R, Patelczyk T, Ignatov M, Bartsch T, Meuffels Kohlstedt PH (2012) An electronic version of Pavlov's dog. *Adv Funct Mater* 22:2744–2749



---

## Slime Mold Computing

Andrew Adamatzky

Unconventional Computing Centre, University of the West of England, Bristol, UK

### Article Outline

#### Glossary

Introduction

Optimization and Graphs

Geometry

Computing Circuits

Mathematical Models and Algorithms Inspired by

*Physarum* but Never Implemented in

Laboratory Experiments

Future Directions

Bibliography

### Glossary

***Physarum polycephalum*** belongs to the order Physarales, subclass Myxogastromycetidae, class Myxomycetes, and division Myxostelida; it is commonly known as a true, acellular, or multiheaded slime mold.

**Shortest path** is a sequence of edges connecting two vertexes in a graph that has a minimal sum of edge weights; in context of the paper, shortest path is determined by sum of distances on Euclidean plane.

**Maze** is a collection of all possible paths between two points.

**Travelling salesman problem** aims to find a shortest path in a graph that visits all nodes and ends in its starting node.

**Spanning tree** of a finite planar set is a connected, undirected, acyclic planar graph, whose vertices are points of the planar set; the tree is a minimal spanning tree where sum of edge lengths is minimal.

**Voronoi diagram** of a set of points is a partition of the plane into such regions that, for any element of the set, a region corresponding to a unique point contains all those points of the plane which are closer to this point than to any other point of the set.

**Delaunay triangulation** of a planar set is a triangulation of the set such that a circumcircle of any triangle does not contain a point of the set; Delaunay triangulation is a dual of Voronoi diagram.

**$\alpha$ -Shape** of a planar set is an intersection of the complements of all closed discs of radius  $\frac{1}{\alpha}$  that includes no points of the set.

**Concave hull** is a connected  $\alpha$ -shape without holes.

**Kolmogorov-Uspensky machine** is a mathematical machine, defined on a dynamically changing graph with distinctly labeled nodes, where a computational process is represented by rules for removing and adding nodes and edges depending on the node labels.

### Introduction

Acellular slime mold *P. polycephalum* has quite sophisticated life cycle (Stephenson et al. 1994), which includes fruit bodies, spores, single-cell amoebas, and syncytium. At one phase of its cycle, the slime mold becomes a plasmodium. The plasmodium is a coenocyte: nuclear divisions occur without cytokinesis. It is a single cell with thousands of nuclei. The plasmodium is a large cell. It grows up to 10 cm when conditions are good. The plasmodium consumes microscopic particles and bacteria. During its foraging behavior, the plasmodium spans scattered sources of nutrients with a network of protoplasmic tubes. The plasmodium optimizes its protoplasmic network to cover all sources of nutrients, stay away from repellents, and minimize transportation of metabolites inside its body. The plasmodium's ability to optimize its shape attracted the attention

of biologists (Nakagaki et al. 2001), then computer scientists (Adamatzky 2010a), and engineers. Thus, the field of slime mold computing was born.

So far, the plasmodium is the only useful for computation stage of *P. polycephalum*'s life cycle. Therefore, further we will use word “*Physarum*” when referring to the plasmodium. Most computing and sensing devices made of the *Physarum* explore one or more key features of the *Physarum*'s physiology and behavior:

- The slime mold senses gradients of chemoattractants and repellents (Durham and Ridgway 1976; Rakoczy 1963; Ueda et al. 1976); it responds to chemical or physical stimulation by changing patterns of electrical potential oscillations (Kishimoto 1958; Ridgway and Durham 1976) and protoplasmic tubes contractions (Teplov et al. 1991; Wohlfarth-Bottermann 1979).
- It optimizes its body to maximize its protoplasm streaming (Dietrich 2015).
- It is made of hundreds, if not thousands, of biochemical oscillators (Kauffman and Wille 1975) with varied modes of coupling (Grebecki and Cieślawska 1978).

Here we offer very short descriptions of actual working prototypes of *Physarum*-based sensors, computers, actuators, and controllers. Details can be found in pioneer book on *Physarum* machines (Adamatzky 2010a) and the “bible” of slime mold computing (Adamatzky 2016).

## Optimization and Graphs

### Shortest Path and Maze

Given a maze, we want to find a shortest path between the central chamber and an exit. This was the first-ever problem solved by *Physarum*. There are two *Physarum* processors which solve the maze. First prototype (Nakagaki et al. 2001) works as follows: The slime mold is inoculated everywhere in a maze. The *Physarum* develops a network of protoplasmic tubes spanning all

channels of the maze. This network represents all possible solutions. Then oat flakes are placed in a source and a destination site. Tube lying along the shortest (or near shortest) path between two sources of nutrients develop increased flow of cytoplasm. This tube becomes thicker. Tubes branching to sites without nutrients become smaller due to lack of cytoplasm flow. They eventually collapse. The sickest tube represents the shortest path between the sources of nutrients. The selection of the shortest protoplasmic tube is implemented via interaction of propagating biochemical, electric potential, and contractile waves in the plasmodium's body; see mathematical model in Tero et al. (2006). The approach is not efficient because we must literally distribute the computing substrates everywhere in the physical representation of the problem. A number of computing elements would be proportional to a sum of lengths of the maze's channels.

Second prototype of the *Physarum* maze solver is based on *Physarum*'s chemoattraction (Adamatzky 2012b). An oat flake is placed in the central chamber. The *Physarum* is inoculated somewhere in a peripheral channel. The oat flake releases chemoattractants. The chemoattractants diffuse along the maze's channels. The *Physarum* explores its vicinity by branching out protoplasmic tubes into opening of nearby channels. When a wave front of diffusing attractants reaches *Physarum*, the *Physarum* halts lateral exploration. Instead it develops an active growing zone propagating along gradient of the attractants' diffusion. The sickest tube represents the shortest path between the sources of nutrients (Fig. 1). The approach is efficient because a number of computing elements is proportional to the length of the shortest path.

### Towers of Hanoi

Given  $n$  discs, each of unique size, and three pegs, we want to move the entire stack to another peg by moving one top disc at a time and not placing a disc on top of smaller disc. The set of all possible configurations and moves of the puzzle forms a planar graph with  $3n$  vertices. To solve the puzzle, one must find shortest paths between configurations of pegs with discs on the graph (Hinz 1989, 1992; Romik 2006). *Physarum* solves shortest



**Slime Mold Computing, Fig. 1** *Physarum* solves maze

path (section “[Shortest Path and Maze](#)”); therefore, it can solve Tower of Hanoi puzzle. This is experimentally demonstrated in Reid and Beekman (2013). Sometimes *Physarum* does not construct an optimal path initially. However, if its protoplasmic networks are damaged and then allowed to regrow the path closer to optimal then before develops (Reid and Beekman 2013).

### Travelling Salesman Problem

Given a graph with weighted edges, find a cyclic route on the graph, with a minimum sum of edge weights, spanning all nodes, where each node is visited just once. Commonly, weight of an edge is a Euclidean length of the edge. *Physarum* is used as component of an experimental device approximating the shortest cyclic route (Zhu et al. 2013). A map of eight cities is considered. A set of solutions is represented by channels arranged in a star graph. The channels merge in a central chamber. There are eight channels for each city. Each channel encodes a city and the step when the city appears in the route. There are 64 channels. *Physarum* is inoculated in the central chamber. The slime mold then propagates into the channels. A node of the data graph is assumed to be visited when its corresponding channel is colonized by *Physarum*. Growth of the *Physarum* in the star shape is controlled by a recurrent neural network.

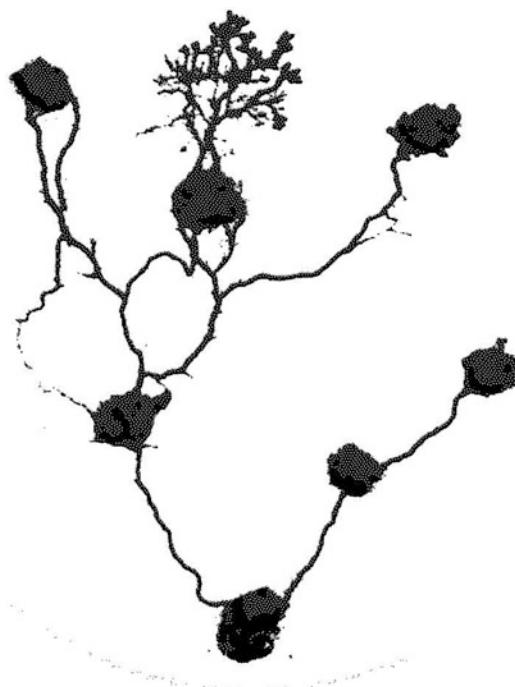
The network takes ratios of colonizations of the channels as input and produces patterns of illumination, projected onto the channels as output. The network is designed to prohibit revisiting of already visited nodes and simultaneous visits to multiple nodes. The *Physarum* propagates into channels and pulls back and then propagates to other channels and pulls back from some of them. Eventually the system reaches a stable solution where no propagation occurs. The stable solution represents the minimal distance cyclic route on the data graph (Zhu et al. 2013).

A rather more natural approximate algorithm of solving the travelling salesman problem is based on a construction of an  $\alpha$ -shape (section “[Concave Hull](#)”). This is how humans solve the problem visually (MacGregor and Ormerod 1996). An approximate solution of the travelling salesman problem by a shrinking blob of simulated *Physarum* is proposed in Jones and Adamatzky (2014a). The *Physarum* is inoculated all over the convex hull of the data set. The *Physarum*’s blob shrinks. It adapts morphologically to the configuration of data nodes. The shrinkage halts when the *Physarum* no longer covers all data nodes. The algorithm is not implemented with real *Physarum*.

### Spanning Tree

A spanning tree of a finite planar set is a connected, undirected, acyclic planar graph, whose vertices are points of the planar set. The tree is a minimal spanning tree where sum of edge lengths is minimal (Nesetril et al. 2001). As algorithm for computing a spanning tree of a finite planar set based on morphogenesis of a neuron’s axonal tree was initially proposed (Adamatzky 1991), planar data points are marked by attractants (e.g., neurotrophins), and a neuroblast is placed at some site. Growth cones sprout new filopodia in the direction of maximal concentration of attractants. If two growth cones compete for the same site of attractants, then a cone with highest energy (closest to previous site or branching point) wins. Fifteen years later, we implemented the algorithm with *Physarum* (Adamatzky 2008).

Degree of *Physarum* branching is inversely proportional to a quality of its substrate.



**Slime Mold Computing, Fig. 2** *Physarum* approximates a spanning tree

Therefore, to reduce a number of random branches, we cultivate *Physarum* not on agar but just humid filter paper. Planar data set is represented by a configuration of oat flakes. *Physarum* is inoculated at one of the data sites. *Physarum* propagates to a virgin oat flake closest to the inoculation site. *Physarum* branches, if there are several virgin flakes nearby. It colonizes the next set of flakes. The propagation goes on until all data sites are spanned by a protoplasmic network. The protoplasmic network approximates the spanning tree (Fig. 2). The resultant tree does not remain static though. Later cycles can be formed, and the tree is transformed to one of proximity graphs, e.g., relative neighborhood graph or Gabriel graph (Adamatzky 2009).

#### Approximation of Transport Networks

Motorway networks are designed with an aim of efficient vehicular transportation of goods and passengers. *Physarum* protoplasmic networks evolved for efficient intracellular transportation of nutrients and metabolites. To uncover

similarities between biological and man-made transport networks and to project behavioral traits of biological networks onto development of vehicular transport networks, we conducted an evaluation and approximation of motorway networks by *Physarum* in 14 geographical regions: Africa, Australia, Belgium, Brazil, Canada, China, Germany, Iberia, Italy, Malaysia, Mexico, the Netherlands, the UK, and the USA (Adamatzky 2012a).

We represented each region with an agar plate, imitated major urban areas with oat flakes, inoculated *Physarum* in a capital, and analyzed structures of protoplasmic networks developed. We found that the networks of protoplasmic tubes grown by *Physarum* match, at least partly, the networks of man-made transport arteries. The shape of a country and the exact spatial distribution of urban areas, represented by sources of nutrients, play a key role in determining the exact structure of the plasmodium network. In terms of absolute matching between *Physarum* networks and motorway networks, the regions studied can be arranged in the following order of decreasing matching: Malaysia, Italy, Canada, Belgium, China, Africa, the Netherlands, Germany, the UK, Australia, Iberia, Mexico, Brazil, and the USA. We compared the *Physarum* and the motorway graphs using such measures as average and longest shortest paths, average degrees, number of independent cycles, the Harary index, the  $\Pi$ -index, and the Randic' index. Using these measures, we find that motorway networks in Belgium, Canada, and China are most affine to *Physarum* networks. With regard to measures and topological indices, we demonstrated that the Randic' index could be considered as most biocompatible measure of transport networks, because it matches very well the slime mold and man-made transport networks yet efficiently discriminates between transport networks of different regions (Adamatzky et al. 2013a).

Many curious discoveries have been made. Just few of them are listed below. All segments of trans-African highways not represented by *Physarum* have components of non-paved roads (Fig. 3) (Adamatzky and Kayem 2013). The east coast transport chain from the Melbourne urban area in the south to the Mackay area in the north, and the



**Slime Mold Computing, Fig. 3** *Physarum* approximates trans-African highways

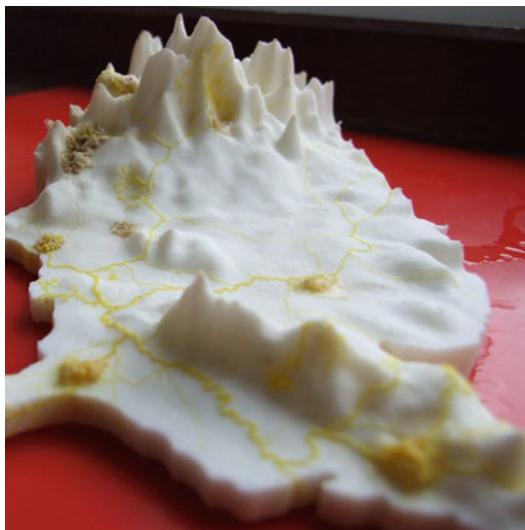
highways linking Alice Springs and Mount Isa and Cloncurry, is represented by the slime mold's protoplasmic tubes in almost all experiments on approximation of Australian highways (Adamatzky and Prokopenko 2012). If the two parts of Belgium were separated with Brussels in Flanders, the Walloon Region of the Belgian transport network would be represented by a single chain from Tournai in the northwest to the Liege area in the northeast and down to southernmost Arlon; motorway links connecting

Brussels with Antwerp, Tournai, Mons, Charleroi, and Namur, and links connecting Leuven with Liege and Antwerp with Genk and Turnhout, are redundant from the *Physarum*'s point of view (Adamatzky et al. 2012). The protoplasmic network forms a sub-network of the man-made motorway network in the Netherlands; a flooding of large area around Amsterdam will lead to substantial increase in traffic at the boundary between flooded and non-flooded area, paralysis and abandonment of the transport network,

and migration of population from the Netherlands to Germany, France, and Belgium (Adamatzky et al. 2013b). *Physarum* imitates the separation of Germany into East Germany and West Germany (Adamatzky and Schubert 2012) in the 1947.

### Mass Migration

People migrate toward sources of safe life and higher income. *Physarum* migrates into environmentally conformable areas and toward the source of nutrients. In Adamatzky and Martinez (2013), we explored this analogy to imitate Mexican migration to the USA. We have made a 3D Nylon terrain of the USA and placed oat flakes, to act as sources of attractants and nutrients, to ten areas with highest concentration of migrants: New York, Jacksonville, Chicago, Dallas, Houston, Denver, Albuquerque, Phoenix, Los Angeles, and San Jose. We inoculated *Physarum* in a locus between Ciudad Juárez and Nuevo Laredo, allowed it to colonize the template for 5–10 days, and analyzed routes of migration (Fig. 4). From the results of laboratory experiments, we extracted topologies of migratory routes and highlighted a role of elevations in shaping the human movement networks.



**Slime Mold Computing, Fig. 4** *Physarum* approximates migration routes from Mexico to the USA

### Experimental Archeology

Experimental archeology uses analytical methods, imaginative experiments, or transformation of a matter (Ascher 1961; Coles 1979; Ingersoll et al. 1977) in a context of human activity in the past. Knowing that *Physarum* is a fruitful substrate for simulating transport routes (section “[Approximation of Transport Networks](#)”) and migration (section “[Mass Migration](#)”), we explored foraging behavior of the *Physarum* to imitate development of Roman roads in the Balkans (Evangelidis et al. 2015). Agar plates were cut in a shape of Balkans area. We placed oat flakes in 17 areas, corresponding to Roman provinces and major settlements, and inoculated *Physarum* in Thessaloniki. We found that *Physarum* imitates growth of Roman roads to a larger extent. For example, the propagation of *Physarum* from Thessaloniki toward the area of Scopje and Stoboi matches the road aligned with the Valley of Axios, which is a key communication artery between the Balkan hinterland and Aegean area from Bronze Age. A range of historical scenarios was uncovered (Evangelidis et al. 2015), including movement along via diagonalis, the long diagonal axis that crossed central Balkan; propagation to the East toward Byzantium, toward the North along the coast of Euxinus Pontus, and from Thessaloniki to Dyrrachium, along the western part of Via Egnatia (Evangelidis et al. 2015).

### Evacuation

Evacuation is a rapid but temporary removal of people from the area of danger. *Physarum* moves away from sources of repellents or areas of uncomfortable environmental conditions. Would *Physarum* be able to find a shortest route of evacuation in geometrically constrained environment? To find out, we undertook a series of experiments (Kalogeiton et al. 2015b). We made a physical, scale-down model of a whole floor of the real office building and inoculated *Physarum* in one of the rooms. In first scenario, we placed a crystal of a salt in the room with *Physarum* in a hope that a gradient of sodium chloride diffusion would repel *Physarum* toward exit of the building along the shortest path. This did not happen. *Physarum* got lost in the template. By placing

attractants near the exit, we rectified the mishap and allowed *Physarum* to find a shortest route away from the “disaster” (Kalogeiton et al. 2015b). Evacuation is one of few problems where computer models of *Physarum* find better solution than the living *Physarum* (Kalogeiton et al. 2015a).

### Space Exploration

We employed the foraging behavior of the *Physarum* to explore scenarios of future colonization of the Moon and the Mars (Adamatzky et al. 2014). We have grown *Physarum* on three-dimensional templates of this planet and analyzed formation of the exploration routes, dynamical reconfiguration of the transportation networks as a response to addition of hubs (Fig. 5). The developed infrastructures were explored using proximity graphs and *Physarum*-inspired algorithms of supply chain designs. Interesting insights about how various lunar missions will develop and how interactions between hubs and landing sites can be established are given in Adamatzky et al. (2014).



**Slime Mold Computing, Fig. 5** *Physarum* imitates exploration of the Moon

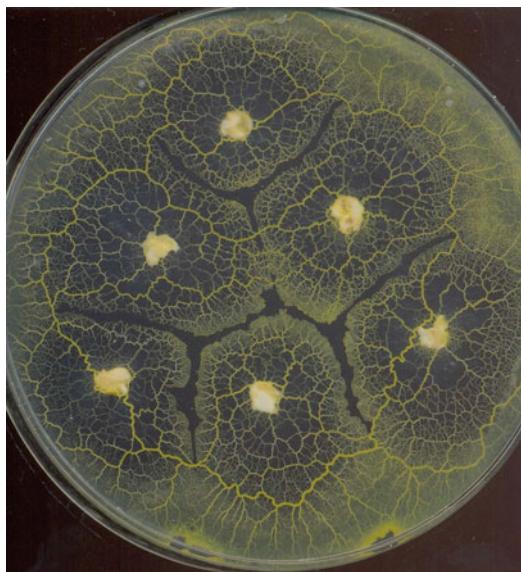
## Geometry

### Voronoi Diagram

Let  $P$  be a nonempty finite set of planar points. A planar Voronoi diagram of the set  $P$  is a partition of the plane into such regions that, for any element of  $P$ , a region corresponding to a unique point  $p$  contains all those points of the plane which are closer to  $p$  than to any other node of  $P$ . A unique region  $\text{vor}(p) = \{z \in \mathbb{R}^2 : d(p, z) < d(p, m) \forall m \in P, m \neq p\}$  assigned to the point  $p$  is called a Voronoi cell of the point  $p$ . The boundary of the Voronoi cell of the point  $p$  is built of segments of bisectors separating pairs of geographically closest points of the given planar set  $P$ . A union of all boundaries of the Voronoi cells determines the planar Voronoi diagram:  $(VD(P)) = \bigcup_{p \in P} \partial \text{vor}(p)$  (Preparata and Shamos 1985).

The basic concept of constructing Voronoi diagrams with reaction-diffusion systems is based on an intuitive technique for detecting the bisector points separating two given points of the set  $P$ . If we drop reagents at the two data points, the diffusive waves, or phase waves if the computing substrate is active, travel outward from the drops. The waves travel the same distance from the sites of origin before they meet one another. The points where the waves meet are the bisector points (Adamatzky et al. 2005).

Plasmodium growing on a nutrient substrate from a single site of inoculation expands circularly as a typical diffusive or excitation wave. When two plasmodium waves encounter each other, they stop propagating. To approximate a Voronoi diagram with *Physarum* (Adamatzky 2010a), we physically map a configuration of planar data points by inoculating plasmodia on a substrate. Plasmodium waves propagate circularly from each data point and stop when they collide with each other. Thus, the plasmodium waves approximate a Voronoi diagram, whose edges are the substrate’s loci not occupied by plasmodia (Fig. 6). Time complexity of the *Physarum* computation is proportional to a maximal distance between two geographically neighboring data points, which is capped by a diameter of the data planar set and does not depend on a number of the data points.



**Slime Mold Computing, Fig. 6** *Physarum* approximates Voronoi diagram

### Delaunay Triangulation

Delaunay triangulation is a dual graph of Voronoi diagram. A Delaunay triangulation of a planar set is a triangulation of the set such that a circumcircle of any triangle does not contain a point of the set (Delaunay 1934). There are two ways to approximate the Delaunay triangulation with *Physarum*. First is based on the setup of Voronoi diagram processor (section “[Voronoi Diagram](#)”). Previously, we wrote that when propagating fronts of *Physarum* meet, they stop. This is true. But what happens after they stop is equally interesting. *Physarum* forms a bridge – a single protoplasmic tube – connecting the stationary *Physarum* fronts. Such protoplasmic tubes typically span geographically in the neighboring sites of inoculation, and they cross sites of first contacts of growing wave fronts. These tubes represent edges of the Delaunay triangulation. This is why we proposed in Shirakawa et al. (2009) that the Voronoi diagram and the Delaunay triangulation are constructed simultaneously by *Physarum* growing on a nutrient agar.

Second method of approximating the Delaunay triangulation is implemented on a non-nutrient agar. We represent planar data points by inoculation sites. The inoculants propagate and

form a planar proximity graph spanning all inoculation sites. At the beginning of such development, a Gabriel graph (Gabriel and Sokal 1969) is formed. Then additional protoplasmic tubes emerge, and the Gabriel graph is transformed to the Delaunay triangulation (Adamatzky 2009).

### Concave Hull

The  $\alpha$ -shape of a planar set  $P$  is an intersection of the complement of all closed discs of radius  $1/\alpha$  that includes no points of  $P$  (Edelsbrunner et al. 1983). A concave hull is a connected  $\alpha$ -shape without holes. This is a non-convex polygon representing area occupied by  $P$ . Given planar set  $P$  represented by physical objects, *Physarum* must approximate concave hull of  $P$  by its thickest protoplasmic tube. We represent data points by somniferous pills placed directly on a non-nutritive agar. The pills emit attractants to “pull” *Physarum* toward  $P$ , but they also emit repellents preventing *Physarum* from growing inside  $P$  (Adamatzky 2012c). The combination of long-distance attracting forces and short-distance (“short” is  $O(D)$ , where  $D$  is a diameter of  $P$ ) repelling forces allows us to implement Jarvis wrapping algorithm (Jarvis 1973). We select a starting point which is extremal point of  $P$ . We pull a rope to other extremal point. We continue until the set  $P$  is wrapped completely. We tested feasibility of the idea in laboratory experiments (Adamatzky 2012c). In each experiment, we arranged several half pills in a random fashion near the center of a Petri dish and inoculated an oat flake colonized by *Physarum* few centimeters away from the set  $P$ . *Physarum* propagates toward set  $P$  and starts enveloping the set with its body and the network of protoplasmic tubes. The plasmodium does not propagate inside configuration of pills. The plasmodium completes approximation of a shape by entirely enveloping  $P$  in a day or two (Fig. 7).

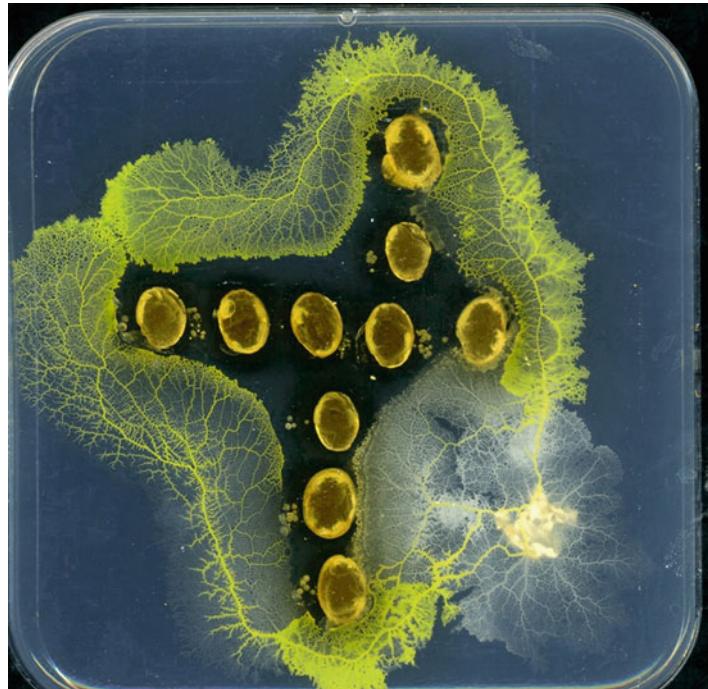
### Computing Circuits

#### Attraction-Based Logical Gates

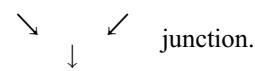
When two growing zones of separate *Physarum* cells meet, they repel if there is a free space to

**Slime Mold Computing,****Fig. 7** *Physarum*

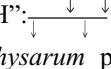
approximates concave hull



deviate to. If there is no opportunity to deviate, the cells merge. This feature is employed in the construction of Boolean logical gates – NOT, OR and AND (Tsuda et al. (2004)). The gates are made of segments of agar gel along which the *Physarum* propagates. To implement input “1” (TRUE) in channel  $x$ , a piece of *Physarum* is inoculated in  $x$ ; otherwise, the input is considered to be “0” (FALSE). Attractants are placed in the end of the output channels to stimulate growth of the *Physarum* toward outputs. The *Physarum* propagates toward closest source of attractants along a shortest path.

The gate OR is a  junction.

*Physarum* placed in one of the inputs propagates toward the output. If each input contains the *Physarum*, the propagating cells merge and appear in the output as if they were a single cell. Even if both inputs are “1,” the *Physarum* cells have no space to avoid collision, and therefore they merge and propagate into the output channel.

The gate AND looks like distorted “H”:  When only one input is “1,” the *Physarum* propagates toward closest attractant

and exits along right output channel. When both inputs are “1,” the *Physarum* from the right input channel propagates into the right output channel. The *Physarum* from the left input channel avoids merging with another *Physarum* and propagates toward left output channel. The left output channel realizes AND.

**Ballistic Logical Gates**

In designs of ballistic gates (Adamatzky 2010b), we employ inertia of the *Physarum* growing zones. On a non-nutrient substrate, the plasmodium propagates as a traveling localization, as a compact wave fragment of protoplasm. The plasmodium localization travels in its originally pre-determined direction for a substantial period of time even when no gradient of chemoattractants is present. We explore this feature of *Physarum* localizations to design a two-input two-output Boolean gates. The gate realizing AND on one output and OR on another output looks like horizontally flipped “K”:  When left input is “1,” the *Physarum* propagates inertially along the vertical (on the right) output channel. The same

happens when right input is “1.” If both inputs are “1,” then the *Physarum* from the right input propagates along vertical output channel, but the *Physarum* from the left input repels from the right input *Physarum* and moves into the left output channel. The left output channel realizes AND and the right output channel realizes OR.

The gate NOT is an asymmetric cross junction:  
 $\downarrow$   
 $\rightarrow \downarrow \rightarrow$ . Vertical input channel is twice as long as horizontal input channel. Vertical input is constant TRUE: *Physarum* is always inoculated there. Horizontal input is a variable. If variable input is “0” then *Physarum* from the constant TRUE vertical input propagates into the vertical output. If variable input is “1,” then *Physarum* from the input channel propagates into the horizontal output channel and blocks path of the *Physarum* representing constant TRUE. Both ballistic gates work very well without attractants. However, they work even better when attractants are placed into output channels. Cascading of the gates into a binary adder is demonstrated in Adamatzky (2010b).

### Optoelectronics Logical Gates

In prototypes of repellent gates (Mayne and Adamatzky 2015a), active growing zones of slime mold representing different inputs interact with each other by electronically switching light inputs and thus invoking photo avoidance. The gates NOT and BAND are constructed using this feature.

The gate NOT is made of two electrodes. The electrodes are connected to a power supply. There is a green LED (with its own independent power supply) on one electrode. The *Physarum* is inoculated on another electrode. Input “1” is represented by LED’s light on. Output is represented by the *Physarum* closing the circuit between two electrodes. When LED is illuminated (input “1”), the *Physarum* does not propagate between electrodes; thus, output “0” is produced. When LED is off (input “0”), the *Physarum* closes the circuit by propagating between the electrodes.

The gate NAND is implemented with two LEDs. When both inputs are “0,” LEDs are off and the *Physarum* closes the circuit between its inoculation electrode and one of the LED

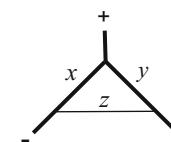
electrodes, chosen at random. When both inputs are “1” and both LEDs are on, they repel *Physarum*. The *Physarum* does not propagate to any electrodes. When only one input is “1,” one LED is on, and another LED is off, the *Physarum* propagates toward the electrode with non-illuminating LED and closes the circuit.

### Frequency Based Logical Gates

The *Physarum* responds to stimulation with light, nutrients, and heating by changing frequency of its electrical potential oscillations. We represent TRUE and FALSE values by different types of stimuli and apply threshold operations to frequencies of the *Physarum* oscillations. We represent Boolean inputs as intervals of oscillation frequency (Whiting et al. 2014). Thus, we experimentally implement OR, AND, NOT, NOR, NAND, XOR, and XNOR gates; see details in Whiting et al. (2014).

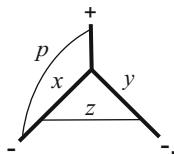
### Micro-fluidic Logical Gates

When a fragment of protoplasmic tube is mechanically stimulated, e.g., gently touched by a hair, a cytoplasmic flow in this fragment halts, and the fragment’s resistivity to the flow dramatically increases. The cytoplasmic flow is then directed via adjacent protoplasmic tubes. A basic gate looks like a “Y” junction of two protoplasmic tubes  $x$  and  $y$  with a horizontal bypass  $z$  between them.



Segments  $x$  and  $y$  are inputs. Segment  $z$  is output. When both input segments are intact and there is a flow of cytoplasm between them, there is no flow of cytoplasm via  $z$ . If one of the input tubes is mechanically stimulated, the flow through this tube stops, and the flow is diverted via the output tube  $z$ . If both input tubes are stimulated, there is no flow via input or output tubes. Thus, we implement XOR gate (Adamatzky and Schubert 2014). A mechanically stimulated fragment restores its flow of cytoplasm in 1 min: the gate is reusable.

By adding one more output (bypass) tube to XOR gate, we produce a gate with two inputs and two outputs:  $z$  and  $p$ .



The output  $z$  represents XOR. The tube  $p$  represents NOR because a cytoplasmic flow is directed via  $p$  if both tubes  $x$  and  $y$  are blocked. More complicated gates and memory devices can be found in Adamatzky and Schubert (2014).

### Intracellular Collision-Based Computing

The paradigm of a collision-based computing originates from the computational universality of the Game of Life, Fredkin-Toffoli conservative logic, and the billiard ball model with its cellular-automaton implementation (Adamatzky 2002). A collision-based computer employs mobile localizations, e.g., gliders in Conway's Game of Life cellular automata, to represent quanta of information in active nonlinear media. Information values, e.g., truth values of logical variables, are given by either the absence or presence of the localizations or by other parameters such as direction or velocity. The localizations travel in space and collide with each other. The results of the collisions are interpreted as computation. *Physarum* ballistic gates (section “[Ballistic Logical Gates](#)”) are also based on collisions, or interactions, between active growing zones of *Physarum*. However, signals per se, represented by growing body of *Physarum*, are not localized. Intracellular vesicles – up 100 nm droplets of liquid encapsulated by a lipid bilayer membrane – could be convenient representations of localized signals.

In Mayne and Adamatzky (2015b), we outlined pathways toward collision-based computing with vesicles inside the *Physarum* cell. The vesicles travel along actin and tubulin network and collide with each other. The colliding vesicles may reflect, fuse, or annihilate. The vesicles reflect in over half of the collisions observed. The vesicles fuse in one of seven collisions. The

vesicles annihilate and unload their cargo in one of ten collisions. The vesicles become paired and travel as a single object in one of ten collisions. Based on the experimental observations, we derive soft spheres collision (Margolus 2002) gates and also gates NOT and FAN-OUT.

### Kolmogorov-Uspensky Machine

In 1950s, Kolmogorov outlined a concept of an algorithmic process, an abstract machine, defined on a dynamically changing graph (Kolmogorov 1953). The structure later became known as Kolmogorov-Uspensky machine (Kolmogorov and Uspenskii 1958). The machine is a computational process on a finite undirected connected graph with distinctly labeled nodes (Gurevich 1988). A computational process travels on the graph, activates nodes, and removes or adds edges. A program for the machine specifies how to replace the neighborhood of an active node with a new neighborhood, depending on the labels of edges connected to the active node and the labels of the nodes in proximity to the active node (Blass and Gurevich 2003). The Kolmogorov-Uspensky machine is more flexible than a Turing machine because it recognizes in real time some predicates not recognizable in real time by the Turing machine (Yu Grigor'ev 1980); it is stronger than any model of computation that requires  $\Omega(n)$  time to access its memory (Cloteaux and Ranjan 2006; Shvachko 1991). “Turing machines formalize computation as it is performed by a human. Kolmogorov-Uspensky machines formalize computation as it performed by a physical process” (Blass and Gurevich 2003).

We implement the Kolmogorov-Uspensky machine in the *Physarum* as follows (Adamatzky 2007). Stationary nodes are represented by sources of nutrients. Dynamic nodes are assigned to branching site of the protoplasmic tubes. The stationary nodes are labeled by food colorings because *Physarum* exhibits a hierarchy of preferences to different colorings. An active zone in the storage graph is selected by inoculating the *Physarum* on one of the stationary nodes. An edge of the Kolmogorov-Uspensky machine is a protoplasmic tube connecting the nodes.

Program and data are represented by the spatial configuration of stationary nodes. Results of the computation over a stationary data node are represented by the configuration of dynamic nodes and edges. The initial state of a *Physarum* machine (*Physarum* implementation of Kolmogorov-Uspensky machine) includes part of an input string (the part which represents the position of *Physarum* relative to stationary nodes), an empty output string, the current instruction in the program, and the storage structure consisting of one isolated node. The physical graph structure developed by *Physarum* is the result of its computation. The *Physarum* machine halts when all data nodes are utilized. At every step of the computation, there is an active node and an active zone (nodes neighboring the active node). The active zone has a limited complexity: all elements of the zone are connected by some chain of edges to the initial node. The size of the active zone may vary depending on the computational task. In the *Physarum* machine, an active node is a trigger of contraction/excitation waves, which spread all over the plasmodium tree and cause pseudopodia to propagate, the shape to change, and protoplasmic veins to annihilate. The active zone comprises stationary and/or dynamic nodes connected to an active node with tubes of protoplasm. Instructions of the *Physarum* machine are INPUT, OUTPUT, GO, HALT, ADD NODE, REMOVE NODE, ADD EDGE, REMOVE EDGE, and IF.

The INPUT is done via distribution of sources of nutrients. The OUTPUT is recorded optically. The SET instruction causes pointers to redirect. It is realized by placing a fresh nutrient source in the experimental container. When a new node is created, all pointers from the old node point to the new node (Adamatzky 2007).

## **Mathematical Models and Algorithms Inspired by *Physarum* but Never Implemented in Laboratory Experiments**

1. A non-quantum implementation of Shor's factorization algorithm (Blakey 2014) is inspired by *Physarum* ability to retain the

time periods of stimulation, the anticipatory behavior (Saigusa et al. 2008).

2. Single-electron circuit solving maze problem (Shinde and Oya 2014) is based on a cellular automaton imitation of *Physarum*.
3. Particle-based model of *Physarum* approximates moving average and low-pass filters in one-dimensional data sets and spatial computation of splines in two-dimensional data set (Jones and Adamatzky 2014b).
4. *Physarum* logic is developed by interpreting basic features of the *Physarum* foraging behavior in terms of process calculi and spatial logic without modal operators (Schumann and Adamatzky 2011).
5. Soft amoeboid robots (Piovanelli et al. 2012) – models of coupled-oscillator-based robots (Umedachi et al. 2013) – are based on general principles of *Physarum* behavior, especially coordination of distant parts of its cell.
6. *Physarum* concurrent games are proposed in Schumann et al. (2014). In these games, rules can change, players update their strategies and actions, and resistance points are reduced to payoffs. At the time these games were proposed, we were unaware about the lethal reaction followed a fusion between *Physarum* of two different strains (Carlile 1972): a degree of lethality depends on the position and size of invasion between strains, which supports the ideas developed in Schumann et al. (2014).
7. *Physarum* is interfaced with a field-programmable array in Mayne et al. (2015). The hybrid system performs predefined arithmetic operations derived by digital recognition of membrane potential oscillations.
8. A *Physarum*-inspired algorithm for solution of the Steiner tree problem is proposed in Liang et al. (2015b) and applied to optimize the minimal exposure problem and worst-case coverage in wireless sensor networks (Liang et al. 2015b; Tsompanas et al. 2015).
9. An abstract implementation of reversible logical gates with *Physarum* is proposed in Schumann (2015).
10. Mechanisms of *Physarum* foraging behavior are employed in robotics algorithm for

- simultaneous localization and mapping (Kalogeiton et al. 2014).
11. A range of algorithms for network optimization is derived from a model of *Physarum* shortest path formation (Bonifaci et al. 2012; Tero et al. 2006). Most of these algorithms are based on a feedback between traffic throughout a tube and the tube's capacity. They include dynamical shortest path algorithms (Zhang et al. 2014), optimal communication paths in wireless sensor networks (Dourvas et al. 2015; Zhang et al. 2015a), supply chains design (Zhang et al. 2015a), shortest path tree problem (Zhang et al. 2015b), design of fault tolerant graphs (Becker et al. 2015), and multi-cast routing (Liang et al. 2015a). Behavior of *Physarum* solver (Tero et al. 2006) was also applied to derive a shortest path on Riemann surface (Miyaji and Ohnishi 2008).
  12. *Physarum*-inspired algorithm for learning Bayesian network structure from data is designed in Schön et al. (2014).
  13. Attraction-based two-input two-output gate realizing AND and OR and three-input two-output gate realizing conjunction of three inputs and negation of one input with disjunction of two other inputs are constructed in particle-based model of *Physarum* (Jones and Adamatzky 2010); the gates are cascaded into a one bit adder.
  14. A nano-device aimed to solve Boolean satisfiability problem and inspired by optimization of protoplasmic networks by *Physarum* is designed in Aono et al. (2012, 2015). The device works on fluctuations generated from thermal energy in nanowires, electrical Brownian ratchets.
  15. Solution of a the “exploration versus exploitation” dilemma by *Physarum* making a choice between colonizing nutrients and escaping illuminated areas is used in a tug of war model (Kim et al. 2010): a parallel search of a space by collectives of locally correlated agents and decision-making in situations of uncertainty. The ideas are developed further in a design of experimental device where a

single photon solves the multiarmed bandit problem (Naruse et al. 2015).

## Future Directions

Slime mold *Physarum polycephalum* is proved to be a universal, in formal and informal senses, sensing and computing substrate, because it is relatively easy to culture and experiment with. The excitement of using the slime mold should not overshadow drawbacks of the *Physarum* computers. Speed is the first deficiency. *Physarum* is a very slow creature. Sometimes it takes *Physarum* several days to solve a computational problem. We cannot speed up the slime mold, but we can locate application domains for *Physarum* computers where speed is not critical. Non-repeatability is the second deficiency. *Physarum* always behaves differently. No two experiments are the same. Results of a computation performed by *Physarum* are only valid when at least dozen of experiments are done because a single trial might mean nothing. *Physarum* solves problems statistically. There may be experimental way to “normalize” the slime mold behavior; these ways might be developed in the future. Will we ever see the *Physarum* in commercial computing or sensing devices? Not tomorrow. In no way *Physarum* can win over the silicon technology which has been optimized nonstop for decades and decades. But a success depends on many factors, not just technological ones. Success is in finding a vacant niche and flourishing there. More likely applications of *Physarum* computers will be in disposable hybrid processing devices used for sensing and decision-making in environments and situations where speed does not matter, but being energy efficient, adaptable, and self-healing is important.

## Bibliography

- Adamatzky A (1991) Neural algorithm for constructing minimal spanning tree. *Neural Netw World* 6:335–339  
 Adamatzky A (2002) Collision-based computing. Springer, London

- Adamatzky A (2007) Physarum machine: implementation of a Kolmogorov-Uspensky machine on a biological substrate. *Parallel Process Lett* 17(04):455–467
- Adamatzky A (2008) Growing spanning trees in plasmodium machines. *Kybernetes* 37(2):258–264
- Adamatzky A (2009) Developing proximity graphs by Physarum polycephalum: does the plasmodium follow the Toussaint hierarchy? *Parallel Process Lett* 19(01):105–127
- Adamatzky A (2010a) *Physarum machines: computers from slime mould*. World Scientific Publishing, London
- Adamatzky A (2010b) Slime mould logical gates: exploring ballistic approach. arXiv preprint arXiv:1005.2301
- Adamatzky A (2012a) Bioevaluation of world transport networks. World Scientific Publishing, London
- Adamatzky A (2012b) Slime mold solves maze in one pass, assisted by gradient of chemo-attractants. *Nano-Bioscience IEEE Trans* 11(2):131–134
- Adamatzky A (2012c) Slime mould computes planar shapes. *Int J Bio-Inspired Comput* 4(3):149–154
- Adamatzky A (ed) (2016) *Advances in Physarum machines: sensing and computing with slime mould*. Springer, Heidelberg
- Adamatzky A, Kayem AVDM (2013) Biological evaluation of trans-African high-ways. *European Phys J Spec Top* 215(1):49–59
- Adamatzky A, Martinez GJ (2013) Bio-imitation of Mexican migration routes to the USA with slime mould on 3D terrains. *J Bionic Eng* 10(2):242–250
- Adamatzky A, Prokopenko M (2012) Slime mould evaluation of Australian motor-ways. *Int J Parallel Emergent Distrib Syst* 27(4):275–295
- Adamatzky A, Schubert T (2012) Schlauschleimer in Reichsautobahnen: slime mould imitates motorway network in Germany. *Kybernetes* 41(7/8):1050–1071
- Adamatzky A, Schubert T (2014) Slime mold microfluidic logical gates. *Mater Today* 17(2):86–91
- Adamatzky A, De Lacy Costello B, Asai T (2005) Reaction-diffusion computers. Elsevier, Amsterdam
- Adamatzky A, De Baets B, Van Dessel W (2012) Slime mould imitation of Belgian transport networks: redundancy, bio-essential motorways, and dissolution. *Int J Unconv Comput* 8(3):235–261
- Adamatzky A, Akl S, Alonso-Sanz R, Van Dessel W, Ibrahim Z, Ilachinski A, Jones J, Kayem AVDM, Martinez GJ, De Oliveira P et al (2013a) Are motorways rational from slime mould's point of view? *Int J Parallel Emergent Distrib Syst* 28(3):230–248
- Adamatzky A, Lees M, Sloot P (2013b) Bio-development of motorway network in the Netherlands: a slime mould approach. *Adv Complex Syst* 16(02n03):1250034
- Adamatzky A, Armstrong R, De Lacy Costello B, Deng Y, Jones J, Mayne R, Schubert T, Ch Sirakoulis G, Zhang X (2014) Slime mould analogue models of space exploration and planet colonisation. *J Br Interplanet Soc* 67:290–304
- Aono M, Kim S-J, Zhu L, Naruse M, Ohtsu M, Hori H, Hara M (2012). Amoeba-inspired sat solver. In: Proc. NOLTA. p 586–589
- Aono M, Kasai S, Kim SJ, Wakabayashi M, Miwa H, Naruse M (2015) Amoeba-inspired nanoarchitectonic computing implemented using electrical brownian ratchets. *Nanotechnology* 26(23):234001
- Ascher R (1961) Experimental archeology. *Am Anthropol* 63(4):793–816
- Becker M, Kromker M, Szczerbicka H (2015) Evaluating heuristic optimization, bio-inspired and graph-theoretic algorithms for the generation of fault-tolerant graphs with minimal costs. In: *Information science and applications*. Springer, Berlin, pp 1033–1041
- Blakey E (2014) Towards non-quantum implementations of shor's factorization algorithm. *Int J Unconv Comput* 10:339–352
- Blass A, Gurevich Y (2003) Algorithms: a quest for absolute definitions. *Bull EATCS* 81:195–225
- Bonifaci V, Mehlhorn K, Varma G (2012) Physarum can compute shortest paths. *J Theor Biol* 309:121–133
- Carlile MJ (1972) The lethal interaction following plasmodial fusion between two strains of the myxomycete physarum polycephalum. *J Gen Microbiol* 71(3):581–590
- Clotheaux B, Ranjan D (2006) Some separation results between classes of pointer algorithms. *DCFS* 6:232–240
- Coles J (1979) *Experimental archaeology*. Academic Press, London
- Delaunay B (1934) Sur la sphère vide. *Izv Akad Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk* 7(793–800):1–2
- Dietrich MR (2015) Explaining the pulse of protoplasm: the search for molecular mechanisms of protoplasmic streaming. *J Integr Plant Biol* 57(1):14–22
- Dourvas N, Tsompanas M-A, Sirakoulis GC, Tsalides P (2015) Hardware acceleration of cellular automata Physarum polycephalum model. *Parallel Process Lett* 25(01):1540006
- Durham AC, Ridgway EB (1976) Control of chemotaxis in physarum polycephalum. *J Cell Biol* 69(1):218–223
- Edelsbrunner H, Kirkpatrick DG, Seidel R (1983) On the shape of a set of points in the plane. *Inf Theory IEEE Trans* 29(4):551–559
- Evangelidis V, Tsompanas M-A, Sirakoulis GC, Adamatzky A (2015) Slime mould imitates development of Roman roads in the Balkans. *J Archaeol Sci Rep* 2:264–281
- Gabriel KR, Sokal RR (1969) A new statistical approach to geographic variation analysis. *Syst Biol* 18(3):259–278
- Grebecki A, Cieślawska M (1978) Plasmodium of physarum polycephalum as a synchronous contractile system. *Cytobiologie* 17(2):335–342
- Gurevich Y (1988) Kolmogorov machines and related issues. *Bull EATCS* 35:71–82
- Hinz AM (1989) The tower of Hanoi. *Enseign Math* 35(2):289–321
- Hinz AM (1992) Shortest paths between regular states of the Tower of Hanoi. *Inf Sci* 63(1):173–181
- Ingersoll D, Yellen JE, Macdonald W (1977) *Experimental archaeology*. Columbia University Press, New York

- Jarvis RA (1973) On the identification of the convex hull of a finite set of points in the plane. *Inf Process Lett* 2(1):18–21
- Jones J, Adamatzky A (2010) Towards Physarum binary adders. *Biosystems* 101(1):51–58
- Jones J, Adamatzky A (2014a) Computation of the traveling salesman problem by a shrinking blob. *Nat Comput* 13(1):1–16
- Jones J, Adamatzky A (2014b) Material approximation of data smoothing and spline curves inspired by slime mould. *Bioinspir Biomim* 9(3):036016
- Kalogeiton VS, Papadopoulos DP, Sirakoulis GC (2014) Hey physarum! Can you perform slam? *Int J Unconv Comput* 10(4):271–293
- Kalogeiton VS, Papadopoulos DP, Georgilas IP, Sirakoulis GC, Adamatzky AI (2015a) Biomimicry of crowd evacuation with a slime mould cellular automaton model. In: Computational intelligence, medicine and biology. Springer, Berlin, pp 123–151
- Kalogeiton VS, Papadopoulos DP, Georgilas IP, Ch Sirakoulis G, Adamatzky AI (2015b) Cellular automaton model of crowd evacuation inspired by slime mould. *Int J Gen Syst* 44(3):354–391
- Kauffman S, Wille JJ (1975) The mitotic oscillator in *Physarum polycephalum*. *J Theor Biol* 55(1):47–93
- Kim S-J, Aono M, Hara M (2010) Tug-of-war model for the two-bandit problem: nonlocally-correlated parallel exploration via resource conservation. *Biosystems* 101(1):29–36
- Kishimoto U (1958) Rhythmicity in the protoplasmic streaming of a slime mould, *physarum polycephalum*. i. A statistical analysis of the electrical potential rhythm. *J Gen Physiol* 41(6):1205–1222
- Kolmogorov AN (1953) On the concept of algorithm. *Uspekhi Mat Nauk* 8(4):175–176
- Kolmogorov AN, Uspenskii VA (1958) On the definition of an algorithm. *Uspekhi Matematicheskikh Nauk* 13(4):3–28
- Liang M, Gao C, Liu Y, Tao L, Zhang Z (2015a) A new physarum network based genetic algorithm for bandwidth-delay constrained least-cost multicast routing. In: Advances in swarm and computational intelligence. Springer, Berlin, pp 273–280
- Liang L, Song Y, Zhang H, Ma H, Vasilakos AV (2015b) Physarum optimization: a biology-inspired algorithm for the steiner tree problem in networks. *Comput IEEE Trans* 64(3):819–832
- MacGregor JN, Ormerod T (1996) Human performance on the traveling salesman problem. *Percept Psychophys* 58(4):527–539
- Margolus N (2002) Universal cellular automata based on the collisions of soft spheres. In: Collision-based computing. Springer, London, pp 107–134
- Mayne R, Adamatzky A (2015a) Slime mould foraging behaviour as optically coupled logical operations. *Int J Gen Syst* 44(3):305–313
- Mayne R, Adamatzky A (2015b) On the computing potential of intracellular vesicles. *PLoS One* 10(10):e0139617
- Mayne R, Tsompanas M-A, Sirakoulis GC, Adamatzky A (2015) Towards a slime mould-FPGA interface. *Biomed Eng Lett* 5(1):51–57
- Miyaji T, Ohnishi I (2008) Physarum can solve the shortest path problem on Riemannian surface mathematically rigorously. *Int J Pure Appl Math* 47(3):353–369
- Nakagaki T, Yamada H, Toth A (2001) Path finding by tube morphogenesis in an amoeboid organism. *Biophys Chem* 92(1):47–52
- Naruse M, Berthel M, Drezet A, Huant S, Aono M, Hori H, Kim S-J (2015) Single-photon decision maker. *Sci Rep* 5:13253
- Nešetřil J, Milková E, Nešetřilová H (2001) Otakar Boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Math* 233(1):3–36
- Piovanelli M, Fujie T, Mazzolai B, Beccai L (2012) A bio-inspired approach towards the development of soft amoeboid microrobots. In: Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS & EMBS International Conference on. IEEE, p 612–616
- Preparata FP, Shamos MI (1985) Computational geometry: an introduction. Springer, New York
- Rakoczy L (1963) Application of crossed light and humidity gradients for the investigation of slime-molds. *Acta Soc Bot Pol* 32(2):393–403
- Reid CR, Beekman M (2013) Solving the towers of Hanoi—how an amoeboid organism efficiently constructs transport networks. *J Exp Biol* 216(9):1546–1551
- Ridgway EB, Durham AC (1976) Oscillations of calcium ion concentrations in *Physarum polycephalum*. *J Cell Biol* 69(1):223–226
- Romik D (2006) Shortest paths in the Tower of Hanoi graph and finite automata. *SIAM J Discret Math* 20(3):610–622
- Saigusa T, Tero A, Nakagaki T, Kuramoto Y (2008) Amoebae anticipate periodic events. *Phys Rev Lett* 100(1):018101
- Schön T, Stetter M, Tomé AM, Puntonet CG, Lang EW (2014) Physarum learner: a bio-inspired way of learning structure from data. *Expert Syst Appl* 41(11):5353–5370
- Schumann A (2017) Conventional and unconventional reversible logic gates on *Physarum polycephalum*. *International Journal of Parallel, Emergent and Distributed Systems* 32(2):218–231
- Schumann A, Adamatzky A (2011) Physarum spatial logic. *New Math Nat Comput* 7(03):483–498
- Schumann A, Pancerz K, Adamatzky A, Grube M (2014) Bio-inspired game theory: the case of *physarum polycephalum*. In: Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, ICST, p 9–16
- Shinde Y, Oya T (2014) Design of single-electron slime-mold circuit and its application to solving optimal path planning problem. *Nonlinear Theory Appl IEICE* 5(s):80–88
- Shirakawa T, Adamatzky A, Gunji Y-P, Miyake Y (2009) On simultaneous construction of Voronoi diagram and Delaunay triangulation by *Physarum polycephalum*. *Int J Bifurcation Chaos* 19(09):3109–3117

- Shvachko KV (1991) Different modifications of pointer machines and their computational power. In: Mathematical Foundations of Computer Science 1991. Springer, Berlin, pp 426–435
- Stephenson SL, Stempel H, Hall I (1994) Myxomycetes: a handbook of slime molds. Timber Press Portland, Oregon
- Teplov VA, Romanovsky YM, Latushkin OA (1991) A continuum model of contraction waves and protoplasm streaming in strands of *physarum plasmodium*. Biosystems 24(4):269–289
- Tero A, Kobayashi R, Nakagaki T (2006) Physarum solver: a biologically inspired method of road-network navigation. Phys A: Stat Mech Appl 363(1):115–119
- Tsompanas M-AI, Mayne R, Sirakoulis GC, Adamatzky AI (2015) A cellular automata bioinspired algorithm designing data trees in wireless sensor networks. Int J Distrib Sens Netw 501:471045
- Tsuda S, Aono M, Gunji Y-P (2004) Robust and emergent physarum logical-computing. Biosystems 73(1):45–55
- Ueda T, Muratsugu M, Kurihara K, Kobatake Y (1976) Chemotaxis in Physarum polycephalum: effects of chemicals on isometric tension of the plasmodial strand in relation to chemotactic movement. Exp Cell Res 100(2):337–344
- Umedachi T, Idei R, Ito K, Ishiguro A (2013) A fluid-filled soft robot that exhibits spontaneous switching among versatile spatiotemporal oscillatory patterns inspired by the true slime mold. Artif Life 19(1):67–78
- Whiting JGH, de Lacy Costello BPJ, Adamatzky A (2014) Slime mould logic gates based on frequency changes of electrical potential oscillation. Biosystems 124:21–25
- Wohlfarth-Bottermann KE (1979) Oscillatory contraction activity in *physarum*. J Exp Biol 81(1):15–32
- Grigoriev YD (1980) Kolmogoroff algorithms are stronger than turing machines. J Math Sci 14(5):1445–1450
- Zhang X, Wang Q, Adamatzky A, Chan FT, Mahadevan S, Deng Y (2014) An improved physarum polycephalum algorithm for the shortest path problem. *The Scientific World Journal*, 2014
- Zhang, Xiaoge, Sankaran Mahadevan, and Yong Deng. Physarum-inspired applications in graph-optimization problems. Parallel Processing Letters 25.01 (2015): 1540005
- Zhu L, Aono M, Kim S-J, Hara M (2013) Amoeba-based computing for traveling salesman problem: long-term correlations between spatially separated individual cells of *physarum polycephalum*. Biosystems 112(1):1–10



## Evolution in Materio

Simon Harding<sup>1</sup> and Julian F. Miller<sup>2</sup>

<sup>1</sup>Department of Computer Science, Memorial University, St. John's, Canada

<sup>2</sup>Department of Electronics, University of York, Heslington, UK

### Article Outline

Glossary

Definition of the Subject

Introduction

Evolutionary Algorithms

Evolution in Materio: Historical Background

Evolution in Materio: Defining Suitable Materials

Evolution in Materio is Verified with Liquid Crystal

Evolution in Materio Using Liquid Crystal:  
Implementational Details

The Computational Power of Materials

Future Directions

Bibliography

### Glossary

**Evolution in materio** The method of applying computer-controlled evolution to manipulate or configure a physical system.

**Evolutionary algorithm** A computer algorithm loosely inspired by Darwinian evolution.

**Generate-and-test** The process of generating a potential solution to a computational problem and testing it to see how good a solution it is. The idea behind it is that no human ingenuity is employed to make good solutions more likely.

**Genotype** A string of information that encodes a potential solution instance of a problem and allows its suitability to be assessed.

**Liquid crystal** Substances that have properties between those of a liquid and a crystal.

### Definition of the Subject

Evolution in materio refers to the use of computers running search algorithms, called evolutionary algorithms, to find the values of variables that should be applied to material systems so that they carry out useful computation. Examples of such variables might be the location and magnitude of voltages that need to be applied to a particular physical system. Evolution in materio is a methodology for programming materials that utilizes physical effects that the human programmer need not be aware of. It is a general methodology for obtaining analog computation that is specific to the desired problem domain. Although a form of this methodology was hinted at in the work of Gordon Pask in the 1950s, it was not convincingly demonstrated until 1996 by Adrian Thompson, who showed that physical properties of a digital chip could be exploited by computer-controlled evolution. This entry describes the first demonstration that such a method can be used to obtain specific analog computation in a non-silicon-based physical material (liquid crystal). The work is important for a number of reasons. Firstly, it proposes a general method for building analog computational devices. Secondly it explains how previously unknown physical effects may be utilized to carry out computations. Thirdly, it presents a method that can be used to *discover* useful physical effects that can form the basis of future computational devices.

### Introduction

#### Physical Computation

Classical computation is founded on a mathematical model of computation based on an abstract (but physically inspired) machine called a Turing machine (Turing 1936). A Turing machine can write or erase symbols on a possibly infinite one-

dimensional tape. Its actions are determined by a table of instructions that determine what the machine will write on the tape (by moving one square left or right) given its state (stored in a state register) and the symbol on the tape. Turing showed that the calculations that could be performed on such a machine accord with the notion of computation in mathematics. The Turing machine is an abstraction (partly because it uses a possibly infinite tape), and to this day it is still not understood what limitations or extensions to the computational power of Turing's model might be possible using real physical processes. Von Neumann and others at the Institute for Advanced Study at Princeton devised a design for a computer based on the ideas of Turing that has formed the foundation of modern computers. Modern computers are digital in operation. Although they are made of physical devices (i.e., transistors), computations are made on the basis of whether a voltage is above or below some threshold. Prior to the invention of digital computers, there have been a variety of analog computing machines. Some of these were purely mechanical (e.g., an abacus, a slide rule, Charles Babbage's difference engine, Vannevar Bush's differential analyzer), but later computing machines were built using operational amplifiers (Bissell 2004).

There are many aspects of computation that were deliberately ignored by Turing in his model of computation. For instance, speed, programmability, parallelism, openness, and adaptivity are not considered. The speed at which an operation can be performed is clearly an important issue since it would be of little use to have a machine that can calculate any computable function but takes an arbitrarily large amount of time to do so. Programmability is another issue that is of great importance. Writing programs directly in the form of instruction tables that could be used with a device based on a Turing is extremely tedious. This is why many high-level computer languages have been devised. The general issue of how to subdivide a computer program into a number of parallel executing processes so that the intended computation is carried out as quickly as possible is still unsolved. Openness refers to systems that can interact with an external

environment during their operation. Openness is exhibited strongly in biological systems where new resources can be added or removed either by an external agency or by the actions taken by the system itself. Adaptivity refers to the ability of systems to change their characteristics in response to an environment.

In addition to these aspects, the extent to which the underlying physics affects both the abstract notion of computation and its tractability has been brought to prominence through the discovery of quantum computation, where Deutsch pointed out that Turing machines implicitly use assumptions based on physics (Deutsch 1985). He also showed that through "quantum parallelism," certain computations could be performed much more quickly than on classical computers. Other forms of physical computation that have recently been explored are reaction-diffusion systems (Adamatzky et al. 2005), DNA computing (Adleman 1994; Amos 2005), and synthetic biology (Weiss et al. 2003).

In the UK a number of grand challenges in computing research have been proposed (UK Computing Research Committee 2005), in particular *Journeys in Non-Classical Computation* (Stepney et al. 2005; Stepney et al. 2006) which seeks to explore, unify, and generalize many diverse nonclassical computational paradigms to produce a mature science of computation.

Toffoli argued that *Nothing Makes Sense in Computing Except in the Light of Evolution* (Toffoli 2005). He argues firstly that a necessary but not sufficient condition for a computation to have taken place is when a novel function is produced from a fixed and finite repertoire of components (i.e., logic gates, protein molecules). He suggests that a sufficient condition requires *intention*. That is to say, we cannot argue that computation has taken place unless a system has arisen for a higher purpose (this is why he insists on intention as being a prerequisite for computation). Otherwise, almost everything is carrying out some form of computation (which is not a helpful point of view). Thus a Turing machine does not carry out computations unless it has been programmed to do so, and since natural evolution constructs organisms that have an increased chance of survival (the higher

“purpose”), we can regard them as carrying out computations. It is in this sense that Toffoli points to the fundamental role of evolution in the definition of a computation as it has provided animals with the ability to have intention.

This brings us to one of the fundamental questions in computation. How can we program a physical system to perform a particular computation? The dominant method used to answer this question has been to construct logic gates and from these build a von Neumann machine (i.e., a digital computer). The mechanism that has been used to devise a computer program to carry out a particular computation is the familiar top-down design process, where ultimately the computation is represented using Boolean operations. According to Conrad this process leads us to pay *The Price of Programmability* (Conrad 1988), whereby in conventional programming and design we proceed by excluding many of the processes that may lead to us solving the problem at hand. Natural evolution does not do this. It is noteworthy that natural evolution has constructed systems of extraordinary sophistication, complexity, and computational power. We argue that it is not possible to construct computational systems of such power using a conventional methodology and that complex software systems that directly utilize physical effects will require some form of search process akin to natural evolution together with a way of manipulating the properties of materials. We suggest that some form of evolution ought to be an appropriate methodology for arriving at *physical* systems that compute. In this chapter we discuss work that has adopted this methodology. We call it evolution in materio.

## Evolutionary Algorithms

Firstly we propose that to overcome the limitations of a top-down design process, we should use a more unconstrained design technique that is more akin to a process of generate-and-test. However, a guided search method is also required that spends more time in areas of the search space that confer favorable traits for computation. One such approach is the use of evolutionary algorithms.

These algorithms are inspired by the Darwinian concepts of survival of the fittest and the genetic inheritance of information. Using a computer, a population of randomly generated solutions is systematically tested, selected, and modified until a solution has been found (Goldberg 1989; Holland 1992; Mitchell 1996).

As in nature, a genetic algorithm optimizes a population of individuals by selecting the ones that are best suited to solving a problem and allowing their genetic makeup to propagate into future generations. It is typically guided only by the evolutionary process and often contains very limited domain-specific knowledge. Although these algorithms are bio-inspired, it is important that any analogies drawn with nature are considered only as analogies.

Their lack of specialization for a problem makes genetic algorithms ideal search techniques where little is known about a problem. As long as a suitable representation is chosen along with a fitness function that allows for ease of movement around a search space, a GA can search vast problem spaces rapidly. Another feature of their behavior is that provided that the genetic representation chosen is sufficiently expressive, the algorithm can explore potential solutions that are unconventional. A human designer normally has a set of predefined rules and strategies that they adopt to solve a problem. These preconceptions may prevent trying a new method and may prevent the designer using a better solution. A genetic algorithm does not necessarily require such domain knowledge. Evolutionary algorithms have been shown to be competitive or surpass human-designed solutions in a number of different areas. The largest conference on evolutionary computation called GECCO has an annual session on evolutionary approaches that have produced human-competitive scientific and technological results. Moreover the increase in computational power of computers makes such results increasingly more likely.

Many different versions of genetic algorithms exist. Variations in representations and genetic operators change the performance characteristics of the algorithm, and depending on the problem, people employ a variety of modifications of the basic algorithm. However, all the algorithms follow a similar basic set of steps.

Firstly the numbers or physical variables that are required to define a potential solution have to be identified and encoded into a data representation that can be manipulated inside a computer program. This is referred to as the encoding step. The representation chosen is of crucial importance as it is possible to inadvertently choose overly constrained representations which limit the portions of the space of potential solutions that will be considered by the evolutionary algorithm. Generally the encoded information is referred to as a genotype, and genotypes are sometimes divided into a number of separate strings called chromosomes. Each entry in the chromosome string is an allele, and one or more of these make up a gene.

The second step is to create inside the computer a number of independently generated genotypes whose alleles have been chosen with uniform probability from the allowed set of values. This collection of genotypes is called a population.

In its most basic form, an individual genotype is a single chromosome made of 1s and 0s. However, it is also common to use integer and floating-point numbers if they are more appropriate for the task at hand. Combinations of different representations can also be used within the same chromosome, and that is the approach used in the work described in this article. Whatever representation is used, it should be able to adequately describe the individual and provide a mechanism where its characteristics can be transferred to future generations without loss of information.

Each of these individuals is then decoded into its phenotype, the outward, physical manifestation of the individual and tested to see how well the candidate solution solves the problem at hand. This is usually returned as a number that is referred to as the *fitness* of the genotype. Typically it is this phase in a genetic algorithm that is the most time consuming.

The next stage is to select what genetic information will proceed to the next generation. In nature the fitness function and selection are essentially the same – individuals that are better suited to the environment survive to reproduce and pass on their genes. In the genetic algorithm a procedure is applied to determine what information gets to proceed.

Genetic algorithms are often generational – where all the old population is removed before moving to the next generation; in nature this process is much less algorithmic. However, to increase the continuity of information between generations, some versions of the algorithm use elitism, where the fittest individuals are always selected for promotion to the next generation. This ensures that good solutions are not lost from the population, but it may have the side effect of causing the genetic information in the population to converge too quickly so that the search stagnates on a suboptimal solution.

To generate the next population, a procedure analogous to sexual reproduction occurs. For example, two individuals will be selected, and they will then have their genetic information combined together to produce the genotype for the offspring. This process is called recombination or crossover. The genotype is split into sections at randomly selected points called crossover points. A “simple” GA has only one of these points; however, it is possible to perform this operation at multiple points.

Sections of the two chromosomes are then put together to form a new individual. This individual shares some of the characteristics of both parents. There are many different ways to choose which members of the population to breed with each other, the aim in general is to try and ensure that fit individuals get to reproduce with other fit individuals. Individuals can be selected with a probability proportional to their relative fitness or selected through some form of tournament, which may choose two or more chromosomes at random from the population and select the fittest.

In natural recombination, errors occur when the DNA is split and combined together. Also, errors in the DNA of a cell can occur at any time under the influence of a mutagen, such as radiation, a virus or toxic chemical. The genetic algorithm also has mutations. A number of alleles are selected at random and modified in some way. For a binary GA, the bit may be flipped; in a real-numbered GA, a random value may be added to or subtracted from the previous allele.

Although GAs often have both mutation and crossover, it is possible to just use mutation.

A mutation-only approach has in some cases been demonstrated to work, and often crossover is seen as a macromutation operator – effectively changing large sections of a chromosome.

After the previous operations have been carried out, the new individuals in the population are then retested and their new fitness scores calculated. Eventually this process leads to an increase in the average fitness of the population, and so the population moves closer toward a solution. This cycle of test, select, and reproduce is continued until a solution is found (or some other termination condition is reached), at which point the algorithm stops. The performance of a genetic algorithm is normally measured in terms of the number of evaluations required to find a solution of a given quality.

## Evolution in Materio: Historical Background

It is arguable that “evolution in materio” began in 1958 in the work of Gordon Pask who worked on experiments to grow neural structures using electrochemical assemblages (Pask 1958; Pask 1959; Cariani 1993; Pickering 2002). Gordon Pask’s goal was to create a device sensitive to either sound or magnetic fields that could perform some form of signal processing – a kind of ear. He realized he needed a system that was rich in structural possibilities and chose to use a metal solution. Using electric currents, wires can be made to self-assemble in an acidic aqueous metal-salt solution (e.g., ferrous sulfate). Changing the electric currents can alter the structure of these wires and their positions – the behavior of the system can be modified through external influence. Pask used an array of electrodes suspended in a dish containing the metal-salt solution and by applying current (either transiently or a slowly changing source) was able to build iron wires that responded differently to two different frequencies of sound – 50 and 100 Hz (Fig. 1).

Pask had developed a system whereby he could manually train the wire formation in such a way that no complete specification had to be given – a complete paradigm shift from previous

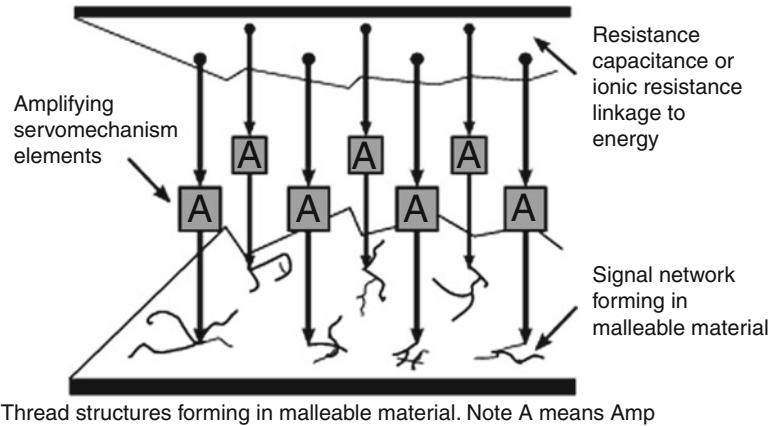
engineering techniques which would have dictated the position and behavior of every component in the system. His training technique relied on making changes to a set of resistors and updating the values with given probabilities – in effect a test-randomly modify-test cycle. We would today recognize this algorithm as some form of evolutionary, hill-climbing strategy – with the test stage as the fitness evaluation.

In 1996 Adrian Thompson started what we might call the modern era of evolution in materio. He was investigating whether it was possible to build working electronic circuits using unconstrained evolution (effectively, generate-and-test) using a reconfigurable electronic silicon chip called a field-programmable gate array (FPGA). Carrying out evolution by defining configurations of actual hardware components is known as *intrinsic* evolution. This is quite possible using FPGAs which are devices that have a two-dimensional array of logic functions that a configuration bit string defines and connects together. Thompson had set himself the task of evolving a digital circuit that could discriminate between an applied 1 and 10 kHz signal (Thompson et al. 1996; Thompson 1996). He found that computer-controlled evolution of the configuring bit strings could relatively easily solve this problem. However, when he analyzed the successful circuits, he found to his surprise that they worked by utilizing subtle electrical properties of the silicon. Despite painstaking analysis and simulation work, he was unable to explain how or what property was being utilized. This lack of knowledge of how the system works, of course, prevents humans from designing systems that are intended to exploit these subtle and complex physical characteristics. However, it does not prevent exploitation through artificial evolution. Since then a number of researchers have demonstrated the viability of intrinsic evolution in silicon devices (Thompson 1996; Bird and Layzell 2002; Layzell 1998; Linden and Altshuler 2001, 1999; Stoica et al. 2003, 2000).

The term *evolution in materio* was first coined by Miller and Downing (Miller and Downing 2002). They argued that the lesson that should be drawn from the work of (Thompson 1996) is

### Evolution in Materio,

**Fig. 1** Pask's experimental setup for growing dendritic wires in ferrous sulfate solution (Pask 1959)



that evolution may be used to exploit the properties of a wider range of materials than silicon.

In summary, evolution in materio can be described as:

Exploitation, using an unconstrained evolutionary algorithm, of the non-linear properties of a malleable or programmable material to perform a desired function by altering its physical or electrical configuration.

Evolution in materio is a subset of a research field known as evolvable hardware. It aims to exploit properties of physical systems with much fewer preconditions and constraints than is usual, and it deliberately tries to avoid paying Conrad's *The Price of Programmability*. However, to get access to physically rich systems, we may have to discard devices designed with human programming in mind. Such devices are often based on abstract idealizations of processes occurring in the physical world. For example, FPGAs are considered as digital, but they are fundamentally analog devices that have been constrained to behave in certain, human understandable ways. This means that intrinsically complex physical processes are carefully manipulated to represent extremely simple effects (e.g., a rapid switch from one voltage level to another). Unconstrained evolution, as demonstrated by Thompson, allows for the analog properties of such devices to be effectively utilized.

We would expect physically rich systems to exhibit nonlinear properties – they will be complex systems. This is because physical systems

generally have huge numbers of parts interacting in complex ways. Arguably, humans have difficulty working with complex systems, and the use of evolution enables us to potentially overcome these limitations when dealing with such systems.

When systems are abstracted, the relationship to the physical world becomes more distant. This is highly convenient for human designers who do not wish to understand, or work with, hidden or subtle properties of materials. Exploitation through evolution reduces the need for abstraction, as it appears evolution is capable of discovering and utilizing any physical effects it can find. The aim of this new methodology in computation is to evolve special purpose computational processors. By directly exploiting physical systems and processes, one should be able to build extremely fast and efficient computational devices. It is our view that computer-controlled evolution is a universal methodology for doing this. Of course, von Neumann machines (i.e., digital computers) are *individually* universal, and this is precisely what confers their great utility in modern technology; however, this universality comes at a price. They ignore the rich computational possibilities of materials and try to create operations that are close to a mathematical abstraction. Evolution in materio is a universal methodology for producing specific, highly tuned computational devices.

It is important not to underestimate the real practical difficulties associated with using an unconstrained design process. Firstly the evolved

behavior of the material may be extremely sensitive to the specific properties of the material sample, so each piece would require individual training. Thompson originally experienced this difficulty; however, in later work he showed that it was possible to evolve the configuration of FPGAs so that they produced reliable behavior in a variety of environmental conditions (Thompson 1998).

Secondly, the evolutionary algorithm may utilize physical aspects of any part of the training setup. Both of these difficulties have already been experienced (Thompson 1996; Layzell 1998). A third problem can be thought of as “the wiring problem.” The means to supply huge amounts of configuration data to a tiny sample. This problem is a very fundamental one. It suggests that if we wish to exploit the full physical richness of materials, we might have to allow the material to grow its own wires and be self-wiring. This has profound implications for intrinsic evolution as artificial hardware evolution requires complete reconfigurability; this implies that one would have to be able to “wipe clean” the evolved wiring and start again with a new artificial genotype. This might be possible by using nanoparticles that assemble into nanowires. These considerations bring us to an important issue in evolution in materio, namely, the problem of choosing suitable materials that can be exploited by computer-controlled evolution.

## Evolution in Materio: Defining Suitable Materials

The obvious characteristic required by a candidate material is the ability to reconfigure it in some way. Liquid crystal, clay, salt solutions, etc. can be readily configured either electrically or mechanically; their physical state can be adjusted, and readjusted, by applying a signal or force. In contrast (excluding its electrical properties) the physical properties of an FPGA would remain unchanged during configuration. It is also desirable to bulk configure the system. It would be infeasible to configure every molecule in the material, so the material should support the ability

to be reconfigured over large areas using a small amount of configuration.

The material needs to perform some form of transformation (or computation) on incident signals that we apply. To do this, the material will have to interfere with the incident signal and perform a modification to it. We will need to be able to observe this modification, in order to extract the result of the computation. To perform a nontrivial computation, the material should be capable of performing complex operations upon the signal. Such capabilities would be maximized if the system exhibited nonlinear behavior when interacting with input signals.

In summary, we can say that for a material to be useful to evolution in materio, it should have the following properties:

- Modify incident signals in observable ways.
- The components of a system (i.e., the molecules within a material) interact with each other locally such that nonlinear effects occur at either the local or global levels.
- It is possible to configure the state of the material locally.
- It is possible to observe the state of the material – either as a whole or in one or more locations.
- For practical reasons we can state that the material should be reconfigurable and that changes in state should be temporary or reversible.

Miller and Downing (Miller and Downing 2002) identified a number of physical systems that have some, if not all, of these desirable properties. They identified liquid crystal as the most promising in this regard as it is digitally writable and reconfigurable and works at a molecular level. Most interestingly, it is an example of mesoscopic organization. Some people have argued that it is within such systems that emergent, organized behavior can occur (Laughlin et al. 2000). Liquid crystals also exhibit the phenomenon of self-assembly. They form a class of substances that are being designed and developed in a field of chemistry called supramolecular chemistry (Lindoy and Atkinson 2000). This is a new and exciting branch

of chemistry that can be characterized as “the designed chemistry of the intermolecular bond.” Supramolecular chemicals are in a permanent process of being assembled and disassembled. It is interesting to consider that conceptually liquid crystals appear to sit on the “edge of chaos” (Langton 1991) in that they are fluids (chaotic) that can be ordered, under certain circumstances.

### Liquid Crystal

Liquid crystal (LC) is commonly defined as a substance that can exist in a mesomorphic state (Demus et al. 1998; Khoo 1995). Mesomorphic states have a degree of molecular order that lies between that of a solid crystal (long-range positional and orientational) and a liquid, gas, or amorphous solid (no long-range order). In LC there is long-range orientational order but no long-range positional order.

LC tends to be transparent in the visible and near infrared and quite absorptive in UV. There are three distinct types of LC: lyotropic, polymeric, and thermotropic. Lyotropic LC is obtained when an appropriate amount of material is dissolved in a solvent. Most commonly this is formed by water and amphiphilic molecules – molecules with a hydrophobic part (water insoluble) and a hydrophilic part (strongly interacting with water). Polymeric LC is basically a polymer version of the aromatic LC discussed. They are characterized by high viscosity and include vinyls and Kevlar. Thermotropic LC (TLC) is the most common form and is widely used. TLC exhibits various liquid crystalline phases as a function of temperature. They can be depicted as rodlike molecules and interact with each other in distinctive ordered structures. TLC exists in three main forms: nematic, cholesteric, and smectic. In nematic LC the molecules are positionally arranged randomly, but they all share a common alignment axis. Cholesteric LC (or chiral nematic) is like nematic; however, they have a chiral orientation. In smectic LC there is typically a layered positionally disordered structure. The three types A, B, and C are defined as follows. In type A the molecules are oriented in alignment with the natural physical axes (i.e., normal to the glass container); however, in type C the common

molecular axes of orientation are at an angle to the container. LC molecules typically are dipolar. Thus, the organization of the molecular dipoles gives another order of symmetry to the LC. Normally the dipoles would be randomly oriented. However in some forms the natural molecular dipoles are aligned with one another. This gives rise to ferroelectric and ferrielectric forms.

There is a vast range of different types of liquid crystal. LC of different types can be mixed. LC can be doped (as in dye-doped LC) to alter their light absorption characteristics. Dye-doped LC film has been made that is optically addressable and can undergo very large changes in refractive index (Khoo et al. 1998). There are polymer-dispersed liquid crystals, which can have tailored, electrically controlled light refractive properties. Another interesting form of LC being actively investigated is discotic LC. These have the form of disordered stacks (one-dimensional fluids) of disk-shaped molecules on a two-dimensional lattice. Although discotic LC is an electrical insulator, it can be made to conduct by doping with oxidants (Chandrasekhar 1998). The oxidants are incorporated into the fluid hydrocarbon chain matrix (between disks). LC is widely known as useful in electronic displays; however, there are in fact many non-display applications too. There are many applications of LC (especially ferroelectric LC) to electrically controlled light modulation: phase modulation, optical correlation, optical interconnects and switches, wavelength filters, and optical neural networks. In the latter case a ferroelectric LC is used to encode the weights in a neural network (Crossland and Wilkinson 1998).

### Conducting and Electroactive Polymers

Conducting polymer composites have been made that rapidly change their microwave reflection coefficient when an electric field is applied. When the field is removed, the composite reverts to its original state. Experiments have shown that the composite can change from one state to the other in the order of 100 ms (Wright et al. 2000). Also, some polymers exhibit electrochromism. These substances change their reflectance when a voltage is applied. This can be reversed by a change in voltage polarity (Mortimer 1997).

Electroactive polymers (Bar-Cohen 2001) are polymers that change their volume with the application of an electric field. They are particularly interesting as voltage-controlled artificial muscle. Organic semiconductors also look promising especially when some damage is introduced. Further details of electronic properties of polymers and organic crystals can be found in (Pope and Swenberg 1999).

### Voltage-Controlled Colloids

Colloids are suspensions of particles of submicron sizes in a liquid. The phase behavior of colloids is not fully understood. Simple colloids can self-assemble into crystals, while multicomponent suspensions can exhibit a rich variety of crystalline structures. There are also electrorheological fluids. These are suspensions of extremely fine nonconducting particles in an electrically insulating fluid. The viscosity of these fluids can be changed in a reversible way by large factors in response to an applied electric field in times of the order of milliseconds (Hao 2005). Also colloids can also be made in which the particles are charged making them easily manipulatable by suitable applied electric fields. Even if the particles are not charged, they may be moved through the action of applied fields using a phenomenon known as dielectrophoresis which is the motion of polarized but electrically uncharged particles in nonuniform electric fields (Khusid and Activos 1996). In work that echoes the methods of Pask

nearly four decades ago, dielectrophoresis has been used to grow tiny gold wires through a process of self-assembly (Hermanson et al. 2001).

### Langmuir-Blodgett Films

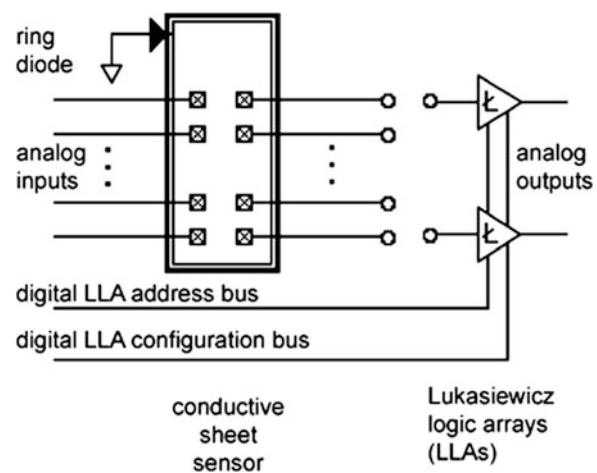
Langmuir-Blodgett films are molecular monolayers of organic material that can be transferred to a solid substrate (Petty 1996). They usually consist of hydrophilic heads and hydrophobic tails attached to the substrate. Multiple monolayers can be built and films can be built with very accurate and regular thicknesses. By arranging an electrode layer above the film it seems feasible that the local electronic properties of the layers could be altered. These systems look like feasible systems whose properties might be exploitable through computer-controlled evolution of the voltages.

### Kirchhoff-Lukasiewicz Machines

The work by Mills (Mills 1995a; Mills et al. 1989) also demonstrates the use of materials in computation. He has designed an “extended analog computer” (EAC) that is a physical implementation of a Kirchhoff-Lukasiewicz machine (KLM) (Mills 1995a). The machines are composed of logical function units connected to a conductive media, typically a conductive polymer sheet. The logical units implement the Lukasiewicz logic – a type of multivalued logic (Mills et al. 1989). Figure 2 shows how the Lukasiewicz logic arrays (LLAs) are connected to the conductive polymer. The

#### Evolution in Materio,

**Fig. 2** Kirchhoff-Lukasiewicz machine



LLAs bridge areas of the sheet together. The logic units measure the current at one point, perform a transformation, and then apply a current source to the other end of the bridge.

Computation is performed by applying current sinks and sources to the conductive polymer and reading the output from the LLAs. Different computations can be performed that are determined by the location of applied signals in the conducting sheet and the configuration of the LLAs. Hence, computation is performed by an interaction of the physics described by Kirchoff's laws and the Lukasiewicz logic units. Together they form a physical device that can solve certain kinds of partial differential equations. Using this form of analog computation, a large number of these equations can be solved in nanoseconds – much faster than on a conventional computer. The speed of computation is dependent on the materials used and how they are interfaced to digital computers, but it is expected that silicon implementations will be capable of finding tens of millions of solutions to the equations per second.

Examples of computation so far implemented in this system include robot control, control of a cyclotron beam (Mills 1995b), models of biological systems (including neural networks) (Mills 1995c), and radiosity-based image rendering.

One of the most interesting features of these devices is the programming method. It is very difficult to understand the actual processes used by the system to perform computation, and until recently most of the reconfiguration has been done manually. This is difficult as the system is not amenable to traditional software development approaches. However, evolutionary algorithms can be used to automatically define the parameters of the LLAs and the placement of current sinks and sources. By defining a suitable fitness function, the configuration of the EAC can be evolved – which removes the need for human interaction and for knowledge of the underlying system.

Although it is clear that such KLMs are clearly using the physical properties of a material to perform computation, the physical state of the material is not reconfigured (i.e., programmed); only the currents in the sheet are changed.

## **Evolution in Materio is Verified with Liquid Crystal**

Harding (Harding and Miller 2004) has verified Miller's intuition about the suitability of liquid crystal as an evolvable material by demonstrating that it is relatively easy to configure liquid crystal to perform various forms of computation.

In 2004, Harding constructed an analog processor that utilizes the physical properties of liquid crystal for computation. He evolved the configuration of the liquid crystal to discriminate between two square waves of many different frequencies. This demonstrated, for the first time, that the principle of using computer-controlled evolution was a viable and powerful technique for using non-silicon materials for computation. The analog processor consists of a passive liquid crystal display mounted on a reconfigurable circuit, known as an evolvable motherboard. The motherboard allows signals and configuration voltages to be routed to physical locations in the liquid crystal.

Harding has shown that many different devices can be evolved in liquid crystal including:

- Tone discriminator. A device was evolved in liquid crystal that could differentiate many different frequencies of square wave. The results were competitive, if not superior to those evolved in the FPGA.
- Logic gates. A variety of two input logic gates were evolved, showing that liquid crystal could behave in a digital fashion. This indicates that liquid crystal is capable of universal computation.
- Robot controller. An obstacle avoidance system for a simple exploratory robot was evolved. The results were highly competitive, with solutions taking fewer evaluations to find compared to other work on evolved robot controllers.

One of the surprising findings in this work has been that it turns out to be relatively easy to evolve the configuration of liquid crystal to solve tasks; i.e., only 40 generations of a modest population of configurations are required to evolve a very good frequency discriminator, compared to the thousands of generations required to evolve a similar

circuit on an FPGA. This work has shown that evolving such devices in liquid crystal is easier than when using conventional components, such as FPGAs. The work is a clear demonstration that evolutionary design can produce solutions that are beyond the scope of human design.

### Evolution in Materio Using Liquid Crystal: Implementational Details

An evolvable motherboard (EM) (Layzell 1998) is a circuit that can be used to investigate intrinsic evolution. The EM is a reconfigurable circuit that rewrites a circuit under computer control. Previous EMs have been used to evolve circuits containing electronic components (Layzell 1998; Crooks 2002); however, they can also be used to evolve in materio by replacing the standard components with a candidate material.

An EM is connected to an Evolvatron. This is essentially a PC that is used to control the evolutionary processes. The Evolvatron also has digital and analog I/O and can be used to provide test signals and record the response of the material under evolution (Figs. 3 and 4).

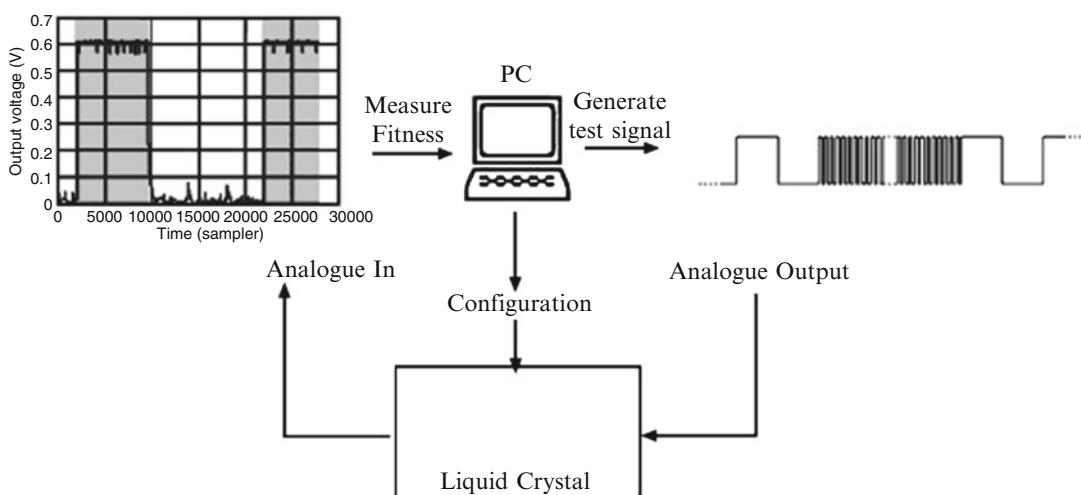
The liquid crystal evolvable motherboard (LCEM) is a circuit that uses four cross-switch matrix devices to dynamically configure the circuits connecting to the liquid crystal. The switches

are used to wire the 64 connections on the LCD to one of eight external connections. The external connections are input voltages, grounding, signals, and connections to measurement devices. Each of the external connectors can be wired to any of the connections to the LCD.

The external connections of the LCEM are connected to the Evolvatron's analog inputs and outputs. One connection was assigned for the incident signal, one for measurement, and the other for fixed voltages. The value of the fixed voltages is determined by the evolutionary algorithm but is constant throughout each evaluation.

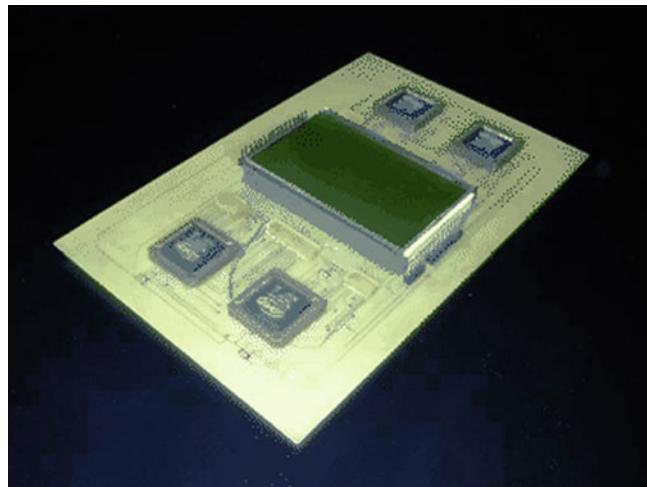
In these experiments the liquid crystal glass sandwich was removed from the display controller it was originally mounted on and placed on the LCEM. The display has a large number of connections (in excess of 200); however, because of PCB manufacturing constraints, we are limited in the size of connection we can make and hence the number of connections. The LCD is therefore roughly positioned over the pads on the PCB, with many of the PCB pads touching more than one of the connectors on the LCD. This means that we are applying configuration voltages to several areas of LC at the same time (Fig. 5).

Unfortunately neither the internal structure nor the electrical characteristics of the LCD are known. This raises the possibility that a configuration may be applied that would damage the

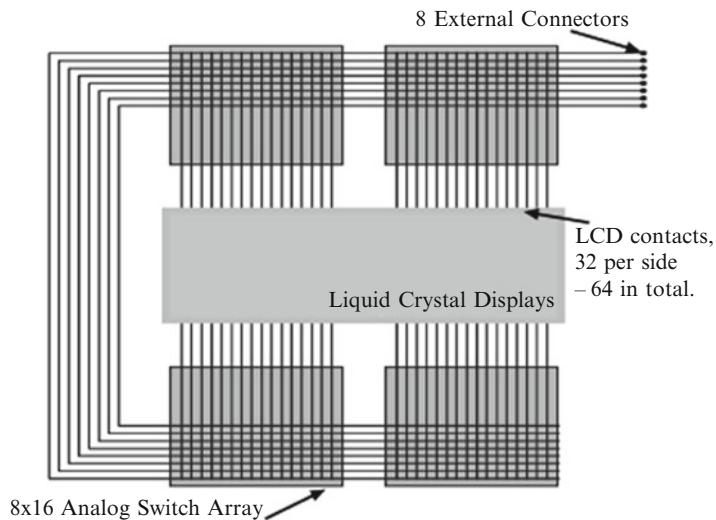


**Evolution in Materio, Fig. 3** Equipment configuration

**Evolution in Materio,**  
**Fig. 4** The LCEM



**Evolution in Materio,**  
**Fig. 5** Schematic  
of LCEM



device. The wires inside the LCD are made of an extremely thin material that could easily be burnt out if too much current flows through them. To guard against this, each connection to the LCD is made through a 4.7 K ohm resistor in order to provide protection against short circuits and to help limit the current in the LCD. The current supplied to the LCD is limited to 100 mA. The software controlling the evolution is also responsible for avoiding configurations that may endanger the device (such as short circuits).

It is important to note that other than the control circuitry for the switch arrays, there are no other active components on the motherboard –

only analog switches, smoothing capacitors, resistors, and the LCD are present.

#### Stability and Repeatability Issues

When the liquid crystal display is observed while solving a problem, it is seen that some regions of the liquid display go dark, indicating that the local molecular direction has been changed. This means that the configuration of the liquid crystal is changing while signals are being applied. To draw an analogy with circuit design, the incident signals would be changing component values or changing the circuit topology, which would have an effect on the behavior of the system.

This is likely to be detrimental to the measured performance of the circuit. When a solution is evolved, the fitness function automatically measures its stability over the period of the evaluation. Changes made by the incident signals can be considered part of the genotype-phenotype mapping. Solutions that cannot cope with their initial configurations being altered will achieve a low score. However, the fitness function cannot measure the behavior beyond the end of the evaluation time. Therein lies the difficulty; in evolution in materio long-term stability cannot be guaranteed.

Another issue concerns repeatability. When a configuration is applied to the liquid crystal, the molecules unlikely go back to exactly where they were when this configuration was tried previously. Assuming that there is a strong correlation between genotype and phenotype, then it is likely that evolution will cope with this extra noise. However, if evolved devices are to be useful, one needs to be sure that previously evolved devices will function in the same way as they did when originally evolved.

In (Stoica et al. 2000) it is noted that the behavior of circuits evolved intrinsically can be influenced by previous configurations; therefore, their behavior (and hence fitness) is dependent not only on the currently evaluated individual's configuration but on those that came before. It is worth noting that this is precisely what happens in natural evolution. For example, in a circuit, capacitors may still hold charge from a previously tested circuit. This charge would then affect the circuit operation; however, if the circuit was tested again with no stored charge, a different behavior would be expected, and a different fitness score would be obtained. Not only does this affect the ability to evolve circuits, but it would mean that some circuits are not valid. Without the influence of the previously evaluated circuits, the current solution may not function as expected. It is expected that such problems will have analogies in evolution in materio. The configurations are likely to be highly sensitive to initial conditions (i.e., conditions introduced by previous configurations).

## Dealing with Environmental Issues

A major problem when working with intrinsic evolution is separating out the computation allegedly being carried out by the target device, and that is actually done by the material being used. For example, while trying to evolve an oscillator, Bird and Layzell discovered that evolution was using part of the circuit for a radio antenna and picking up emissions from the environment (Bird and Layzell 2002). Layzell also found that evolved circuits were sensitive to whether or not a soldering iron was plugged (not even switched on) in another part of the room (Layzell 1998)!

An evolved device is not useful if it is highly sensitive to its environment in unpredictable ways, and it will not always be clear what environmental effects the system is using. It would be unfortunate to evolve a device for use in a spacecraft, only to find out it fails to work once out of range of a local radio tower!

To minimize these risks, we will need to check the operation of evolved systems under different conditions. We will need to test the behavior of a device using a different setup in a different location. It will be important to know if a particular configuration only works with one particular sample of a given material.

## The Computational Power of Materials

In (Lloyd 2000), Lloyd argued that the theoretical computing power of a kilogram of material is far more than is possible with a kilogram of traditional computer. He notes that computers are subject to the laws of physics and that these laws place limits on the maximum speed they can operate and the amount of information it can process. Lloyd shows that if we were fully able to exploit a material, we would get an enormous increase in computing power. For example, with 1 kg of matter we should be able to perform roughly  $5 \times 10^{50}$  operations per second and store  $10^{31}$  bits. Amazingly, contemporary quantum computers do operate near these theoretical limits (Lloyd 2000).

A small amount of material also contains a large number of components (regardless of

whether we consider the molecular or atomic scale). This leads to some interesting thoughts. If we can exploit materials at this level, we would be able to do a vast amount of computation in a small volume. A small size also hints at low power consumption, as less energy has to be spent to perform an operation. Many components also provide a mechanism for reliability through redundancy. A particularly interesting observation, especially when considered in terms of non-von Neumann computation, is the massive parallelism we may be able to achieve. The reason that systems such as quantum, DNA, and chemical computation can operate so quickly is that many operations are performed at the same time. A programmable material might be capable of performing vast numbers of tasks simultaneously and therefore provide a computational advantage.

In commercial terms, small is often synonymous with low cost. It may be possible to construct devices using cheaply available materials. Reliability may not be an issue, as the systems could be evolved to be massively fault tolerant using their intrinsic redundancy. Evolution is capable of producing novel designs. Koza has already rediscovered circuits that infringe on recent patents, and his genetic programming method has “invented” brand new circuit designs (Koza 1999). Evolving in materio could produce many novel designs, and indeed given the infancy of programmable materials, all designs may be unique and hence patentable.

## Future Directions

The work described here concerning liquid crystal computational devices is at an early stage. We have merely demonstrated that it is possible to evolve configurations of voltages that allow a material to perform desired computations. Any application that ensues from this work is unlikely to be a replacement for a simple electronic circuit. We can design and build those very successfully. What we have difficulty with is building complex, fault-tolerant systems for performing complex computation. It appears that nature managed to do this. It used a simple process of a repetitive

test and modify, and it did this in a universe of unimaginable physical complexity. If nature can exploit the physical properties of a material and its surroundings through evolution, then so should we.

There are many important issues that remain to be addressed. Although we have made some suggestions about materials worthy of investigation, it is at present unclear which materials are most suitable. An experimental platform needs to be constructed that allows many materials to be tried and investigated. The use of microelectrode arrays in a small-volume container would allow this. This would also have the virtue of allowing internal signals in the materials to be inspected and potentially understood.

We need materials that are rapidly configurable. They must not be fragile and sensitive to minute changes in physical setup. They must be capable of maintaining themselves in a stable configuration. The materials should be complex and allow us to carry out difficult computations more easily than conventional means. One would like materials that can be packaged into small volumes. The materials should be relatively easily interfaced with. So far, material systems have been configured by applying a constant configuration pattern; however, this may not be appropriate for all systems. It may be necessary to put the physical system under some form of responsive control, in order to program and then keep the behavior stable.

We may or may not know if a particular material can be used to perform some form of computation. However, we can treat our material as a “black box” and using evolution as a search technique, automatically discover what, if any, computations our black box can perform. The first step is to build an interface that will allow us to communicate with a material. Then we will use evolution to find a configuration we can apply using this platform and then attempt to find a mapping from a given problem to an input suitable for that material and a mapping from the material’s response to an output. If this is done correctly, we might be automatically able to tell if a material can perform computation and then classify the computation.

When we evolve in materio, using mappings evolved in software, how can we tell when the material is giving us any real benefit? The lesson

of evolution in materio has been that the evolved systems can be very difficult to analyze, and the principal obstacle to the analysis is the problem of separating out the computational role that each component plays in the evolved system. These issues are by no means just a problem for evolution in materio. They may be an inherent part of complex evolved systems. Certainly the understanding of biological systems is providing immense challenges to scientists.

The single most important aspect that suggests that evolution in materio has a future is that natural evolution has produced immensely sophisticated material computational systems. It would seem foolish to ignore this and merely try to construct computational devices that operate according to one paradigm of computation (i.e., Turing). Oddly enough, it is precisely the sophistication of the latter that allows us to attempt the former.

## Bibliography

### Primary Literature

- Adamatzky A, Costello BDL, Asai T (2005) Reaction-diffusion computers. Elsevier, Amsterdam
- Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266(11):1021–1024
- Amos M (2005) Theoretical and experimental DNA computation. Springer, Berlin
- Bar-Cohen Y (2001) Electroactive polymer (EAP) actuators as artificial muscles – reality, potential and challenges. SPIE Press
- Bird J, Layzell P (2002) The evolved radio and its implications for modelling the evolution of novel sensors. In: Proceedings of congress on evolutionary computation, pp 1836–1841
- Bissell C (2004) A great disappearing act: the electronic analogue computer. In: IEEE conference on the history of electronics, pp 28–30
- Cariani P (1993) To evolve an ear: epistemological implications of gordon pask's electrochemical devices. *Syst Res* 3:19–33
- Chandrasekhar S (1998) Columnar, discotic nematic and lamellar liquid crystals: their structure and physical properties. In: handbook of liquid crystals, vol 2B. Wiley-VCH, pp 749–780
- Conrad M (1988) The price of programmability. The universal Turing machine. pp 285–307
- Crooks J (2002) Evolvable analogue hardware. Meng project report, The University of York
- Crossland WA, Wilkinson TD (1998) Nondisplay applications of liquid crystals. In: Handbook of liquid crystals, vol 1. Wiley-VCH, pp 763–822
- Demus D, Goodby JW, Gray GW, Spiess HW, Vill V (eds) (1998) Handbook of liquid crystals, vol 4. Wiley-VCH, New York, p 2180
- Deutsch D (1985) Quantum theory, the church-turing principle and the universal quantum computer. *Proc R Soc Lond A* 400:97–117
- Goldberg D (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading
- Hao T (2005) Electrorheological fluids: the non-aqueous suspensions. Elsevier Science, Amsterdam
- Harding S, Miller JF (2004) Evolution in materio: a tone discriminator in liquid crystal. In: Proceedings of the congress on evolutionary computation 2004 (CEC'2004), vol 2, pp 1800–1807
- Hermanson KD, Lumsdon SO, Williams JP, Kaler EW, Velev OD (2001) Dielectrophoretic assembly of electrically functional microwires from nanoparticle suspensions. *Science* 294:1082–1086
- Holland J (1992) Adaptation in natural and artificial systems, 2nd edn. MIT Press, Cambridge
- Khoo IC (1995) Liquid crystals: physical properties and nonlinear optical phenomena. Wiley, New York
- Khoo IC, Slussarenko S, Guenther BD, Shih MY, Chen P, Wood WV (1998) Optically induced space-charge fields, dc voltage, and extraordinarily large nonlinearity in dye-doped nematic liquid crystals. *Opt Lett* 23(4):253–255
- Khusid B, Activos A (1996) Effects of interparticle electric interactions on dielectrophoresis in colloidal suspensions. *Phys Rev E* 54(5):5428–5435
- Koza JR (1999) Human-competitive machine intelligence by means of genetic algorithms. In: Booker L, Forrest S, Mitchell M, Riolo R (eds) Festschrift in honor of John H Holland. Center for the Study of Complex Systems, Ann Arbor, pp 15–22
- Langton C (1991) Computation at the edge of chaos: phase transitions and emergent computation. In: Emergent computation, MIT Press, pp 12–37
- Laughlin RB, Pines D, Schmalian J, Stojkovic BP, Wolynes P (2000) The middle way. *Proc Natl Acad Sci* 97(1):32–37
- Layzell P (1998) A new research tool for intrinsic hardware evolution. In: Proceedings of the second international conference on evolvable systems: from biology to hardware. Lecture notes in computer science, vol 1478. Springer, Berlin, pp 47–56
- Linden DS, Altshuler EE (1999) Evolving wire antennas using genetic algorithms: a review. In: 1st NASA/DoD workshop on evolvable hardware. IEEE Computer Society, pp 225–232
- Linden DS, Altshuler EE (2001) A system for evolving antennas in-situ. In: 3rd NASA/DoD workshop on evolvable hardware. IEEE Computer Society, pp 249–255
- Lindoy LF, Atkinson IM (2000) Self-assembly in supramolecular systems. Royal Society of Chemistry, Cambridge
- Lloyd S (2000) Ultimate physical limits to computation. *Nature* 406:1047–1054
- Miller JF, Downing K (2002) Evolution in materio: looking beyond the silicon box. In: Proceedings of NASA/DoD evolvable hardware workshop, pp 167–176

- Mills JW (1995) Polymer processors. Technical report TR580, Department of Computer Science, University of Indiana
- Mills JW (1995) Programmable vlsi extended analog computer for cyclotron beam control. Technical report TR441, Department of Computer Science, University of Indiana
- Mills JW (1995) The continuous retina: image processing with a single sensor artificial neural field network. Technical report TR443, Department of Computer Science, University of Indiana
- Mills JW, Beavers MG, Daffinger CA (1989) Lukasiewicz logic arrays. Technical report TR296, Department of Computer Science, University of Indiana
- Mitchell M (1996) An introduction to genetic algorithms. MIT Press, Cambridge, MA
- Mortimer RJ (1997) Electrochromic materials. *Chem Soc Rev* 26:147–156
- Pask G (1958) Physical analogues to the growth of a concept. In: Mechanization of thought processes. Symposium 10, National Physical Laboratory, pp 765–794
- Pask G (1959) The natural history of networks. In: Proceedings of international tracts. In: Computer science and technology and their application, vol 2, pp 232–263
- Petty MC (1996) Langmuir-Blodgett films: an introduction. Cambridge University Press, Cambridge
- Pickering A (2002) Cybernetics and the mangle: Ashby, beer and pask. *Soc Stud Sci* 32:413–437
- Pope M, Swenberg CE (1999) Electronic processes of organic crystals and polymers. Oxford University Press, Oxford
- Stepney S, Braunstein SL, Clark JA, Tyrrell A, Adamatzky A, Smith RE, Addis T, Johnson C, Timmis J, Welch P, Milner R, Partridge D (2005) Journeys in non-classical computation I: a grand challenge for computing research. *Int J Parallel Emerg Distrib Syst* 20(1):5–19
- Stepney S, Braunstein S, Clark J, Tyrrell A, Adamatzky A, Smith R, Addis T, Johnson C, Timmis J, Welch P, Milner R, Partridge D (2006) Journeys in non-classical computation II: initial journeys and waypoints. *Int J Parallel Emerg Distrib Syst* 21(2):97–125
- Stoica A, Zebulum RS, Keymeulen D (2000) Mixtrinsic evolution. In: Proceedings of the third international conference on evolvable systems: from biology to hardware (ICES2000). Lecture notes in computer science, vol 1801. Springer, Berlin, pp 208–217
- Stoica A, Zebulum RS, Guo X, Keymeulen D, Ferguson MI, Duong V (2003) Silicon validation of evolution-designed circuits. In: Proceedings. NASA/DoD conference on evolvable hardware, pp 21–25
- Thompson A (1996) An evolved circuit, intrinsic in silicon, entwined with physics. ICES 390–405
- Thompson A (1998) On the automatic design of robust electronics through artificial evolution. In: Sipper M, Mange D, Pérez-Uribe A (eds) Evolvable systems: from biology to hardware, vol 1478. Springer, New York, pp 13–24
- Thompson A, Harvey I, Husbands P (1996) Unconstrained evolution and hard consequences. In: Sanchez E, Tomassini M (eds) Towards evolvable hardware: the evolutionary engineering approach. Lecture notes in computer science, vol 1062. Springer, Berlin, pp 136–165
- Toffoli T (2005) Nothing makes sense in computing except in the light of evolution. *Int J Unconv Comput* 1(1): 3–29
- Turing AM (1936) On computable numbers, with an application to the entscheidungsproblem. *Proc Lond Math Soc* 42(2):230–265
- UK Computing Research Committee (2005) Grand challenges in computer research. [http://www.ukcrc.org.uk/grand\\_challenges/](http://www.ukcrc.org.uk/grand_challenges/)
- Weiss R, Basu S, Hooshangi S, Kalmbach A, Karig D, Mehreja R, Netravali I (2003) Genetic circuit building blocks for cellular computation, communications, and signal processing. *Nat Comput* 2(1):47–84
- Wright PV, Chambers B, Barnes A, Lees K, Despotakis A (2000) Progress in smart microwave materials and structures. *Smart Mater Struct* 9:272–279

## Books and Reviews

- Analog Computer Museum and History Center. Analog computer reading list. <http://dcoward.best.vwh.net/analog/readlist.htm>
- Bringsjord S (2001) In computation, parallel is nothing, physical everything. *Minds Mach* 11(1):95–99
- Feynman RP (2000) Feynman lectures on computation. Perseus Books Group, Reading
- Fifer S (1961) Analogue computation: theory, techniques, and applications. McGraw-Hill, New York
- Greenwood GW, Tyrrell AM (2006) Introduction to evolvable hardware: a practical guide for designing self-adaptive systems. Wiley-IEEE Press, Piscataway
- Hey AJG (ed) (2002) Feynman and computation. Westview Press
- Penrose R (1989) The emperor's new mind, concerning computers, minds, and the laws of physics. Oxford University, Oxford
- Piccinini G. The physical church-turing thesis: modest or bold? <http://www.umsl.edu/~piccininig/CTModestorBold5.htm>
- Raichman N, Ben-Jacob N, Segev R (2003) Evolvable hardware: genetic search in a physical realm. *Phys A* 326:265–285
- Sekanina L (2004) Evolvable components: from theory to hardware implementations, 1st edn. Springer, Heidelberg
- Siegelmann HT (1999) Neural networks and analog computation, beyond the Turing limits. Birkhauser, Boston
- Sienko T, Adamatzky A, Rambidi N, Conrad M (2003) Molecular computing. MIT Press, Cambridge
- Thompson A (1999) Hardware evolution: automatic design of electronic circuits in reconfigurable hardware by artificial evolution, 1st edn. Springer, Heidelberg



## Reversible Computing

Kenichi Morita

Hiroshima University, Higashi-Hiroshima, Japan

### Article Outline

#### Glossary

#### Definition of the Subject

#### Introduction

#### Reversible Logic Elements and Circuits

#### Reversible Turing Machines

#### Other Models of Reversible Computing

#### Future Directions

#### Bibliography

### Glossary

**Billiard ball model** The billiard ball model (BBM) is a physical model of computation proposed by Fredkin and Toffoli (1982). It consists of idealized balls and reflectors. Balls can collide with other balls or reflectors. It is a reversible dynamical system, since it is assumed that collisions are elastic, and there is no friction. Fredkin and Toffoli showed that a reversible logic gate called Fredkin gate, which is known to be logically universal, can be embedded in BBM. Hence, a universal computer can be realized in the space of BBM.

**Reversible cellular automaton** A cellular automaton (CA) consists of a large number of finite automata called cells interconnected uniformly, and each cell changes its state depending on its neighboring cells. A reversible cellular automaton (RCA) is one whose global function (i.e., a transition function from the configurations to the configurations) is injective. RCAs can be thought as spatiotemporal models of reversible physical systems as well as reversible computing models. (See the entry “Reversible Cellular Automata.”)

**Reversible logic element** It is a primitive of which logic circuits can be composed and whose operation is defined by an injective function. There are two kinds of such elements: those without memory and those with memory. A reversible logic element without memory is nothing but a reversible logic gate. A reversible logic element with memory is a kind of a reversible finite automaton with an output, which is also useful for constructing reversible computing systems.

**Reversible logic element with memory** A reversible logic element with memory (RLEM) is defined as a reversible sequential machine, a kind of a finite automaton that has an output as well as an input. Two-state RLEMs are particularly important. A rotary element (RE) is a typical example of a two-state RLEM (Morita 2001). It is known that reversible computing systems such as reversible Turing machines can be built very concisely using only REs.

**Reversible logic gate** It is a gate whose logical function is an injection. Fredkin gate and Toffoli gate are typical reversible logic gates with three input lines and three output lines. They are universal reversible logic gates in the sense that any logical function (even if it is not injective) can be realized by using only Fredkin or Toffoli gate by allowing constant inputs and garbage outputs (i.e., useless signals). It is shown by Fredkin and Toffoli (1982) that garbage signals can be reversibly erased, and hence any logical function is realized by a garbageless circuit composed only of a universal gate.

**Reversible Turing machine** A reversible Turing machine (RTM) is a TM such that every computational configuration has at most one predecessor, and thus it is a “backward deterministic” TM. Bennett (1973) showed that for any (irreversible) Turing machine, there is an RTM that simulates the former and leaves no garbage information on the tape when it halts. Hence, Turing machines still have computational universality even if the constraint of reversibility is added.

## Definition of the Subject

A reversible computing system is defined as a “backward deterministic” system, where every computational configuration (state of the whole system) has exactly one previous configuration. Hence, a backward computation can be performed by its “inverse” computing system. Though its definition is rather simple, it is closely related to physical reversibility. The study of reversible computing originated from an investigation of heat generation (or energy dissipation) in computing systems. It deals with problems on how computation can be carried out efficiently and elegantly in reversible computing models and how such systems can be implemented in reversible physical systems. Since reversibility is one of the fundamental microscopic physical laws of nature, and future computing systems will surely become a nanoscale size, it is very important to investigate these problems.

In this entry, we discuss several models of reversible computing from the viewpoint of computing theory (hence, we do not discuss problems of physical implementation in detail). We deal with several reversible computing models of different levels: from an element level to a system level. They are reversible logic elements and circuits, the billiard ball model – an idealized reversible physical model, reversible Turing machines, and some other computing systems. We see these models are related to each other, i.e., reversible Turing machines are composed of reversible logic elements, and reversible logic elements are implemented in the billiard ball model. We can also see even quite simple reversible systems have very rich computing ability in spite of the constraint of reversibility. Such computing models will give new insights for the future computing systems.

## Introduction

The definitions of reversible computing systems are rather simple. They are the systems whose computation process can be traced in the reverse direction in a unique way from the end of the computation to the start of it. In other words,

they are “backward deterministic” computing models (of course, a more precise definition should be given for each model of reversible computing). But, if we explain reversible computing systems only in this way, it is hard to understand why they are interesting. Their importance lies in the fact that it is closely related to physical reversibility.

Landauer (1961) argued the relation between logical irreversibility and physical irreversibility. He pointed out that any irreversible logical operation, such as simple erasure of information from the memory, or simple merge of two paths of a program, is associated with physical irreversibility, and hence it necessarily causes heat generation. In particular, if 1 bit of information is erased, at least  $kT \ln 2$  of energy will be dissipated, where  $k$  is the Boltzmann constant and  $T$  is the absolute temperature. This is called “Landauer’s principle.” (Note that besides simple erasure and simple merge, there are reversible erasure and reversible merge, which are not irreversible operations and are useful for constructing garbage-less reversible Turing machine as we shall see in section “Reversible Turing Machines.”) After the work of Landauer, various studies related to “thermodynamics of computation” appeared (Bennett 1973, 1982, 1987, 2003; Bennett and Landauer 1985; Keyes and Landauer 1970). There is also an argument on computation from the physical viewpoint by Feynman (1996), which contains an idea leading to quantum computing (see, e.g., Deutsch 1985; Feynman 1982; Gruska 1999).

Bennett (1973) studied reversible Turing machine from the viewpoint of Landauer’s principle. He proved that for any irreversible Turing machine, there is a reversible Turing machine that simulates the former and leaves no garbage information on the tape. The notion of garbage-less computation is important, because disposal of garbage information is actually equivalent to erasure of the information, and hence leads to energy dissipation. The result of Bennett suggests that any computation can be performed efficiently in a physically reversible system.

Since then, various kinds of reversible computing models have been proposed and investigated: reversible logic elements and circuits (Fredkin

and Toffoli 1982; Morita 2001; Toffoli 1980, 1981), the billiard ball model (Fredkin and Toffoli 1982) – an idealized physical model of computation, reversible cellular automata (Toffoli 1977; Toffoli and Margolus 1990), reversible counter machines (Morita 1996), and so on. It has been known that all these models have *computational universality* (or *Turing universality*), i.e., any (irreversible) Turing machine can be simulated in these models.

In this entry, we discuss reversible computing from the theoretical point of view. We show how they can have computational universality and how different they are from conventional models. We can see that in some cases, computation in reversible systems is carried out in a very different fashion from that of traditional irreversible computing models.

An outline of this article is as follows. In section “Reversible Logic Elements and Circuits,” two types of reversible logic elements, i.e., those without memory and with memory, are discussed, and it is shown how logic circuits are composed of them. In section “Reversible Turing machines,” a reversible version of a Turing machine is investigated as a theoretical model of a reversible computer. In section “Other Models of Reversible Computing,” a survey on several kinds of reversible computing models other than reversible Turing machines is given. In section “Future Directions,” prospective research problems and open problems are given.

## Reversible Logic Elements and Circuits

A *logic element* is a primitive such that logic circuits are constructed out of it. If we choose a set of logic elements adequately, we can realize a computing system as a circuit using only the elements in the set. A reversible logic element is one whose operation is described by an injective mapping. There are two kinds of such elements: those without memory and those with memory. A memory-less logic element is usually called a *logic gate*. For example, AND, OR, and NOT gates are typical ones used in the traditional logic circuits. A logic element with memory is a

kind of finite automaton having an output as well as an input. As we shall see, the latter type of element is useful for constructing reversible computing systems. In this section, we explore useful reversible logic elements and investigate how we can compose reversible machines from them. We also discuss the relation between the reversible logic elements and reversible physical systems using the billiard ball model of computation proposed in Fredkin and Toffoli (1982).

### Reversible Logic Gate

**Definition 1** An  $m$ -input  $n$ -output logic gate is an element that realizes a logical function  $f: \{0, 1\}^m \rightarrow \{0, 1\}^n$ . It is called a reversible logic gate, if  $f$  is injective (note that  $m \leq n$  must hold in this case).

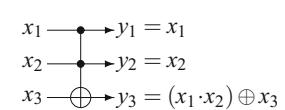
An early study on reversible logic gates was made by Petri (1967). Later, Toffoli (1980, 1981) and Fredkin and Toffoli (1982) studied them in connection with physical reversibility. We first give typical examples of reversible logic gates.

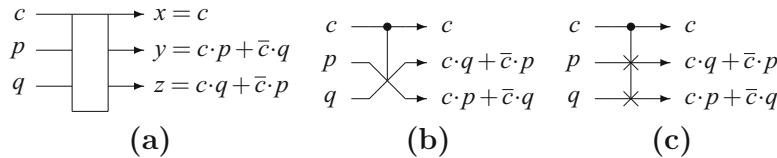
The *generalized AND/NAND gate* of order  $n$  proposed by Toffoli (1980) is one that realizes the function  $\theta^{(n)}: (x_1, \dots, x_{n-1}, x_n) \mapsto (x_1, \dots, x_{n-1}, (x_1 \cdot x_2 \cdots x_{n-1}) \oplus x_n)$ , where  $(x_1, \dots, x_n) \in \{0, 1\}^n$  and  $\oplus$  is the operation of exclusive OR. We can easily verify that  $\theta^{(n)}$  is injective, and  $(\theta^{(n)})^{-1} = \theta^{(n)}$ . The gate that realizes  $\theta^{(2)}$  is sometimes called the *controlled NOT (CNOT)*, which is an important gate in quantum computing (see, e.g., Gruska 1999). In fact, it is shown that all unitary operations are expressed as compositions of CNOT gates and 1-bit quantum gates (Barenco et al. 1995). The gate for  $\theta^{(3)}$  is called the *Toffoli gate* (Fig. 1).

*Fredkin gate* proposed by Fredkin and Toffoli (1982) is a 3-input 3-output logic gate that realizes the function  $\varphi: (c, p, q) \mapsto (c, c \cdot p + \bar{c} \cdot q, \bar{c} \cdot p + c \cdot q)$ . Figure 2 shows its pictorial representation. It is also easy to verify that  $\varphi$  is injective and  $\varphi^{-1} = \varphi$  holds.

### Reversible Computing,

**Fig. 1** The generalized AND/NAND gate of order 3 called Toffoli gate

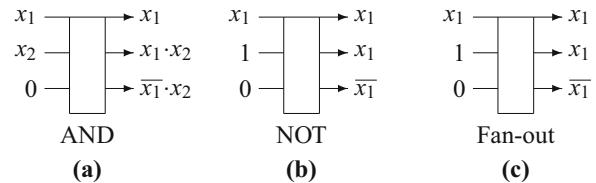




**Reversible Computing, Fig. 2** Fredkin gate. The pictorial representation (a) is the original one given in (Fredkin and Toffoli 1982), while (b, c) are often used in the design theory of quantum logic circuits

## **Reversible Computing,**

**Fig. 3** Implementing AND, NOT, and fan-out by Fredkin gate



**Definition 2** A set  $E$  of logic elements is called logically universal, if any logical function can be realized by a circuit composed only of the elements in  $E$ .

$$\forall \mathbf{x} \in \{0, 1\}^m, \forall \mathbf{y} \in \{0, 1\}^n, \exists \mathbf{g} \in \{0, 1\}^{m+k-n} : \\ f(\mathbf{x}) = \mathbf{y} \Rightarrow F(\mathbf{x}, \mathbf{c}) = (\mathbf{y}, \mathbf{g})$$

It is well known that the set {AND, NOT, fan-out} is logically universal, where “fan-out” is the function such that  $x \mapsto (x, x)$  ( $x \in \{0, 1\}$ ), i.e., the copying function. If we add a delay element (i.e., a memory) to a logically universal set, we can construct any sequential machine and thus can build a universal computer from them (as an infinite circuit).

We can see that AND, NOT, and fan-out can be realized by Fredkin gates (Fredkin and Toffoli 1982) as shown in Fig. 3, if we allow to supply *constant inputs* 0s and 1s and allow to generate *garbage outputs* besides the true inputs and outputs. For example, in Fig. 3a, the constant 0 is supplied to the third input line, and garbage signals  $x_1$  and  $\overline{x_1} \cdot x_2$  are generated besides the true output  $x_1 \cdot x_2$ . By above, we have the following theorem.

**Theorem 1** (Fredkin and Toffoli 1982) *The set {Fredkin gate} is logically universal, if we allow constant inputs and garbage outputs in a circuit. More precisely, for any logical function  $f: \{0, 1\}^m \rightarrow \{0, 1\}^n$ , which is generally non-injective, there exist  $k \geq 0$ , a constant  $\mathbf{c} \in \{0, 1\}^k$ , and an injection  $F: \{0, 1\}^{m+k} \rightarrow \{0, 1\}^{m+k}$  that is realized by a circuit composed of Fredkin gates and satisfies the following.*

This theorem states that any logical function  $f$  can be realized by a circuit composed only of Fredkin gates, if we allow to give a constant  $c$  and to generate a garbage  $g$  in the circuit.

It is also possible to show logical universality of the set {Toffoli gate} (this is left as an exercise for the readers). Besides Fredkin gate and Toffoli gate, there are many logically universal ones. The total number of 3-input 3-output reversible logic gates is  $8! = 40320$ , and most of them are logically universal. In fact, it is known that 38976 gates among them are universal (De Vos 2010). On the other hand, Toffoli (1980) showed any 2-input 2-output reversible gate is composed only of CNOT and NOT. Since it is easy to see that {CNOT, NOT} is not logically universal, we can conclude no set of 2-input 2-output reversible gates is logically universal.

Fredkin and Toffoli (1982) also showed a method of reducing garbage signals in a circuit composed of reversible logic gates.

**Theorem 2** (Fredkin and Toffoli 1982) For any logical function  $f: \{0, 1\}^m \rightarrow \{0, 1\}^n$ , there exist  $k \geq 0$ , a constant  $\mathbf{c} \in \{0, 1\}^k$ , and an injection  $G: \{0, 1\}^{m+2n+k} \rightarrow \{0, 1\}^{m+2n+k}$  that is realized by a circuit composed of Fredkin gates and satisfies the following.

$$\forall \mathbf{x} \in \{0, 1\}^m, \forall \mathbf{y} \in \{0, 1\}^n : \\ f(\mathbf{x}) = \mathbf{y} \Rightarrow G(\mathbf{x}, \mathbf{c}, \mathbf{0}, \mathbf{1}) = (\mathbf{x}, \mathbf{c}, \mathbf{y}, \bar{\mathbf{y}})$$

Here,  $\mathbf{0} \in \{0\}^n$ ,  $\mathbf{1} \in \{1\}^n$ , and  $\bar{\mathbf{y}}$  is the bitwise complement of  $\mathbf{y}$ .

The circuit for  $G$  is composed of three modules: a circuit for  $F$  in Theorem 1, its inverse circuit, and a fan-out circuit consisting of  $n$  copies of Fig. 3c. Note that, the “inverse” of a circuit composed of Fredkin gates is obtained by exchanging the input and the output of each gate, since the inverse of Fredkin gate is itself. The circuit for  $G$  first computes  $F(\mathbf{x}, \mathbf{c}) = (\mathbf{y}, \mathbf{g})$  by the circuit for  $F$ . Then, it computes the mapping  $(\mathbf{y}, \mathbf{0}, \mathbf{1}) \mapsto (\mathbf{y}, \mathbf{y}, \bar{\mathbf{y}})$  by the fan-out circuit to obtain a copy of  $\mathbf{y}$ . Finally,  $F^{-1}(\mathbf{y}, \mathbf{g}) = (\mathbf{x}, \mathbf{c})$  is computed by the circuit for  $F^{-1}$ , and thus the garbage is reversibly erased and converted into the clean constant.

The circuit for  $G$  does not give a large amount of garbage signals. But, it produces  $\mathbf{x}$  and  $\bar{\mathbf{y}}$  besides the true output  $\mathbf{y}$ , which may be regarded as another kind of garbage (though the amount is generally smaller than that of  $\mathbf{g}$ ). Especially, when we construct a circuit with memories (i.e., a sequential machine), this kind of garbage is produced at every time step. In this sense, the above circuit is an “almost” garbage-less one. However, as we shall see later, if we construct only “reversible” sequential machines, then completely garbage-less circuits can be obtained.

### Reversible Logic Element with Memory

In the classical design theory of logic circuits, two sorts of logic elements are supposed. They are logic gates (such as AND, OR, NOT, etc.) and a delay element (i.e., a memory like a flip-flop). Its design technique is also divided into two phases: First, we implement Boolean functions as combinatorial logic circuits consisting of gates and then construct sequential machines from combinatorial logic circuits and delay elements. In this way, the entire process of constructing digital circuits is systematized, and various methods of circuit minimization can be applied, especially for the first phase.

Using reversible logic gates and delay elements is a method along this line. However, *reversible logic elements with memory* (RLEM) are also useful in reversible computing. The advantages of using RLEM are as follows. First, as we shall see below, reversible computing systems, such as reversible Turing machines, can be constructed very concisely using RLEM. In addition, construction methods are very different from the traditional ones using logic gates, and thus RLEM give new vistas in the theory of reversible computing. Second, even very simple RLEM are universal in the sense any reversible sequential machine can be built from them. In fact, it is known that *all* the two-state RLEM except only four are universal. Third, in a circuit composed of RLEM, there is no need of synchronizing two or more coming signals at each element. This is because a signal coming from some other place interacts only with the state of the RLEM, which can be regarded as a “stationary” signal. Hence, no clock signal is required.

### Definitions on Reversible Logic Elements with Memory

We first give a definition of a sequential machine of Mealy type (i.e., a finite automaton with an output) and its reversibility.

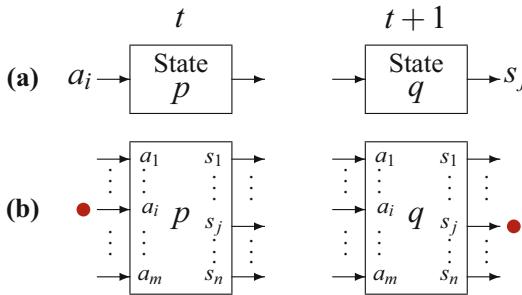
**Definition 3** A sequential machine ( $SM$ ) is defined by

$$M = (Q, \Sigma, \Gamma, q_0, \delta),$$

where  $Q$  is a finite nonempty set of states;  $\Sigma$  and  $\Gamma$  are finite nonempty sets of input and output symbols, respectively;  $q_0 \in Q$  is an initial state; and  $\delta : Q \times \Sigma \rightarrow Q \times \Gamma$  is a mapping called a move function. A variation of an  $SM$   $M = (Q, \Sigma, \Gamma, \delta)$ , where no initial state is specified, is also called an  $SM$  for convenience. The  $SM$   $M$  is called a reversible sequential machine ( $RSM$ ) if  $\delta$  is injective (hence  $|\Sigma| \leq |\Gamma|$ ).

A reversible logic element with memory is an  $RSM$  with the same numbers of input and output symbols.

**Definition 4** A reversible logic element with memory ( $RLEM$ ) is a reversible sequential machine  $M = (Q, \Sigma, \Gamma, \delta)$  such that  $|\Sigma| = |\Gamma|$ . It is also called a  $|Q|$ -state  $|\Gamma|$ -symbol  $RLEM$ .



**Reversible Computing, Fig. 4** (a) An operation of an RLEM and (b) an interpretation of the RLEM as a module having decoded input ports and output ports

In an RLEM  $M$ , if the present state is  $p$ , the input symbol is  $a_i$ , and  $\delta(p, a_i) = (q, s_j)$ , then the next state is  $q$ , and the output is  $s_j$  as in Fig. 4a. To use an RLEM as a logic element for composing a logic circuit, we interpret it as an RLEM having “decoded” input ports and output ports as shown in Fig. 4b. Namely, for each input symbol, there is a unique input port, to which a particle can be given. Likewise, for each output symbol, there is a unique output port, from which a particle can appear. Hence, particles should not be given to two or more input ports at a time. When composing a circuit using RLEMs, we assume each output port of an RLEM can be connected to at most one input port of another (may be the same) RLEM. Thus, fan-out of an output is not allowed.

### Rotary Element: A Typical RLEM

The class of 2-state RLEMs is particularly important, since they are simple yet powerful. We consider a specific 2-state RLEM called *rotary element* (RE) (Morita 2001) defined as follows.

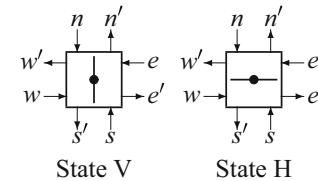
$$M_{\text{RE}} = (\{H, V\}, \{n, e, s, w\}, \{n', e', s', w'\}, \delta_{\text{RE}})$$

The move function  $\delta_{\text{RE}}$  is shown in Table 1.

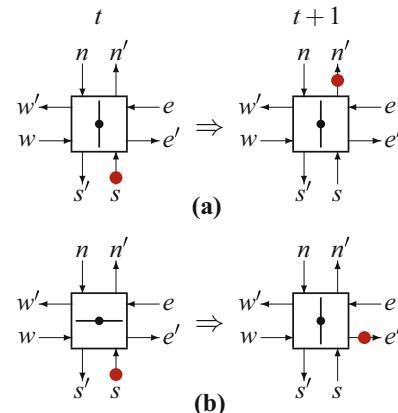
The operation of RE can be understood by the following intuitive interpretation. It has two states H and V; four input lines  $n, e, s$ , and  $w$ ; and four output lines  $n', e', s'$ , and  $w'$  (Fig. 5). A particle (or token) can be given to one of the input lines. We interpret that RE has a “rotatable bar” to control the moving direction of the particle. When no particle exists, nothing happens. If a

**Reversible Computing, Table 1** The move function  $\delta_{\text{RE}}$  of rotary element  $M_{\text{RE}}$

Present state	Input			
	$n$	$e$	$s$	$w$
State H	V $w'$	H $w'$	V $e'$	H $e'$
State V	V $s'$	H $n'$	V $n'$	H $s'$



**Reversible Computing, Fig. 5** Two states of rotary element (RE)

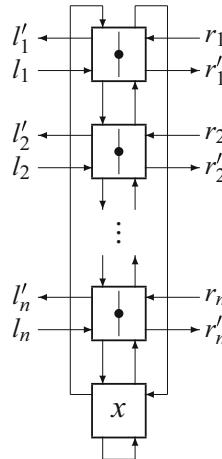


**Reversible Computing, Fig. 6** Operations of RE: (a) the parallel case and (b) the orthogonal case

particle comes from a direction parallel to the bar, then it goes out from the output line of the opposite side (i.e., it goes straight ahead) without affecting the direction of the bar (Fig. 6a). If a particle comes from a direction orthogonal to the bar, then it makes a right turn and rotates the bar by 90° (Fig. 6b).

The following theorem states that RE is *universal* in the sense any RSM can be realized as a circuit composed of REs.

**Theorem 3** (Morita 2003) *For any reversible sequential machine  $M$ , there is a garbage-less circuit  $C$  composed only of REs that simulates*

**Reversible Computing,****Fig. 7** RE column, where  $x \in \{H, V\}$ **Reversible Computing, Table 2** The move function of the RE column

	Input	
State $x$	$l_i$	$r_i$
$H$ (marked)	$V l'_i$	$H l'_i$
$V$ (unmarked)	$V r'_i$	$H r'_i$

*M. The circuit C is completely garbage-less in the sense that it has no extra input/output lines other than true input/output lines (hence, it is a “closed” circuit except the true input/output).*

We show a construction method of a circuit  $C$  by an example. First, we consider a circuit module composed of  $n + 1$  REs shown in Fig. 7. It is called an *RE column*. In a resting state, each RE in the RE column is in the state  $V$  except the bottom one indicated by  $x$ . If  $x$  is in the state  $H$ , the RE column is called in the *marked* state, otherwise *unmarked* state. It can be regarded as if it is a 2-state RSM with  $2n$  input symbols  $\{l_1, \dots, l_n, r_1, \dots, r_n\}$  and  $2n$  output symbols  $\{l'_1, \dots, l'_n, r'_1, \dots, r'_n\}$ , though it has many transient states. Its “macroscopic” move function is shown in Table 2.

We now consider an example of an RSM  $M_1 = (\{q_1, q_2, q_3\}, \{a_1, a_2\}, \{b_1, b_2\}, \delta_1)$ , where its move function  $\delta_1$  is shown in Table 3. Prepare three RE columns with  $n = 2$ , each of which corresponds to each state of  $M_1$ , and connect them as shown in Fig. 8. If  $M_1$  is in the state  $q_1$ , then only the first RE column is set to the marked state. When an

**Reversible Computing, Table 3** The move function  $\delta_1$  of an RSM  $M_1$ 

State	Input	
	$a_1$	$a_2$
$q_1$	$q_2 b_1$	$q_3 b_2$
$q_2$	$q_2 b_2$	$q_1 b_1$
$q_3$	$q_1 b_2$	$q_3 b_1$

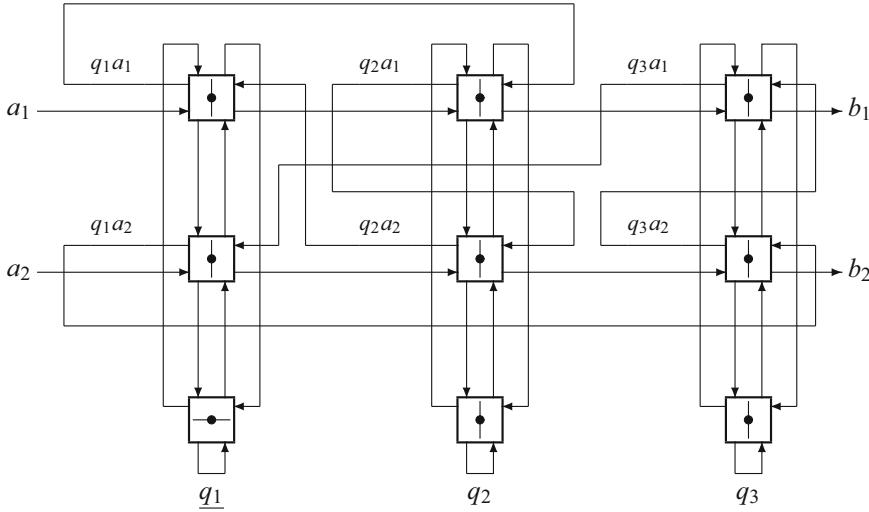
input signal comes from, e.g., the input line  $a_2$ , then the signal first goes out from the line  $l'_2$  of the first RE column, setting this column to the unmarked state. Then, this signal enters the third column from the line  $r_2$ . This makes the third RE column marked and finally goes out from the output line  $b_2$ , which realizes  $\delta_1(q_1, a_2) = (q_3, b_2)$ .

**Universality of 2-State RLEM**s

There are infinitely many 2-state RLEMs besides RE, if we do not restrict the number of input/output symbols. Hence, there will be many universal RLEMs. Surprisingly, it has been shown that nondegenerate 2-state RLEMs except only four are all universal (Morita et al. 2012).

First, we classify 2-state RLEMs. We can see the total number of 2-state  $k$ -symbol RLEMs is  $(2k)!$ , and they are numbered from 0 to  $(2k)! - 1$  in some lexicographic order (Morita et al. 2005). To indicate that it is a  $k$ -symbol RLEM, the prefix “ $k$ -” is attached to its serial number like RLEM 4–289. Here, we use a pictorial representation for 2-state RLEM. Consider, as an example, 2-state 4-symbol RLEM  $M_{4-289} = (\{0, 1\}, \{a_1, a_2, a_3, a_4\}, \{s_1, s_2, s_3, s_4\}, \delta_{4-289})$ . Its move function  $\delta_{4-289}$  is given in Table 4. Then, it is represented by Fig. 9, where the two states 0 and 1 are represented by two boxes. Solid and dotted lines in each box describe the input-output relation for each state. A solid line shows that the state goes to the other, and a dotted line shows the state remains unchanged. For example, if the RLEM 4–289 receives an input symbol  $a_3$  in the state 0, then it gives the output  $s_1$  and goes to the state 1. As in the case of RE, we interpret that each input/output symbol represents an occurrence of a particle at the corresponding input/output port.

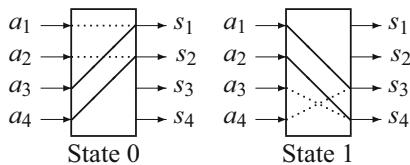
We regard two RLEMs are *equivalent* if one can be obtained by renaming the states and/or



**Reversible Computing, Fig. 8** A garbage-less circuit made of REs that simulates  $M_1$

**Reversible Computing, Table 4** The move function  $\delta_{4-289}$  of the 2-state RLEM 4-289

Present state	Input			
	$a_1$	$a_2$	$a_3$	$a_4$
State 0	0 $s_1$	0 $s_2$	1 $s_1$	1 $s_2$
State 1	0 $s_3$	0 $s_4$	1 $s_4$	1 $s_3$



**Reversible Computing, Fig. 9** Pictorial representation of 2-state RLEM 4-289, which is equivalent to RE

the input/output symbols of the other. For example, RE and RLEM 4-289 are equivalent. It is verified by the following correspondence:  $H \mapsto 0$ ,  $V \mapsto 1$ ,  $n \mapsto a_3$ ,  $s \mapsto a_4$ ,  $e \mapsto a_1$ ,  $w \mapsto a_2$ ,  $n' \mapsto s_3$ ,  $s' \mapsto s_4$ ,  $e' \mapsto s_2$ ,  $w' \mapsto s_1$ .

It has been shown that the numbers of equivalence classes of 2-state 2-, 3-, and 4-symbol RLEMs are 8, 24, and 82, respectively (Morita et al. 2005). Figure 10 shows all representative RLEMs in the equivalence classes of 2- and

3-symbol RLEMs. The representatives are so chosen that it has the smallest number in the class.

Among  $k$ -symbol RLEMs, there are *degenerate* ones, each of which is either equivalent to simple connecting wires (e.g., RLEM 3-3) or equivalent to a  $k'$ -symbol RLEM such that  $k' < k$  (e.g., RLEM 3-6). Its precise definition is found in (Morita et al. 2012). In Fig. 10, they are indicated by “eq. to wires” or “eq. to 2- $n$ .” Thus, *nondegenerate*  $k$ -symbol RLEMs are the main concern of the study. It is known that the numbers of nondegenerate 2-, 3-, and 4-symbol RLEMs are 4, 14, and 55, respectively.

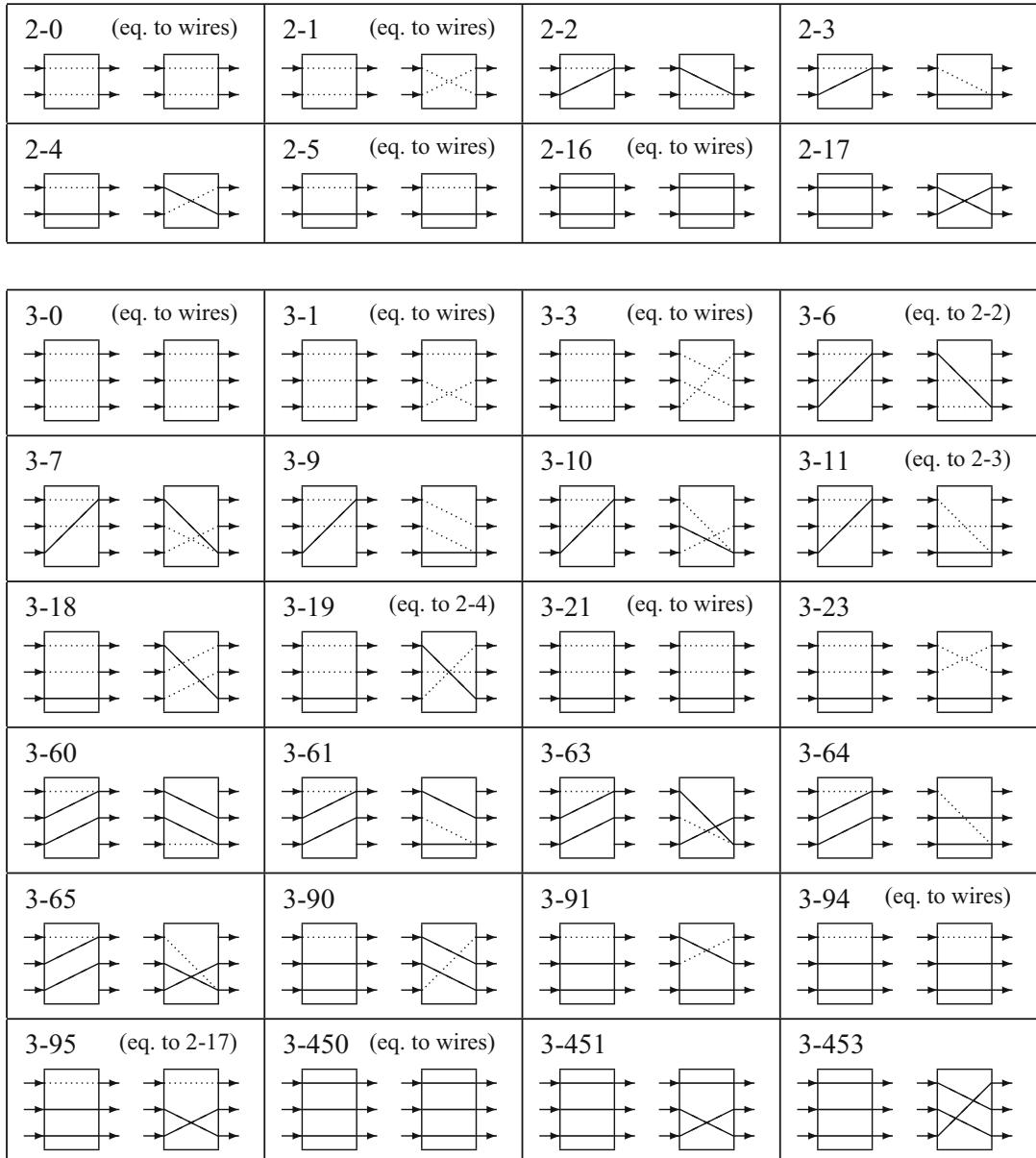
It has been shown that the following three lemmas hold.

**Lemma 1** (Lee et al. 2008; Morita et al. 2012) *RE can be composed of RLEM 3–10.*

**Lemma 2** (Lee et al. 2008) *RLEM 3–10 can be composed of RLEMs 2–3 and 2–4.*

**Lemma 3** (Morita et al. 2012) *RLEMs 2–3 and 2–4 can be composed of any one of 14 nondegenerate 3-symbol RLEMs.*

By above, we obtain the next lemma that entails universality of all nondegenerate 3-symbol RLEMs.



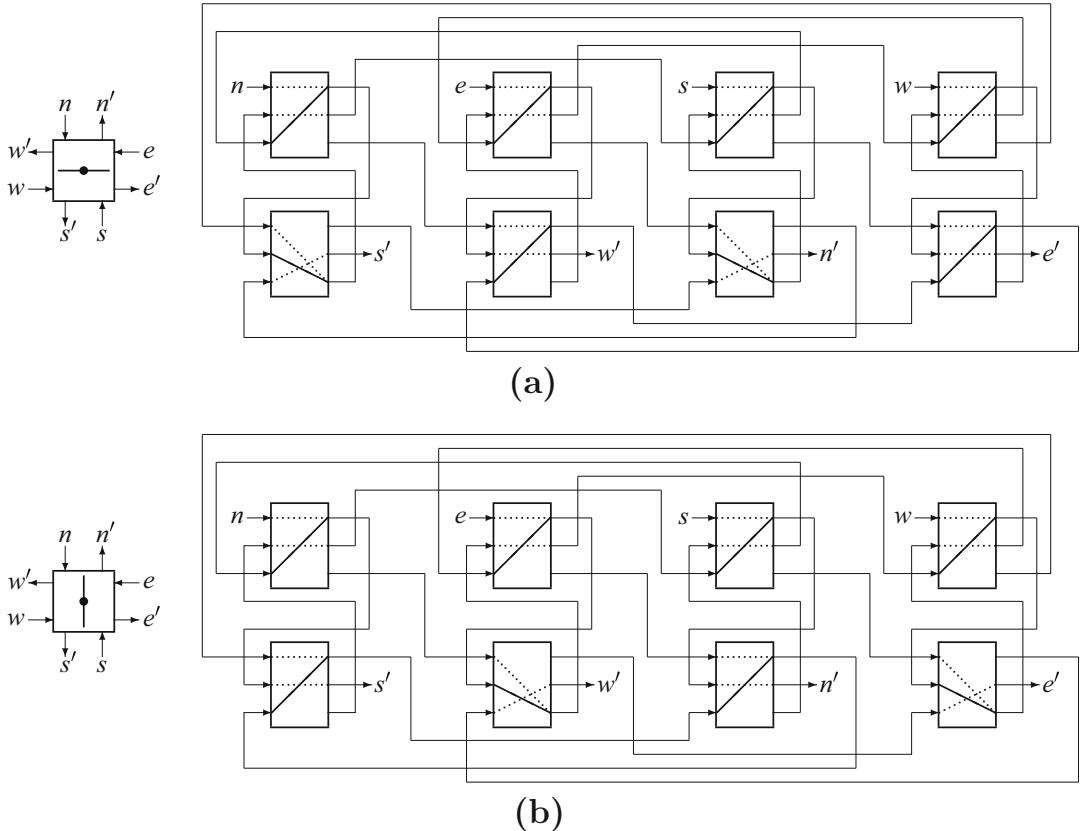
**Reversible Computing, Fig. 10** Representatives of 8 equivalence classes of 24 2-symbol RLEMs (*top*), and those of 24 equivalence classes of 720 3-symbol RLEMs (*bottom*) (Morita et al. 2005). The indications “eq. to wires” and “eq. to 2-*n*” mean it is equivalent to connecting

wires, and it is equivalent to RLEM 2-*n*, respectively. Thus they are degenerate ones. The total numbers of 2- and 3-symbol nondegenerate RLEMs are 4 and 14, respectively

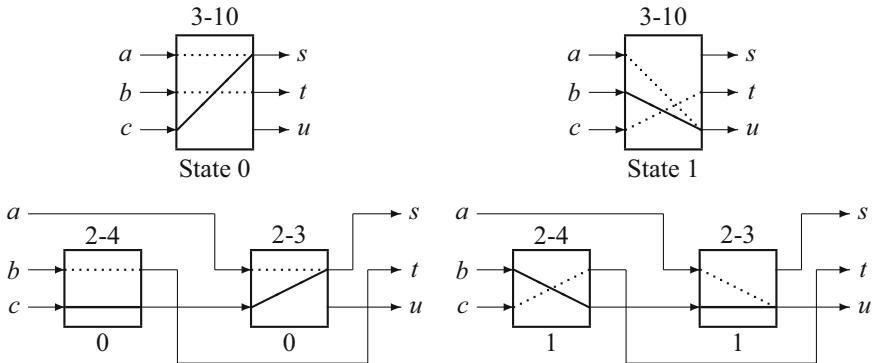
**Lemma 4** (Morita et al. 2012) *RE can be constructed by any one of 14 nondegenerate 3-symbol RLEMs.*

Lemmas 1, 2, and 3 are proved by designing circuits composed of given RLEMs which

correctly simulate the target RLEMs. These circuits are shown in Figs. 11, 12, and 13. Lemma 1 is proved by a circuit made of RLEMs 3–10 that simulates an RE, which was first given in (Lee et al. 2008). Later, a simpler circuit was



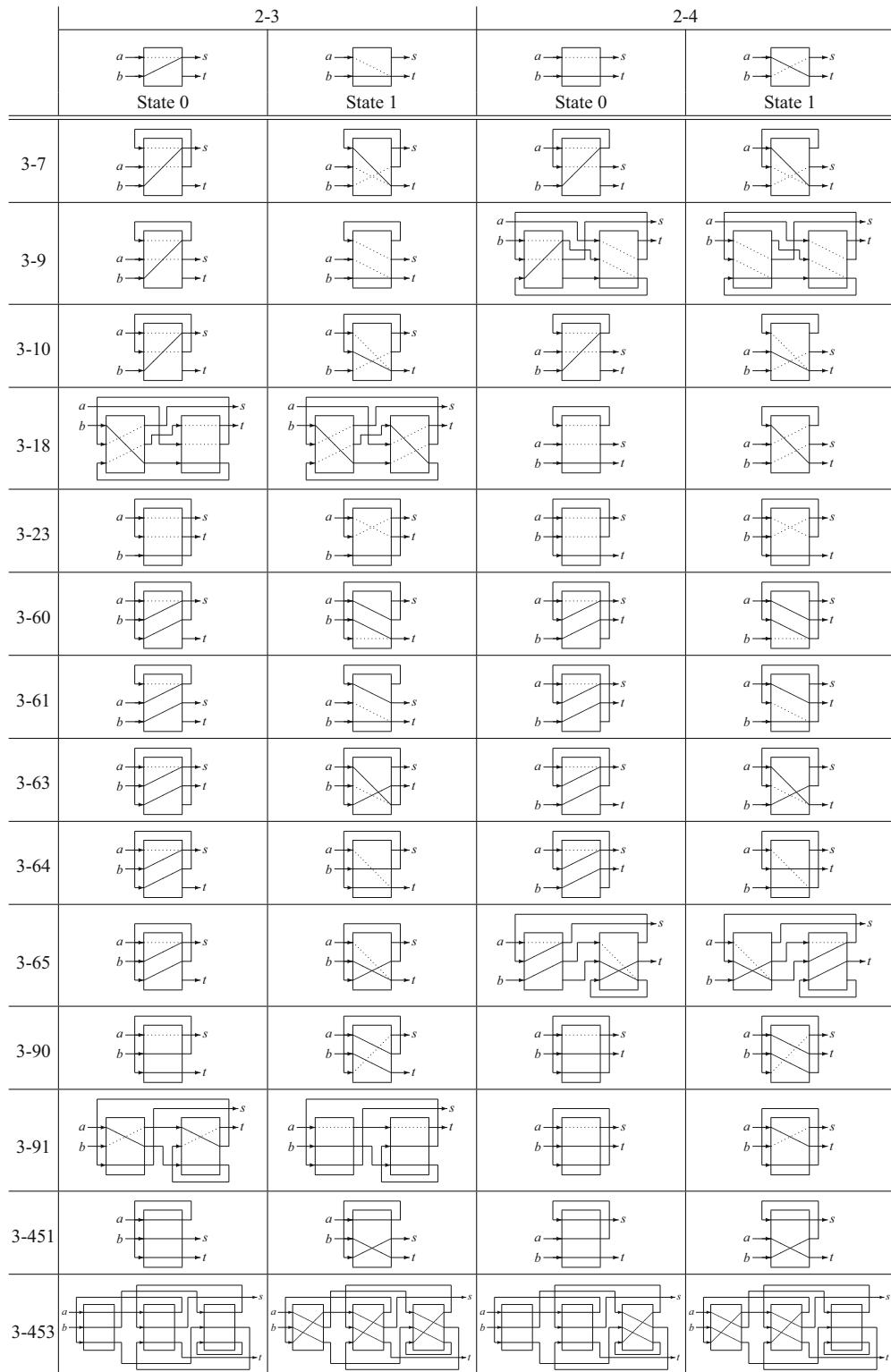
**Reversible Computing, Fig. 11** A circuit composed of RLEM 3–10 that simulates RE (Morita et al. 2012). (a, b) correspond to the states H and V of RE, respectively



**Reversible Computing, Fig. 12** A circuit composed of RLEM 2–3 and 2–4 that simulates RLEM 3–10 (Lee et al. 2008). The lower left and the lower right figures correspond to the states 0 and 1 of RLEM 3–10, respectively

given in (Morita et al. 2012), which is shown in Fig. 11. Next, Lemma 2 is proved by a circuit made of RLEM 2–3 and 2–4 that simulates RLEM 3–10 shown in Fig. 12 (Lee et al. 2008).

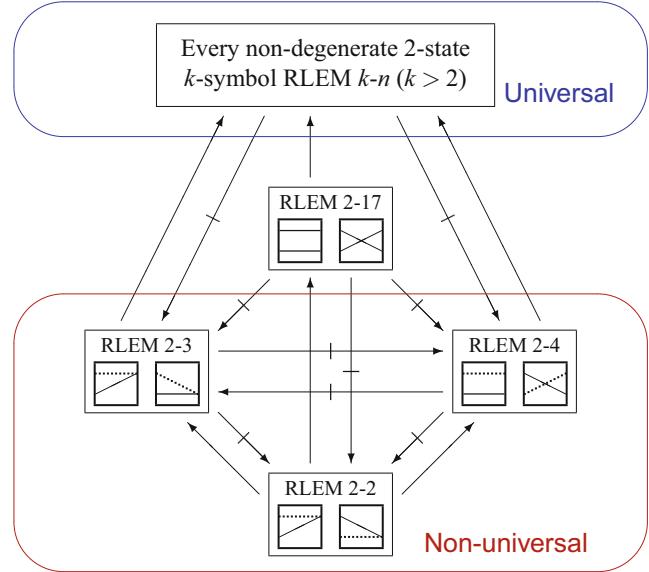
Finally, Lemma 3 is proved by 28 circuits composed of each of 14 nondegenerate 3-symbol RLEM 3–10 that simulate RLEM 2–3 and 2–4 as in Fig. 13.



**Reversible Computing, Fig. 13** Circuits composed of each of 14 nondegenerate 3-symbol RLEMs that simulate RLEMs 2-3 and 2-4 (Morita et al. 2012)

### Reversible Computing,

**Fig. 14** A hierarchy among 2-state RLEM s. Here,  $A \rightarrow B$  ( $A \not\rightarrow B$ , respectively) represents that  $A$  can (cannot) be simulated by  $B$



The following lemma gives a relation between  $k$ -symbol RLEM s and  $(k - 1)$ -symbol RLEM s. Its proof is found in Morita et al. (2012).

**Lemma 5** (Morita et al. 2012) *Let  $M_k$  be an arbitrary nondegenerate  $k$ -symbol RLEM ( $k > 2$ ). Then, there exists a nondegenerate  $(k - 1)$ -symbol RLEM  $M_{k-1}$  that can be simulated by  $M_k$ .*

By Theorem 3 and Lemmas 4 and 5, we have the next theorem stating that almost all nondegenerate 2-state RLEM s are universal. Note that universal RLEM s can simulate each other.

**Theorem 4** (Morita et al. 2012) *Every nondegenerate 2-state  $k$ -symbol RLEM is universal if  $k > 2$ .*

On the other hand, there are four nondegenerate 2-state 2-symbol RLEM s 2–2, 2–3, 2–4, and 2–17 (Fig. 10). So far, nonuniversality of RLEM s 2–2, 2–3, and 2–4 has been shown in Mukai et al. (2014).

**Theorem 5** (Mukai et al. 2014) *RLEM s 2–2, 2–3, and 2–4 are nonuniversal.*

The following lemma says RLEM 2–2 is the weakest one among nondegenerate 2-state RLEM s.

**Lemma 6** (Mukai et al. 2014) *RLEM 2–2 can be simulated by any one of RLEM s 2–3, 2–4, and 2–17.*

Figure 14 summarizes the above results. It is not known whether RLEM 2–17 is universal or not. On the other hand, it is shown that any two combinations among RLEM s 2–3, 2–4, and 2–17 are universal (Lee et al. 2008; Mukai et al. 2014).

We finally mention the following relation between RSMs and reversible logic gates.

**Theorem 6** (Morita 1990) *For any RSM  $M$ , we can construct a completely garbage-less logic circuit composed of Fredkin gates and delay elements that simulates  $M$ .*

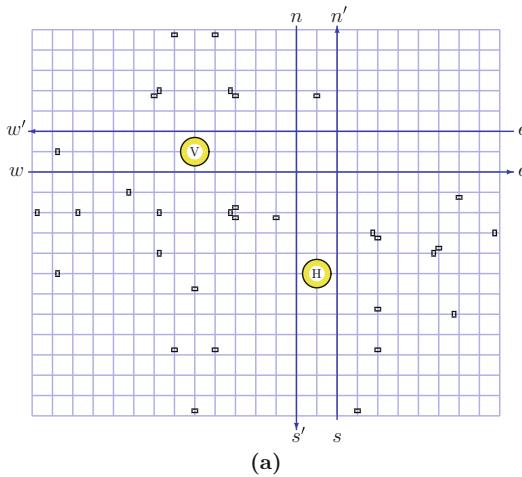
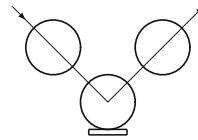
### Billiard Ball Model (BBM)

Fredkin and Toffoli (1982) proposed an interesting reversible physical model of computing called the *billiard ball model* (BBM). It consists of idealized balls and reflectors (Fig. 15), and balls can collide with other balls or reflectors. We assume collisions are elastic, and there is no friction. Hence, there is no energy dissipation inside of this model. Fredkin and Toffoli showed that Fredkin gate is realizable in BBM. Therefore, reversible computing systems can be embedded in the space of BBM.

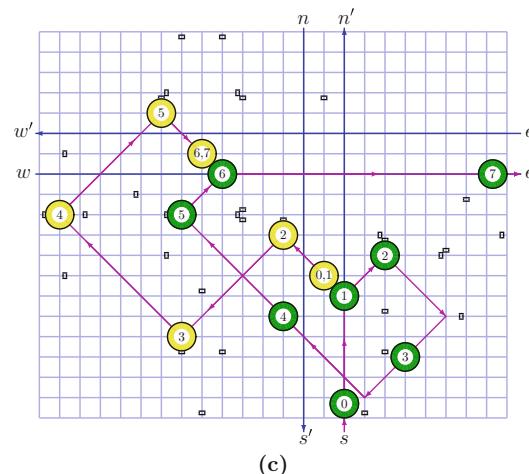
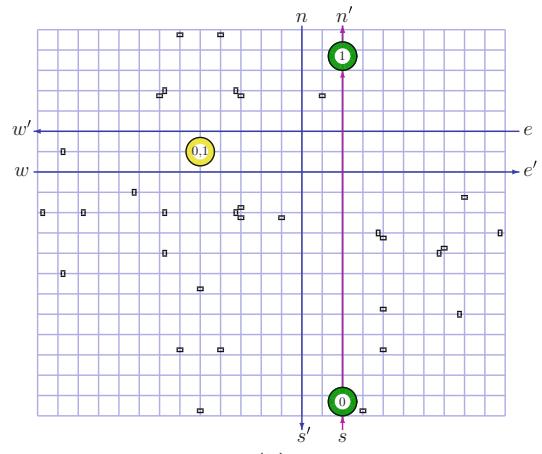
It is shown that RE can be simulated directly in BBM as in Fig. 16 (Morita 2008a, 2012). It consists of one stationary ball called a *state ball* and many reflectors indicated by small rectangles. A state ball is placed at the position of H or V in Fig. 16a

depending on the state of the simulated RE. A moving ball called a *signal ball* can be given to any one of the input lines  $n$ ,  $e$ ,  $s$ , and  $w$ . Then, the operation of an RE is simulated by collisions of balls and reflectors. Figure 16b shows the case where the state of RE is V and the input is  $s$  (corresponding to Fig. 6a). In this case, the state ball is at the position V, and a signal ball is given to the input

**Reversible Computing,**  
**Fig. 15** A ball and a  
reflector in the billiard ball  
model (BBM)



port  $s$  with any speed. This is a trivial case, i.e., the signal ball simply moves from the position 0 to 1 without interacting with reflectors and the state ball. Hence, the transition  $\delta_{\text{RE}}(V, s) = (V, n')$  is correctly simulated. Figure 16c shows the case where the state of RE is H and the input is  $s$  (corresponding to Fig. 6b). In this case, the signal ball given to the input port  $s$  collides with the state ball at the position 1. Then, the two balls move along the paths shown in the figure. Each ball is reflected several times by the reflectors on each path. Finally, the two balls collide again at the positions indicated by 6. The state ball stops at the position, and the signal ball moves eastward. Thus, the transition  $\delta_{\text{RE}}(H, s) = (V, e')$  is simulated.



**Reversible Computing, Fig. 16** (a) Rotary element (RE) realized in BBM, where small rectangles are reflectors, and H and V show possible positions of a state ball for

the states H and V. (b) The case  $\delta_{\text{RE}}(V, s) = (V, n')$ . (c) The case  $\delta_{\text{RE}}(H, s) = (V, e')$

An advantage of this method is that, in an idle state, the state ball is stationary. Hence, a signal ball can be given at any moment and with any speed. Thus, there is no need of synchronizing two balls. In Mukai and Morita (2012), it is also shown that any  $m$ -state  $k$ -symbol RLEM can be realized in BBM in a systematic way when  $k \leq 4$ .

## Reversible Turing Machines

A reversible Turing machine (RTM) is a standard model in reversible computing as in the case of traditional (i.e., irreversible) computation theory. The notion of an RTM first appeared in the paper of Lecerf (1963), where unsolvability of the halting problem and some related problems on RTMs is shown. Bennett (1973) showed that every irreversible Turing machine (TM) can be simulated by an RTM with three tapes without leaving garbage information on the tapes when it halts. He also pointed out the significance of RTMs, since they are closely related to physical reversibility and the problem of energy dissipation in computing process.

In this section, after giving definitions on RTMs, we explain the method of Bennett for converting a given irreversible TM into an equivalent three-tape RTM. We then discuss the problems how RTMs can be built by reversible logic elements and how a small universal RTM can be constructed.

### Definitions on Reversible Turing Machines (RTMs)

We give two formulations for TMs, i.e., the quadruple formulation and the quintuple formulation. The quadruple formulation introduced by Bennett (1973) makes it easy to define an “inverse” TM for a given RTM. The inverse RTM, which “undoes” the computation performed by the original RTM, plays a key role for garbage-less computation. On the other hand, the quintuple formulation is also useful, since TMs can be concisely described by it. In addition, most classical TMs are defined in quadruple form, and thus it makes convenient to compare with them.

**Definition 5** A one-tape Turing machine in the quadruple form is defined by

$$T = (Q, S, q_0, F, s_0, \delta),$$

where  $Q$  is a nonempty finite set of states,  $S$  is a nonempty finite set of symbols,  $q_0$  is an initial state ( $q_0 \in Q$ ),  $F$  is a set of final (halting) states ( $F \subseteq Q$ ),  $s_0$  is a special blank symbol ( $s_0 \in S$ ), and  $\delta$  is a move relation, which is a subset of  $(Q \times S \times S \times Q) \cup (Q \times \{/ \} \times \{-, 0, +\} \times Q)$ . Each element of  $\delta$  is a quadruple, and either of the form  $[p, s, s', q] \in (Q \times S \times S \times Q)$  or  $[p, /, d, q] \in (Q \times \{/ \} \times \{-, 0, +\} \times Q)$ . The symbols “ $-$ ,” “ $0$ ,” and “ $+$ ” stand for “left-shift,” “zero-shift,” and “right-shift,” respectively. The quadruple  $[p, s, s', q]$  means if  $T$  reads the symbol  $s$  in the state  $p$ , then write  $s'$  and go to the state  $q$ . The quadruple  $[p, /, d, q]$  means if  $T$  is in  $p$ , then shift the head to the direction  $d$  and go to the state  $q$ .

Determinism and reversibility of a TM are defined symmetrically as given in the next definition. Hence, we can regard a reversible TM as a “backward deterministic” TM.

**Definition 6** Let  $T = (Q, S, q_0, F, s_0, \delta)$  be a TM in the quadruple form. It is called a deterministic Turing machine if the following condition holds for any pair of distinct quadruples  $[p_1, b_1, c_1, q_1]$  and  $[p_2, b_2, c_2, q_2]$  in  $\delta$ .

If  $p_1 = p_2$ , then  $b_1 \neq / \wedge b_2 \neq / \wedge b_1 \neq b_2$ .

On the other hand,  $T$  is called a reversible Turing machine (RTM) if the following condition holds for any pair of distinct quadruples  $[p_1, b_1, c_1, q_1]$  and  $[p_2, b_2, c_2, q_2]$  in  $\delta$ .

If  $q_1 = q_2$ , then  $b_1 \neq / \wedge b_2 \neq / \wedge c_1 \neq c_2$ .

Note that, in what follows, we consider only deterministic Turing machines. So, we omit the word “deterministic” henceforth.

It is easy to extend the above definition to multi-tape TMs. For example, a two-tape TM is defined by

$$T = (Q, (S_1, S_2), q_0, F, (s_{1,0}, s_{2,0}), \delta).$$

A quadruple in  $\delta$  is either of the form  $[p, [s_1, s_2], [s'_1, s'_2], q] \in (Q \times (S_1 \times S_2) \times (S_1 \times S_2) \times Q)$  or of the form  $[p, /, [d_1, d_2], q] \in (Q \times \{/ \} \times \{-, 0, +\}^2 \times Q)$ . Determinism and reversibility are defined similarly as above. Namely,  $T$  is reversible if the following condition holds for any pair of distinct quadruples  $[p_1, X_1, [b_1, b_2], q_1]$  and  $[p_2, X_2, [c_1, c_2], q_2]$  in  $\delta$ .

If  $q_1 = q_2$ , then  $X_1 \neq / \wedge X_2 \neq / \wedge [b_1, b_2] \neq [c_1, c_2]$ .

Next, we define a TM in the quintuple form.

**Definition 7** A one-tape Turing machine in the quintuple form is defined by

$$T = (Q, S, q_0, F, s_0, \delta),$$

where  $Q, S, q_0, F, s_0$  are the same as in Definition 5. The move relation  $\delta$  is a subset of  $(Q \times S \times S \times \{-, 0, +\} \times Q)$ . Each element of  $\delta$  is a quintuple of the form  $[p, s, s', d, q]$ . It means if  $T$  reads the symbol  $s$  in the state  $p$ , then write  $s'$ , shift the head to the direction  $d$ , and go to the state  $q$ .

**Definition 8** Let  $T = (Q, S, q_0, F, s_0, \delta)$  be a TM in the quintuple form. We say  $T$  is deterministic if the following condition holds for any pair of distinct quintuples  $[p_1, s_1, s'_1, d_1, q_1]$  and  $[p_2, s_2, s'_2, d_2, q_2]$  in  $\delta$ .

If  $p_1 = p_2$ , then  $s_1 \neq s_2$ .

We say  $T$  is reversible if the following condition holds for any pair of distinct quintuples  $[p_1, s_1, s'_1, d_1, q_1]$  and  $[p_2, s_2, s'_2, d_2, q_2]$  in  $\delta$ .

If  $q_1 = q_2$ , then  $s'_1 \neq s'_2 \wedge d_1 = d_2$ .

**Proposition 1** For any RTM  $T_5$  in the quintuple form, there is an RTM  $T_4$  in the quadruple form that simulates each step of the former in two steps.

**Proof** Let  $T_5 = (Q, S, q_0, F, s_0, \delta)$ . We define  $T_4 = (Q', S, q_0, F, s_0, \delta')$  as follows. Let  $Q' = Q \cup \{q'|q \in Q\}$ . The set  $\delta'$  is given by the next procedure. First, set the initial value of  $\delta'$  to the empty set. Next, for each  $q \in Q$  do the following operation. Let  $[p_1, s_1, s'_1, d_1, q], [p_1, s_2, s'_2, d_2, q], \dots, [p_m, s_m, s'_m, d_m, q]$  be all the quintuples in  $\delta$  whose fifth element is  $q$ . Note that  $d_1 = d_2 = \dots = d_m$  holds and  $s'_1, s'_2, \dots, s'_m$  are pairwise distinct, because  $T_5$  is reversible. Then, include the  $m+1$  quadruples  $[p_1, s_1, s'_1, q'], [p_1, s_2, s'_2, q'], \dots, [p_m, s_m, s'_m, q']$  and  $[q', /, d_1, q]$  in  $\delta'$ . It is easy to see that  $T_4$  has the required property (Q.E.D.).

By Proposition 1, we see the definition of an RTM in the quintuple form is compatible with that in the quadruple form. The converse of Proposition 1 is also easy to show. It is left for readers as an exercise.

### Universality of RTMs and Garbage-Less Computation

We now discuss computational universality of RTMs. It is actually easy to convert an irreversible TM to an RTM, because we can construct an RTM that simulates the former and records all the information which quadruples were executed by the irreversible TM by using a special history tape. But, by this method, a large amount of garbage information will be left at the end of a computation. An important point is that such a garbage can be reversibly erased. Hence, a garbage-less computation is possible as shown in the next theorem.

**Theorem 7** (Bennett 1973) For any (irreversible) one-tape TM, we can construct a three-tape RTM that simulates the former and leaves only an input string and an output string on the tapes when it halts (hence, it leaves no garbage information).

**Proof** Let  $T = (Q, S, q_0, \{q_f\}, 0, \delta)$  be a given irreversible TM in the quadruple form. We assume  $T$  satisfies the following conditions (it is easy to modify a given TM so that it meets the conditions).

- (i) It has a rightward-infinite tape whose leftmost square always keeps the blank symbol.
- (ii) When  $T$  halts in the state  $q_f$ , it does so at the leftmost symbol.
- (iii) The output is given from the second square of the tape to the right when  $T$  halts.
- (iv) The output string does not contain blank symbols.
- (v) The initial state  $q_0$  does not appear as the fourth element of any quadruple.
- (vi) There is only one quadruple in  $\delta$  whose fourth element is  $q_f$ .

Let  $m$  be the total number of quadruples contained in  $\delta$ , and we assume the numbers  $1, \dots, m$  are assigned uniquely to these quadruples.

An RTM  $T^R$  that simulates  $T$  has three tapes: a working tape (for simulating the tape of  $T$ ), a history tape (for recording the movement of  $T$  at each step), and an output tape (for writing an output string). We assume  $Q = \{q_0, q_1, \dots, q_f\}$ . It is defined as follows:

$$\begin{aligned} T^R &= (Q', (S, \{0, 1, \dots, m\}, S), q_0, \{p_0\}, (0, 0, 0), \delta') \\ Q' &= \{q_0, q_1, \dots, q_f\} \cup \{q'_0, q'_1, \dots, q'_f\} \cup \{c_1, c'_1, c_2, c'_2\} \\ &\quad \cup \{p_0, p_1, \dots, p_f\} \cup \{p'_0, p'_1, \dots, p'_f\} \end{aligned}$$

Computation of  $T^R$  has three stages: (1) the forward computation stage, (2) the copying stage, and (3) the backward computation stage. The set  $\delta'$  of quadruples is defined as the union  $\delta_1 \cup \delta_2 \cup \delta_3$  of the sets of quadruples given below. We assume when  $T^R$  starts to move, an input string is written in the working tape, and the history and the output tapes contain only blank symbols.

1. The quadruple set  $\delta_1$  for the forward computation stage:

When  $T^R$  simulates  $T$  step by step, it records on the history tape which quadruple of  $T$  was used. By this operation,  $T^R$  can be reversible. This is realized by giving the quadruple set  $\delta_1$  as follows.

- (i) If the  $h$ -th quadruple ( $h = 1, \dots, m$ ) of  $T$  is

$$[q_i, s_j, s_k, q_l] \quad (q_i, q_l \in Q, s_j, s_k \in S),$$

then include the following quadruples in  $\delta_1$ .

$$\begin{aligned} [q_i, /, [0, +, 0], q'_i] \\ [q'_i, [s_j, 0, 0], [s_k, h, 0], q_l] \end{aligned}$$

- (ii) If the  $h$ -th quadruple ( $h = 1, \dots, m$ ) of  $T$  is

$$[q_i, /, d, q_l] \quad (q_i, q_l \in Q, d \in \{-, 0, +\})$$

then include the following quadruples in  $\delta_1$ .

$$\begin{aligned} [q_i, /, [d, +, 0], q'_i] \\ [q'_i, [x, 0, 0], [x, h, 0], q_l] \quad (x \in S) \end{aligned}$$

2. The quadruple set  $\delta_2$  for the copying stage:

If the forward computation stage is completed, and  $T^R$  becomes in the state  $q_f$ , then it copies the output string on the working tape to the output tape. This is performed by giving the quadruple set  $\delta_2$  as below. Here let  $n$  be the identification number of the quadruple that contains  $q_f$  as the fourth element.

$$\begin{aligned} [q_f[0, n, 0], [0, n, 0], c_1] \\ [c_1, /, [+0, +], c_2] \\ [c_2, [y, n, 0], [y, n, y], c_1] \quad (y \in S - \{0\}) \\ [c_2, [0, n, 0], [0, n, 0], c'_2] \\ [c'_1, [y, n, y], [y, n, y], c'_2] \quad (y \in S - \{0\}) \\ [c'_2, /, [-0, -], c'_1] \\ [c'_1, [0, n, 0], [0, n, 0], p_f] \end{aligned}$$

3. The quadruple set  $\delta_3$  for the backward computation stage:

After the copying process, the backward computation stage starts in order to reversibly erase garbage information left on the history tape. This is performed by an inverse TM of the forward computing. The quadruple set  $\delta_3$  is as follows.

(i) If the  $h$ -th quadruple ( $h = 1, \dots, m$ ) of  $T$  is

$$[q_i, s_j, s_k, q_l] \quad (q_i, q_l \in Q, s_j, s_k \in S)$$

then include the following quadruples in  $\delta_3$ .

$$\begin{aligned} & [p'_i, /, [0, -, 0], p_i] \\ & [p_l, [s_k, h, 0], [s_j, 0, 0], p'_i] \end{aligned}$$

(ii) If the  $h$ -th quadruple ( $h = 1, \dots, m$ ) of  $T$  is

$$[q_i, /, d, q_l] \quad (q_i, q_l \in Q, d \in \{-, 0, +\})$$

then include the following quadruples in  $\delta_3$ . Note that if  $d = -$  ( $0, +$ , respectively), then  $d' = +$  ( $0, -$ ).

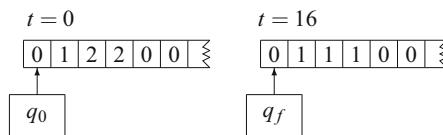
$$\begin{aligned} & [p'_i, /, [d', -, 0], p_i] \\ & [p_l, [x, h, 0], [x, 0, 0], p'_i] \quad (x \in S) \end{aligned}$$

It is easy to see that  $T^R$  simulates  $T$  correctly. Reversibility of  $T^R$  is verified by checking the quadruple set  $\delta'$ . In particular, the set  $\delta_3$  of quadruples is reversible since  $T$  is deterministic (Q.E.D.).

**Example 1** We consider the following irreversible TM  $T_{\text{erase}}$  and convert it into an RTM  $T_{\text{erase}}^R$  by the method of Theorem 7.

$$\begin{aligned} T_{\text{erase}} &= (\{q_0, q_1, q_2, q_3, q_4, q_f\}, \{0, 1, 2\}, q_0, \{q_f\}, 0, \delta_{\text{erase}}) \\ \delta_{\text{erase}} &= \{1: [q_0, /, +, q_1], 2: [q_1, 0, 0, q_4], 3: [q_1, 1, 1, q_2], 4: [q_1, 2, 1, q_2], \\ & 5: [q_2, /, +, q_1], 6: [q_3, 0, 0, q_f], 7: [q_3, 1, 1, q_4], 8: [q_4, /, -, q_3]\} \end{aligned}$$

When given an input string consisting of 1s and 2s,  $T_{\text{erase}}$  rewrites all the occurrences of 2s into 1s and halts (see Fig. 17). The set of quadruples of the RTM  $T_{\text{erase}}^R$  that simulates  $T_{\text{erase}}$  is given in Fig.



**Reversible Computing, Fig. 17** The initial and the final computational configuration of  $T_{\text{erase}}$  for the given input string 122

18, and its computation process for the input 122 is shown in Fig. 19.

It is also possible to convert an arbitrary (irreversible) TM into an equivalent one-tape 2-symbol RTM (Morita 2015b; Morita et al. 1989).

There are studies related to computational complexity of RTMs (Bennett 1989; Buhrman et al. 2001; Jacopini et al. 1990; Lange et al. 2000; Morita 2014a). Bennett (1989) showed the following space-time trade-off theorem for RTMs, where an interesting method of reducing space is given. Note that the RTM in Theorem 7 uses large amount of space, which is proportional to the time the simulated TM uses.

**Theorem 8** (Bennett 1989) For any  $\varepsilon > 0$  and any (irreversible) TM that uses time  $T$  and space  $S$ , we can construct an RTM that simulates the former using time  $O(T^{1+\varepsilon})$  and space  $O(S \log T)$ .

On the other hand, Lange et al. (2000) showed that language recognition capability of (irreversible) TMs with space bound  $S$  is exactly the same as that of RTMs with the same space bound.

**Theorem 9** (Lange et al. 2000) For any (irreversible) TM that uses space  $S$ , we can construct an RTM that simulates the former using space  $O(S)$ .

In Morita (2014a), equivalence of non-deterministic reversible TMs and deterministic reversible TMs and a simplified proof of Theorem 9 are shown.

### Constructing RTMs by REs

It is possible to construct a garbage-less reversible logic circuit that simulates a given RTM. We have already seen that any RSM can be realized as a garbage-less reversible RE circuit (Theorem 3). In Morita (2001), it is shown that an RTM can be decomposed into tape cell modules and a finite control module, each of which is an RSM. Therefore, any RTM can be composed of REs as a garbage-less circuit.

Figure 20 shows an example of an RTM  $T_{\text{parity}}$  composed only of REs (Morita 2001, 2015a).

**Reversible Computing,**  
**Fig. 18** The quadruple set  
of the RTM  $T_{\text{erase}}^R$

$[q_0, /, [+ +, 0], q'_0]$	$[p'_0, /, [ -, -, 0], p_0]$
$[q'_0, [x, 0, 0], [x, 1, 0], q_1]$	$[p_1, [x, 1, 0], [x, 0, 0], p'_0]$
$[q_1, /, [0, +, 0], q'_1]$	$[p'_1, /, [0, -, 0], p_1]$
$[q'_1, [0, 0, 0], [0, 2, 0], q_4]$	$[p_4, [0, 2, 0], [0, 0, 0], p'_1]$
$[q'_1, [1, 0, 0], [1, 3, 0], q_2]$	$[p_2, [1, 3, 0], [1, 0, 0], p'_1]$
$[q'_1, [2, 0, 0], [1, 4, 0], q_2]$	$[p_2, [1, 4, 0], [2, 0, 0], p'_1]$
$[q_2, /, [+ +, 0], q'_2]$	$[p'_2, /, [ -, -, 0], p_2]$
$[q'_2, [x, 0, 0], [x, 5, 0], q_1]$	$[p_1, [x, 5, 0], [x, 0, 0], p'_2]$
$[q_3, /, [0, +, 0], q'_3]$	$[p'_3, /, [0, -, 0], p_3]$
$[q'_3, [0, 0, 0], [0, 6, 0], q_f]$	$[p_f, [0, 6, 0], [0, 0, 0], p'_3]$
$[q'_3, [1, 0, 0], [1, 7, 0], q_4]$	$[p_4, [1, 7, 0], [1, 0, 0], p'_3]$
$[q_4, /, [ -, +, 0], q'_4]$	$[p'_4, /, [ +, -, 0], p_4]$
$[q'_4, [x, 0, 0], [x, 8, 0], q_3]$	$[p_3, [x, 8, 0], [x, 0, 0], p'_4]$
$[q_f, [0, 6, 0], [0, 6, 0], c_1]$	$[c'_1, [0, 6, 0], [0, 6, 0], p_f]$
$[c_1, /, [+ 0, +], c_2]$	$[c'_2, /, [ -, 0, -], c'_1]$
$[c_2, [y, 6, 0], [y, 6, y], c_1]$	$[c'_1, [y, 6, y], [y, 6, y], c'_2]$
$[c_2, [0, 6, 0], [0, 6, 0], c'_2]$	

There is a finite control in the left part of this figure. To the right of it, infinitely many copies of tape cell modules are connected. (Here, some ad hoc techniques are used to reduce the number of REs rather than to use the systematic method employed in Theorem 3). Note that this circuit is closed except the Begin, Accept, and Reject ports.

It is a small RTM that tests whether a given unary number is even. It has the set of quintuples  $\{[q_0, 0, 1, +, q_1], [q_1, 0, 1, -, q_{\text{acc}}], [q_1, 1, 0, +, q_2], [q_2, 0, 1, -, q_{\text{rej}}], [q_2, 1, 0, +, q_1]\}$ . The computing process for the input string 0110 is as follows:  $q_00110 \vdash 1 q_1110 \vdash 10 q_210 \vdash 100 q_10 \vdash 10 q_{\text{acc}}01$ . If the RTM accepts (rejects, respectively) the input, a signal goes out from the Accept (Reject) port. Since RE can be implemented in BBM as shown in Fig. 16, the whole circuit that simulates the RTM is also embeddable in BBM.

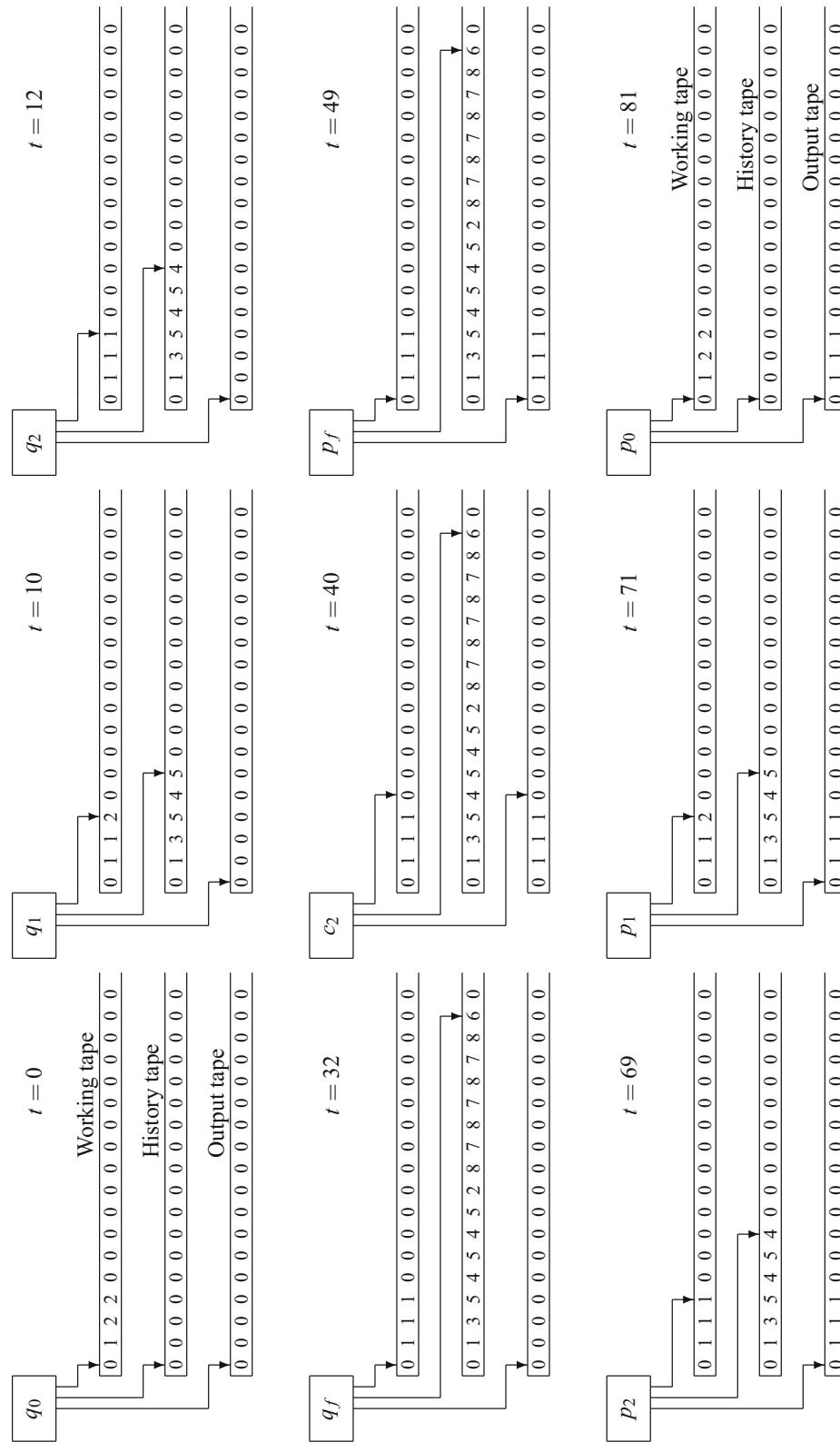
We can also realize RTMs by Fredkin gates and delay elements as a garbage-less circuit using the method of composing RSMs by Fredkin gates shown in Morita (1990). If we do so, however, we need a larger number of Fredkin gates than that of REs. In addition, when designing the circuit, signal delays and timings should be very carefully

adjusted, because signals must be synchronized at each gate. On the other hand, the circuit of Fig. 20 works correctly even if each RE operates asynchronously, since only one signal is moving in this circuit. In an RE, a single moving signal interacts with a rotatable bar, where the latter is thought as a kind of stationary signal. Hence, its operation may be performed at any time, and there is no need of synchronizing signals at each RE.

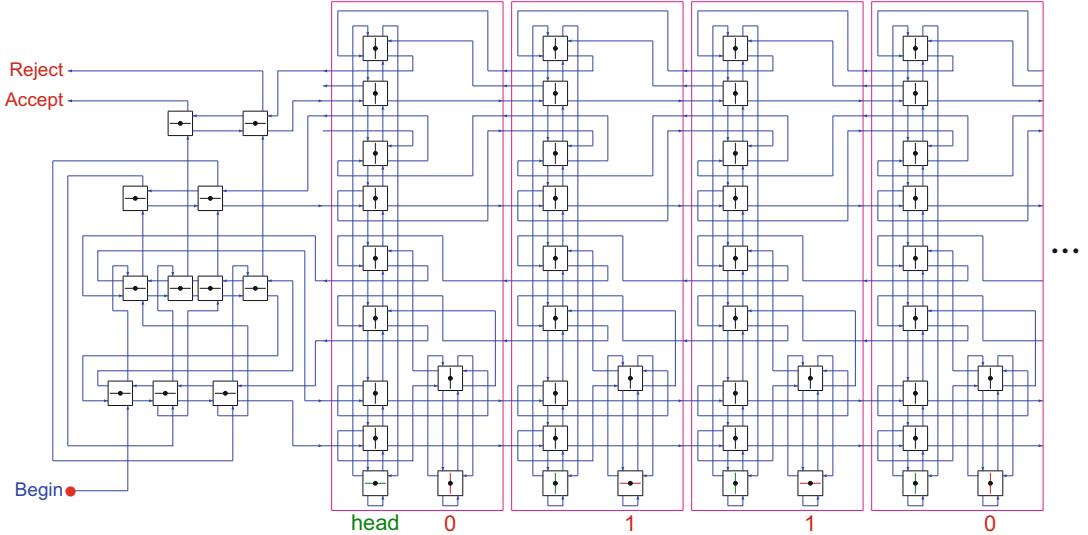
### Universal RTMs

A universal TM (UTM) is one that can simulate any TM, or equivalently, that can compute any recursive function. From Theorem 7, we can see a universal RTM (URTM) exists. However, if we use the technique in the proof of Theorem 7 to convert a UTM into a URTM, the numbers of states and symbols will become very large.

As for classical (i.e., irreversible) UTMs, the following small UTMs have been proposed so far, where UTM( $m, n$ ) denotes an  $m$ -state  $n$ -symbol UTM. They are UTM(7,4) by Minsky (1967); UTM(5,5), UTM(4,6), and UTM(2,18) by Rogozhin (1996); UTM(3,9) by Kudlek and Rogozhin (2002); and UTM(6,4), UTM(9,3), and UTM(15,2) by Neary and Woods (2009). These



**Reversible Computing, Fig. 19** The computing process of the RTM for the input string 122: the forward computation stage (from  $t = 0$  to  $t = 32$ ), the copying stage (from  $t = 49$  to  $t = 81$ ) to  $t = 48$ ), and the backward computation stage (from  $t = 49$  to  $t = 81$ )



**Reversible Computing, Fig. 20** A circuit composed only of REs that simulates the RTM  $T_{\{\text{parity}\}}$  (Morita 2001, 2015a)

small UTMs simulates a 2-tag system (Minsky 1967) or its variant, which is a simple string rewriting system having computational universality.

In the following, we describe a 17-state 5-symbol URTM (URTM(17,5)) by Morita and Yamaguchi (2007). It simulates a cyclic tag system (CTAG) given by Cook (2004), which is a very simple string rewriting system with computational universality. Note that Cook (2004) used CTAGs to show universality of the elementary cellular automaton of rule 110.

Since the notion of halting was not defined explicitly in the original definition of a CTAG, we use here a modified definition of a CTAG with the halting property, which can simulate a two-tag system with the halting property (Morita and Yamaguchi 2007).

**Definition 9** A cyclic tag system (CTAG) is defined by

$$C = (k, \{Y, N\}, (\text{halt}, p_1, \dots, p_{k-1})),$$

where  $k$  ( $k = 1, 2, \dots$ ) is the length of a cycle (i.e., period),  $\{Y, N\}$  is the (fixed) alphabet, and  $(p_1, \dots, p_{k-1}) \in (\{Y, N\}^*)^{*k-1}$  is a  $(k-1)$ -tuple of production rules. A pair  $(v, m)$  is called an instantaneous description (ID) of  $C$ , where  $v \in \{Y, N\}^*$  and  $m \in \{0, \dots, k-1\}$ . The nonnegative

integer  $m$  is called a phase of the ID. A halting ID is an ID of the form  $(Yv, 0)$  ( $v \in \{Y, N\}^*$ ). The transition relation  $\Rightarrow$  is defined as follows. For any  $(v, m), (v', m') \in \{Y, N\}^* \times \{0, \dots, k-1\}$ ,

$$\begin{aligned} (Yv, m) &\Rightarrow (v', m') \text{ if } [m \neq 0] \wedge [m' = m + 1 \bmod k] \wedge [v' = vp_m], \\ (Nv, m) &\Rightarrow (v', m') \text{ if } [m' = m + 1 \bmod k] \wedge [v' = v]. \end{aligned}$$

A sequence of IDs  $((v_0, m_0), \dots, (v_n, m_n))$  is called a complete computation starting from  $v \in \{Y, N\}^*$  if  $(v_0, m_0) = (v, 0)$ ,  $(v_i, m_i) \Rightarrow (v_{i+1}, m_{i+1})$  ( $i = 0, 1, \dots, n-1$ ), and  $(v_n, m_n)$  is a halting ID.

**Example 2** Consider the CTAG  $C_1 = (3, \{Y, N\}, (\text{halt}, YN, YY))$ . The complete computation starting from  $NYY$  is  $(NYY, 0) \Rightarrow (YY, 1) \Rightarrow (YYN, 2) \Rightarrow (YNYY, 0)$ .

We now show there is a 17-state 5-symbol URTM that simulates any CTAG. A URTM (17,5)  $T_{17,5}$  in the quintuple form is as follows (Morita and Yamaguchi 2007).

$$T_{17,5} = (\{q_0, \dots, q_{16}\}, \{b, Y, N, *, \$\}, q_0, b, \delta),$$

where the set  $\delta$  of quintuples is shown in Table 5. (Note that, in a construction of a UTM, the final state is usually omitted from the state set). There are 67 quintuples in total. If a CTAG halts with a

**Reversible Computing,**

**Table 5** The set of quintuples of the URTM  $T_{17,5}$  (Morita and Yamaguchi 2007)

	$b$	$Y$	$N$	*	\$
$q_0$	$\$ - q_2$	$\$ - q_1$	$b - q_{13}$		
$q_1$	Halt	$Y - q_1$	$N - q_1$	$* + q_0$	$b - q_1$
$q_2$	$* - q_3$	$Y - q_2$	$N - q_2$	$* - q_2$	Null
$q_3$	$b + q_{12}$	$b + q_4$	$b + q_7$	$b + q_{10}$	
$q_4$	$Y + q_5$	$Y + q_4$	$N + q_4$	$* + q_4$	$\$ + q_4$
$q_5$	$b - q_6$				
$q_6$	$Y - q_3$	$Y - q_6$	$N - q_6$	$* - q_6$	$\$ - q_6$
$q_7$	$N + q_8$	$Y + q_7$	$N + q_7$	$* + q_7$	$\$ + q_7$
$q_8$	$b - q_9$				
$q_9$	$N - q_3$	$Y - q_9$	$N - q_9$	$* - q_9$	$\$ - q_9$
$q_{10}$		$Y + q_{10}$	$N + q_{10}$	$* + q_{10}$	$\$ + q_{11}$
$q_{11}$		$Y + q_{11}$	$N + q_{11}$	$* + q_{11}$	$Y + q_0$
$q_{12}$		$Y + q_{12}$	$N + q_{12}$	$* + q_{12}$	$\$ - q_3$
$q_{13}$	$* - q_{14}$	$Y - q_{13}$	$N - q_{13}$	$* - q_{13}$	$\$ - q_{13}$
$q_{14}$	$b + q_{16}$	$Y - q_{14}$	$N - q_{14}$	$b + q_{15}$	
$q_{15}$	$N + q_0$	$Y + q_{15}$	$N + q_{15}$	$* + q_{15}$	$\$ + q_{15}$
$q_{16}$		$Y + q_{16}$	$N + q_{16}$	$* + q_{16}$	$\$ - q_{14}$

halting ID, then  $T_{17,5}$  halts in the state  $q_1$ . If the string becomes an empty string, then it halts in the state  $q_2$ . In Table 5, it is indicated by “null.”

Figure 21 shows how the CTAG  $C_1$  with the initial string  $NYY$  in Example 2 is simulated by the URTM  $T_{17,5}$ . On the tape of the URTM, the production rules (halt,  $NY$ ,  $YY$ ) of  $C_1$  are expressed by the reversal sequence over  $\{Y, N, *\}$ , i.e.,  $YY^*$

$YN^{**}$ , where  $*$  is used as a delimiter between rules and “halt” is represented by the empty string. Note that in the initial tape of  $T_{17,5}$  ( $t = 0$ ), the rightmost  $*$  is replaced by  $b$ . This indicates that the phase is 0. In general, if the phase is  $i - 1$ , then the  $i$ -th  $*$  from the right is replaced by  $b$ . This symbol  $b$  is called a “phase marker.” On the other hand, the given initial string for  $C_1$  is placed to the right of the rules, where  $\$$  is used as a delimiter.

The IDs in the complete computation  $(NYY, 0) \Rightarrow (YY, 1) \Rightarrow (YN, 2) \Rightarrow (YNY, 0)$  of  $C_1$  appear in the computational configurations of  $T_{17,5}$  at  $t = 0, 6, 59$ , and 142, respectively. The symbol  $\$$  in the final string ( $t = 148$ ) should be regarded as  $Y$ .

It is easy to see that  $T_{17,5}$  is reversible by checking the set of quintuples shown in Table 5 according to the definition of an RTM. Intuitively, its reversibility is guaranteed from the fact that no information is erased in the whole simulation process. Furthermore, every branch of the program caused by reading the symbol  $Y$  or  $N$  is “merged reversibly” by writing the original symbol. For

example, the states  $q_{11}$  and  $q_{15}$  go to the same state  $q_0$  by writing  $Y$  and  $N$ , respectively, using the quintuples  $[q_{11}, Y, +, q_0]$  and  $[q_{15}, b, N, +, q_0]$ .

Besides URTM(17,5), several small URTMs that simulate CTAGs have been shown. They are URTM(10,8), URTM(13,7), URTM(15,6), URTM(24,4), and URTM(32,3) given by Morita (2008, 2015b, 2017).

It is generally difficult to design an RTM that has only two symbols or a very small number of states. To obtain an RTM with a small number of states, general procedures for converting a given many-state RTM into a 4-state and 3-state RTMs are given in Morita (2014b), though the number of symbols of the resulting RTMs becomes very large.

**Lemma 7** (Morita 2014b) *For any 1-tape  $m$ -state  $n$ -symbol RTM  $T$ , we can construct a 1-tape 4-state  $(2mn + n)$ -symbol RTM  $T'$  that simulates  $T$ .*

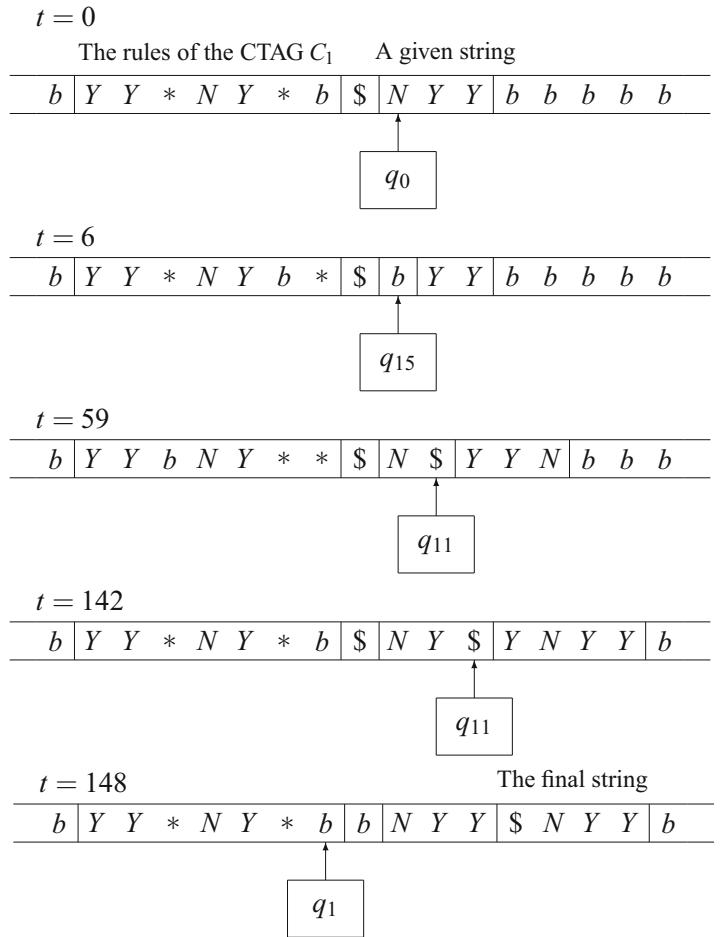
**Lemma 8** (Morita 2014b) *For any 1-tape  $m$ -state  $n$ -symbol RTM  $T$ , we can construct a 1-tape 3-state RTM  $T'$  with  $O(m^2n^3)$ -symbols that simulates  $T$ .*

Applying Lemma 7 to URTM(10,8), we can obtain URTM(4,168). Likewise, applying Lemma 8 to URTM(32,3), we obtain URTM(3, 36654). On the other hand, so far it is not known whether 2-state URTM exists.

To construct a 2-symbol URTM, we can use a method of converting a many-symbol RTM into

**Reversible Computing,**

**Fig. 21** Simulating the CTAG  $C_1$  in Example 2 by the URTM  $T_{17,5}$



a 2-symbol RTM (Morita et al. 1989). In particular, the following lemma is shown in Morita (2015b) to convert a 4-symbol RTM to a 2-symbol RTM.

**Lemma 9** (Morita 2015b) *For any 1-tape  $m$ -state 4-symbol RTM  $T$ , we can construct a 1-tape  $m'$ -state 2-symbol RTM  $T^\dagger$  that simulates  $T$  such that  $m' \leq 6m$ .*

By this method, we obtain URTM(138,2) from URTM(24,4).

**Other Models of Reversible Computing**

There are several models of reversible computing that are not dealt with in the previous sections. Here, we discuss some of them briefly.

A counter machine (CM) is a simple model of computing consisting of a read-only input tape, a finite number of counters, and a finite control. It is known that a CM with only two counters has computational universality (Minsky 1967). A *reversible counter machine* (RCM) is a backward deterministic version of a CM. An RCM with only two counters is known to be computationally universal (Morita 1996), though it is a very simple model. This is useful to show universality of other reversible systems.

A *reversible finite automaton* (RFA) is also a backward deterministic version of a finite automaton. Note that an RSM in section “Reversible Logic Element with Memory” is another kind of reversible finite automaton augmented by an output mechanism. Angluin (1982) and Pin (1992) studied RFAs from the formal language theory

and showed the class of RFAs is less powerful than the class of irreversible finite automata in their language-accepting capability. Their model can be regarded as a one-way RFA. Kondaks and Watrous (1997) studied a two-way RFA and proved that the class of two-way RFAs exactly characterizes the class of regular languages. Thus, in the two-way case, the language-accepting capability does not decrease by the constraint of reversibility.

A multihead finite automaton consists of a finite control, read-only input tape, and a finite number of heads. A *reversible multihead finite automaton* (RMFA) was first studied in Morita (2011). Similar to the case of RFAs, language-accepting capability of two-way  $k$ -head MFAs does not decrease by the constraint of reversibility (Morita 2013), while one-way  $k$ -head RMFAs is less powerful than one-way  $k$ -head MFAs (Kutrib and Malcher 2013).

A reversible pushdown automaton, which consists of an input tape, an input head that can move only rightward, a pushdown memory, and a finite control, was studied by Kutrib and Malcher (2012). They showed that a reversible deterministic pushdown automaton is strictly less powerful than a deterministic pushdown automaton.

A *reversible cellular automaton* (RCA) is an important model, because it can deal with reversible spatiotemporal phenomena. It can be thought as an abstract model of a reversible space. So far, many interesting results on RCAs have been shown (see, e.g., Kari 2005; Morita 2008; Toffoli and Margolus 1990).

## Future Directions

### How Can We Realize Reversible Computers as a Hardware?

Although we have discussed reversible computing mainly from the standpoint of computation theory, it is a very important problem how reversible computers can be realized as a hardware. So far, there have been many interesting attempts from engineering side: e.g., implementing reversible logic as electrically controlled switches (Merkle 1993), c-MOS implementation of

reversible logic gates and circuits (De Vos et al. 2000), adiabatic circuits for reversible computer (Frank 1999; Frank et al. 1998), and synthesis of reversible logic circuits (Al-Rabadi 2004; De Vos 2010; Saeedi and Markov 2013; Shende et al. 2003; Thapliyal and Ranganathan 2010; Wille and Drechsler 2010). However, ultimately, reversible logic elements and computers should be implemented in atomic or molecular level. It is plausible that some nano-scale phenomena can be used as primitives of reversible computing, because the microscopic physical law is reversible. Of course, finding such solution is very difficult, but it is an interesting and challenging problem left for future investigations.

### How Simple Can Reversible Computers Be?

To find very simple reversible logic elements with universality is an important problem from both theoretical and practical viewpoints. We can assume hardware implementation will generally become easier, if we can find simpler logical primitives from which reversible computers can be built. As we have discussed in section “Reversible Logic Gates,” the Fredkin gate and the Toffoli gate are logically universal gates with the minimum number of inputs and outputs. On the other hand, as for reversible logic elements with memory (RLEM) given in section “Reversible Logic Element with Memory,” there are 14 kinds of 2-state 3-symbol universal RLEM (Morita et al. 2005; Ogiro et al. 2005), which have a smaller number of symbols than RE. Since three nondegenerate 2-state 2-symbol RLEM among four have been shown to be nonuniversal (Theorem 5), we have a conjecture that 2-state 3-symbol universal RLEM are the minimum ones. However, it is left open whether the 2-state 2-symbol RLEM 2–17 is universal or not. It is also an interesting problem to find much smaller universal reversible Turing machines than those listed in section “Universal RTMs.”

### Novel Architectures for Reversible Computers

If we try to construct more realistic computers from reversible logic primitives, we shall need new design theories suited for it. For example, a

circuit composed of REs shown in Fig. 20 has a very different feature from traditional logic circuits made of gates. There will surely be many other possibilities of new design techniques that cannot be imagined from the classical theory of logic circuits. To realize efficient reversible computers, development of novel design methods and architectures are necessary.

### Asynchronous and Continuous-Time Reversible Models

There are still other problems between reversible physical systems and reversible computing systems. The first one is asynchronous systems versus synchronous ones. When building a computing system with a huge number of elements, it is preferable if it is realized as an asynchronous system, because if a clock is eliminated, then power dissipation is reduced, and each element can operate at any time independent to other elements (Lee et al. 2004; Peper et al. 2003). However, it is a difficult problem to define reversibility in asynchronous systems properly. The second problem is continuous time versus discrete time. Natural physical systems are continuous (at least they seem so), while most reversible computing models are defined as discrete systems. To bridge a gap between reversible physical systems and reversible computing models, and to implement the latter in the former systems, we shall need further investigations on these problems.

## Bibliography

### Primary Literature

- Al-Rabadi A-N (2004) Reversible logic synthesis. Springer, Berlin, Heidelberg
- Angluin D (1982) Inference of reversible languages. J ACM 29:741–765
- Barenco A, Bennett C-H, Cleve R, DiVincenzo DP, Margolus N, Shor P, Sleator T, Smolin J, Weinfurter H (1995) Elementary gates for quantum computation. Phys Rev A 52:3457–3467
- Bennett C-H (1973) Logical reversibility of computation. IBM J Res Dev 17:525–532
- Bennett C-H (1982) The thermodynamics of computation – a review. Int J Theor Phys 21:905–940
- Bennett C-H (1987) Demons, engines, and the second law. Sci Am 257:108–111

- Bennett C-H (1989) Time/space trade-offs for reversible computation. SIAM J Comput 18:766–776
- Bennett C-H (2003) Notes on Landauer's principle, reversible computation, and Maxwell's Demon. Stud Hist Philos Mod Phys 34:501–510
- Bennett C-H, Landauer R (1985) The fundamental physical limits of computation. Sci Am 253:38–46
- Buhrman H, Tromp J, Vitányi P (2001) Time and space bounds for reversible simulation. In: Proceedings of the ICALP 2001, LNCS 2076. pp 1017–1027
- Cook M (2004) Universality in elementary cellular automata. Compl Syst 15:1–40
- De Vos A (2010) Reversible computing: fundamentals, quantum computing, and applications. Wiley-VCH, Weinheim
- De Vos A, Desoete B, Adamski A, Pietrzak P, Sibinski M, Widerski T (2000) Design of reversible logic circuits by means of control gates. In: Soudris D, Pirsch P, Barke E (eds) Proceedings of the PATMOS 2000. LNCS 1918. pp 255–264
- Deutsch D (1985) Quantum theory, the Church-Turing principle and the universal quantum computer. Proc R Soc Lond A 400:97–117
- Feynman R-P (1982) Simulating physics with computers. Int J Theor Phys 21:467–488
- Feynman R-P (1996) In: Hey AJG, Allen RW (eds) Feynman lectures on computation. Perseus Books, Reading
- Frank M-P (1999) Reversibility for efficient computing. PhD thesis, MIT
- Frank M-P, Vieri C, Ammer M-J, Love N, Margolus N-H, Knight T-E (1998) A scalable reversible computer in silicon. In: Calude CS, Casti J, Dinneen MJ (eds) Unconventional models of computation. Springer, Singapore, pp 183–200
- Fredkin E, Toffoli T (1982) Conservative logic. Int J Theor Phys 21:219–253
- Gruska J (1999) Quantum computing. McGraw-Hill, London
- Jacopini G, Mentrasti P, Sontacchi G (1990) Reversible Turing machines and polynomial time reversibly computable functions. SIAM J Discret Math 3:241–254
- Kari J (2005) Reversible cellular automata. In: de Felice C, Restivo A (eds) Proceedings of the DLT 2005. LNCS 3572. pp 57–68
- Keyes R-W, Landauer R (1970) Minimal energy dissipation in logic. IBM J Res Dev 14:152–157
- Kondacs A, Watrous J (1997) On the power of quantum finite state automata. In: Proceedings of the 36th FOCS. IEEE, pp 66–75
- Kudlek M, Rogozhin Y (2002) A universal Turing machine with 3 states and 9 symbols. In: Kuich W, Rozenberg G, Salomaa A (eds) Proceedings of the DLT 2001. LNCS 2295. pp 311–318
- Kutrib M, Malcher A (2012) Reversible pushdown automata. J Comput Syst Sci 78:1814–1827
- Kutrib M, Malcher A (2013) One-way reversible multi-head finite automata. In: Glück R, Yokoyama T (eds) Proceedings of the RC 2012. LNCS 7581. pp 14–28

- Landauer R (1961) Irreversibility and heat generation in the computing process. *IBM J Res Dev* 5:183–191
- Lange K-J, McKenzie P, Tapp A (2000) Reversible space equals deterministic space. *J Comput Syst Sci* 60:354–367
- Lecerf Y (1963) Machines de Turing réversibles – récursive insolubilité en  $n \in \mathbb{N}$  de l'équation  $u = \theta^n u$ , où  $\theta$  est un isomorphisme de codes. *C R Hebd Séances Acad Sci* 257:2597–2600
- Lee J, Peper F, Adachi S, Morita K (2004) Universal delay-insensitive circuits with bidirectional and buffering lines. *IEEE Trans Comput* 53:1034–1046
- Lee J, Peper F, Adachi S, Morita K (2008) An asynchronous cellular automaton implementing 2-state 2-input 2-output reversed-twin reversible elements. In: Umeo H et al (eds) *Proceedings of the ACRI 2008*. LNCS 5191. pp 67–76
- Merkle R-C (1993) Reversible electronic logic using switches. *Nanotechnology* 4:20–41
- Minsky M-L (1967) Computation: finite and infinite machines. Prentice-Hall, Englewood Cliffs
- Morita K (1990) A simple construction method of a reversible finite automaton out of Fredkin gates, and its related problem. *Trans IEICE Jpn E-73:978–984*
- Morita K (1996) Universality of a reversible two-counter machine. *Theor Comput Sci* 168:303–320
- Morita K (2001) A simple reversible logic element and cellular automata for reversible computing. In: Margenstern M, Rogozhin Y (eds) *Proceedings of the MCU 2001*. LNCS 2055. pp 102–113
- Morita K (2003) A new universal logic element for reversible computing. In: Martin-Vide C, Mitrana V (eds) *Grammars and automata for string processing*. Taylor & Francis, London, pp 285–294
- Morita K (2008) Reversible computing and cellular automata – a survey. *Theor Comput Sci* 395:101–131
- Morita K (2011) Two-way reversible multi-head finite automata. *Fundam Inform* 110:241–254
- Morita K (2012) Reversible computing. Kindai Kagaku-sha Co., Ltd., Tokyo. ISBN 978-4-7649-0422-4 (in Japanese)
- Morita K (2013) A deterministic two-way multi-head finite automaton can be converted into a reversible one with the same number of heads. In: Glück R, Yokoyama T (eds) *Proceedings of the RC 2012*. LNCS 7581. pp 29–43. Slides with figures of computer simulation: Hiroshima University Institutional Repository. <http://ir.lib.hiroshima-u.ac.jp/00033875>
- Morita K (2014a) Reversibility in space-bounded computation. *Int J Gen Syst* 43:697–712
- Morita K (2014b) Reversible Turing machines with a small number of states. In: Bensch S, Freund R, Otto F (eds) *Proceedings of the NCMA 2014*. pp 179–190. Slides with figures of computer simulation: Hiroshima University Institutional Repository. <http://ir.lib.hiroshima-u.ac.jp/00036075>
- Morita K (2015a) Constructing reversible Turing machines by reversible logic element with memory. In: Adamatzky A (ed) *Automata, universality, computation*. Springer, Cham, pp 127–138. Slides with figures of computer simulation: Hiroshima University Institutional Repository. <http://ir.lib.hiroshima-u.ac.jp/00029224>
- Morita K (2015b) Universal reversible Turing machines with a small number of tape symbols. *Fundam Inform* 138:17–29
- Morita K (2017) Two small universal reversible Turing machines. In: Adamatzky A (eds) *Advances in unconventional computing vol.1: theory*. Springer, Cham, pp 221–237
- Morita K, Yamaguchi Y (2007) A universal reversible Turing machine. In: Durand-Lose JO, Margenstern M (eds) *Proceedings of the MCU 2007*. LNCS 4664. pp 90–98
- Morita K, Shirasaki A, Gono Y (1989) A 1-tape 2-symbol reversible Turing machine. *Trans IEICE Jpn E-72:223–228*
- Morita K, Ogiro T, Tanaka K, Kato H (2005) Classification and universality of reversible logic elements with one-bit memory. In: Margenstern M (eds) *Proceedings of the MCU 2004*. LNCS 3354. pp 245–256
- Morita K, Ogiro T, Alhazov A, Tanizawa T (2012) Non-degenerate 2-state reversible logic elements with three or more symbols are all universal. *J Mult Valued Log Soft Comput* 18:37–54
- Mukai Y, Morita K (2012) Realizing reversible logic elements with memory in the billiard ball model. *Int J Unconv Comput* 8:47–59
- Mukai Y, Ogiro T, Morita K (2014) Universality problems on reversible logic elements with 1-bit memory. *Int J Unconv Comput* 10:353–373
- Neary T, Woods D (2009) Four small universal Turing machines. *Fundam Inform* 91:123–144
- Ogiro T, Kanno A, Tanaka K, Kato H, Morita K (2005) Nondegenerate 2-state 3-symbol reversible logic elements are all universal. *Int J Unconv Comput* 1:47–67
- Peper F, Lee J, Adachi S, Mashiko S (2003) Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers. *Nanotechnology* 14:469–485
- Petri C-A (1967) Grundsätzliches zur Beschreibung diskreter Prozesse. In: Händler W, Peschl E, Unger H (eds) *Proceedings of the 3rd Colloquium über Automatentheorie*. Birkhäuser Verlag, Basel, pp 121–140
- Pin J-E (1992) On reversible automata. In: Simon I (ed) *Proceedings of the LATIN '92*. LNCS 583. pp 401–416
- Rogozhin Y (1996) Small universal Turing machines. *Theor Comput Sci* 168:215–240
- Saeedi M, Markov I-L (2013) Synthesis and optimization of reversible circuits – a survey. *ACM Comput Surv* 45:21
- Shende V-V, Prasad A-K, Markov I-L, Hayes J-P (2003) Synthesis of reversible logic circuits. *IEEE Trans Comput Aided Des Integr Circ Syst* 22:710–722
- Thapliyal H, Ranganathan N (2010) Design of reversible sequential circuits optimizing quantum cost, delay, and garbage outputs. *ACM J Emerg Technol Comput Syst* 6:14:1–14:31

- Toffoli T (1977) Computation and construction universality of reversible cellular automata. *J Comput Syst Sci* 15:213–231
- Toffoli T (1980) Reversible computing. In: de Bakker JW, van Leeuwen J (eds) *Automata, languages and programming*. LNCS 85. Springer, Berlin, pp 632–644
- Toffoli T (1981) Bicontinuous extensions of invertible combinatorial functions. *Math Syst Theory* 14:12–23
- Toffoli T, Margolus N (1990) Invertible cellular automata: a review. *Phys D* 45:229–253
- Wille R, Drechsler R (2010) Towards a design flow for reversible logic. Springer, Dordrecht
- Books and Reviews**
- Adamatzky A (ed) (2002) Collision-based computing. Springer, London
- Bennett CH (1988) Notes on the history of reversible computation. *IBM J Res Dev* 32:16–23
- Milburn GJ (1998) The Feynman processor. Perseus Books, Reading
- Morita K (2017) Theory of reversible computing, Springer, Tokyo
- Vitanyi P (2005) Time, space, and energy in reversible computing. In: Proceedings of the 2005 ACM international conference on computing frontiers, pp 435–444

---

## **Part II**

### **Theory of Computation and Nature-Inspired Algorithms**



## Bacterial Computing

Martyn Amos

Department of Computing and Mathematics,  
Manchester Metropolitan University, Manchester,  
UK

### Article Outline

Glossary

Definition of the Subject

Introduction

Motivation for Bacterial Computing

The Logic of Life

Rewiring Genetic Circuitry

Successful Implementations

Future Directions

Bibliography

### Glossary

**DNA** Deoxyribonucleic acid. Molecule that encodes the genetic information of cellular organisms.

**Operon** Set of functionally related genes with a common promoter (“on switch”).

**Plasmid** Small circular DNA molecule used to transfer genes from one organism to another.

**RNA** Ribonucleic acid. Molecule similar to DNA, which helps in the conversion of genetic information to proteins.

**Transcription** Conversion of a genetic sequence into RNA.

**Translation** Conversion of an RNA sequence into an amino acid sequence (and, ultimately, a protein).

### Definition of the Subject

Bacterial computing is a conceptual subset of *synthetic biology*, which is itself an emerging scientific discipline largely concerned with the

engineering of biological systems. The goals of synthetic biology may be loosely partitioned into four sets: (1) to better understand the fundamental operation of the biological system being engineered, (2) to extend synthetic chemistry and create improved systems for the synthesis of molecules, (3) to investigate the “optimization” of existing biological systems for human purposes, and (4) to develop and apply rational engineering principles to the design and construction of biological systems. It is on these last two goals that we focus in this entry.

The main benefits that may accrue from these studies are both theoretical and practical; the construction and study of synthetic biosystems could improve our quantitative understanding of the fundamental underlying processes, as well as suggest plausible applications in fields as diverse as pharmaceutical synthesis and delivery, biosensing, tissue engineering, bionanotechnology, biomaterials, energy production, and environmental remediation.

### Introduction

Complex natural processes may often be described in terms of networks of computational components, such as Boolean logic gates or artificial neurons. The interaction of biological molecules and the flow of information controlling the development and behavior of organisms are particularly amenable to this approach, and these models are well established in the biological community. However, only relatively recently have papers appeared proposing the use of such systems to perform useful, *human-defined* tasks. For example, rather than merely using the network analogy as a convenient technique for clarifying our understanding of complex systems, it may now be possible to harness the power of such systems for the purposes of computation.

Despite the relatively recent emergence of biological computing as a distinct research area, the

link between biology and computer science is not a new one. Of course, for years, biologists have used computers to store and analyze experimental data. Indeed, it is widely accepted that the huge advances of the Human Genome Project (as well as other genome projects) were only made possible by the powerful computational tools available. Bioinformatics has emerged as “the science of the twenty-first century,” requiring the contributions of truly interdisciplinary scientists who are equally at home at the lab bench or writing software at the computer.

However, the seeds of the relationship between biology and computer science were sown over 50 years ago, when the latter discipline did not even exist. When, in the seventeenth century, the French mathematician and philosopher René Descartes declared to Queen Christina of Sweden that animals could be considered a class of machines, she challenged him to demonstrate how a clock could reproduce. Three centuries later in 1951, with the publication of *The General and Logical Theory of Automata* (von Neumann 1941), John von Neumann showed how a machine could indeed construct a copy of itself. Von Neumann believed that the behavior of *natural* organisms, although orders of magnitude more complex, was similar to that of the most intricate machines of the day. He believed that life was based on *logic*. We now begin to look at how this view of life may be used, not simply as a useful analogy, but as the practical foundation of a whole new engineering discipline.

## Motivation for Bacterial Computing

Here we consider the main motivations behind recent work on bacterial computing (and, more broadly, synthetic biology). Before recombinant DNA technology made it possible to construct new genetic sequences, biologists were restricted to crudely “knocking out” individual genes from an organism’s genome and then assessing the damage caused (or otherwise). Such knockouts gradually allowed them to piece together fragments of causality, but the process was very time-consuming and error prone.

Since the dawn of genetic engineering – with the ability to synthesize novel gene segments –

biologists have been in a position to make much more finely tuned modifications to their organism of choice, this generating much more refined data. Other advances in biochemistry have also contributed, allowing scientists to – for example – investigate new types of genetic systems with, for example, twelve bases, rather than the traditional four (Geyer et al. 2003). Such creations have yielded valuable insights into the mechanics of mutation, adaptation, and evolution. Researchers in synthetic biology are now extending their work beyond the synthesis of single genes and are now introducing whole new gene *complexes* into organisms.

The objectives behind this work are both theoretical and practical. As Benner and Seymour argue (Benner and Sismour 2005), “... a synthetic goal forces scientists to cross uncharted ground to encounter and solve problems that are not easily encountered through [top-down] analysis. This drives the emergence of new paradigms [“world views”] in ways that analysis cannot easily do.” Drew Endy agrees. “Worst-case scenario, it’s a complementary approach to traditional discovery science” (Hopkin 2004). “Best-case scenario, we get systems that are simpler and easier to understand ...” Or, to put it bluntly, “Let’s build new biological systems - systems that are easier to understand because we made them that way” (Morton 2005). As well as shedding new light on the underlying biology, these novel systems may well have significant practical utility. Such new creations, according to Endy’s “personal wish list,” might include “generating biological machines that could clean up toxic waste, detect chemical weapons, perform simple computations, stalk cancer cells, lay down electronic circuits, synthesize complex compounds and even produce hydrogen from sunlight” (Hopkin 2004). In the next section, we begin to consider how this might be achieved, by first describing the underlying logic of genetic circuitry.

## The Logic of Life

Twenty years after von Neumann’s seminal paper, Francois Jacob and Jacques Monod identified

specific natural processes that could be viewed as behaving according to logical principles:

The logic of biological regulatory systems abides not by Hegelian laws but, like the workings of computers, by the propositional algebra of George Boole. (Monod 1970).

This conclusion was drawn from earlier work of Jacob and Monod (Monod et al. 1963). In addition, Jacob and Monod described the “lactose system” (Jacob and Monod 1961), which is one of the archetypal examples of a Boolean biosystem. We describe this system shortly, but first give a brief introduction to the operation of genes in general terms.

### DNA as the Carrier of Genetic Information

The *central dogma* of molecular biology (Crick 1970) is that DNA produces RNA, which in turn produces proteins. The basic “building blocks” of genetic information are known as *genes*. Each gene codes for one specific *protein* and may be turned on (*expressed*) or off (*repressed*) when required.

### Transcription and Translation

We now describe the processes that determine the structure of a protein and hence its function. Note that in what follows we assume the processes described occur in bacteria rather than in higher organisms such as humans. For a full description of the structure of the DNA molecule, see the chapter on *DNA computing*. In order for a DNA sequence to be converted into a protein molecule, it must be read (*transcribed*) and the transcript converted (*translated*) into a protein. Transcription of a gene produces a *messenger RNA* (mRNA) copy, which can then be translated into a protein.

Transcription proceeds as follows. The mRNA copy is synthesized by an enzyme known as *RNA polymerase*. In order to do this, the RNA polymerase must be able to recognize the specific region to be transcribed. This specificity requirement facilitates the regulation of genetic expression, thus preventing the production of unwanted proteins. Transcription begins at specific sites within the DNA sequence, known as *promoters*.

These promoters may be thought of as “markers,” or “signs,” in that they are not transcribed into RNA. The regions that *are* transcribed into RNA (and eventually translated into protein) are referred to as *structural genes*. The RNA polymerase recognizes the promoter, and transcription begins. In order for the RNA polymerase to begin transcription, the double helix must be opened so that the sequence of bases may be read. This opening involves the breaking of the hydrogen bonds between bases. The RNA polymerase then moves along the DNA *template* strand in the 3 → 5' direction. As it does so, the polymerase creates an *antiparallel* mRNA chain (i.e., the mRNA strand is the equivalent of the Watson-Crick complement of the template). However, there is one significant difference, in that RNA contains uracil instead of thymine. Thus, in mRNA terms, “U binds with A.”

The RNA polymerase moves along the DNA, the DNA recoiling into its double-helix structure behind it, until it reaches the end of the region to be transcribed. The end of this region is marked by a *terminator* which, like the promoter, is not transcribed.

### Genetic Regulation

Each step of the conversion, from stored information (DNA), through mRNA (messenger), to protein synthesis (effector), is itself catalyzed by other effector molecules. These may be enzymes or other factors that are required for a process to continue (e.g., sugars). Consequently, a loop is formed, where products of one gene are required to produce further gene products and may even influence that gene’s own expression. This process was first described by Jacob and Monod in 1961 (Jacob and Monod 1961), a discovery that earned them a share of the 1965 Nobel Prize in Physiology or Medicine.

Genes are composed of a number of distinct regions, which control and encode the desired product. These regions are generally of the form promoter-gene-terminator. Transcription may be regulated by effector molecules known as *inducers* and *repressors*, which interact with the promoter and increase or decrease the level of transcription. This allows effective control over

the expression of proteins, avoiding the production of unnecessary compounds. It is important to note at this stage that, in reality, genetic regulation does not conform to the digital “on-off” model that is popularly portrayed; rather, it is continuous or analog in nature.

### The *Lac* Operon

One of the most well-studied genetic systems is the *lac operon*. An *operon* is a set of functionally related genes with a common promoter. An example of this is the *lac* operon, which contains three structural genes that allow *E. coli* to utilize the sugar lactose.

When *E. coli* is grown on the sugar glucose, the product of the (separate and unrelated to the *lac* operon) *lacI* gene *represses* the transcription of the *lacZYA* operon (i.e., the operon is turned off). However, if lactose is supplied *together with* glucose, a lactose by-product is produced which interacts with the repressor molecule, preventing it from repressing the *lacZYA* operon. This derepression does not itself initiate transcription, since it would be inefficient to utilize lactose if the more common sugar glucose were still available. The operon is *positively regulated* (i.e., “encouraged”) by a different molecule, whose level increases as the amount of available glucose decreases. Therefore, if lactose were present as the sole carbon source, the *lacI* repression would be relaxed and the high “encouraging” levels would activate transcription, leading to the synthesis of the *lacZYA* gene products. Thus, the promoter is under the control of two sugars, and the *lacZYA* operon is only transcribed when lactose is *present* and glucose is *absent*.

In essence, Jacob and Monod showed how a gene may be thought of (in very abstract terms) as a binary switch and how the state of that switch might be affected by the presence or absence of certain molecules. Monod’s point, made in his classic book *Chance and Necessity* and quoted above, was that the workings of biological systems operate not by Hegel’s philosophical or metaphysical logic of understanding, but according to the formal, mathematically grounded logical system of George Boole.

What Jacob and Monod found was that the transcription of a gene may be regulated by molecules known as *inducers* and *repressors*, which either increase or decrease the “volume” of a gene (corresponding to its level of transcription, which is not always as clear-cut and binary as Monod’s quote might suggest). These molecules interact with the promoter region of a gene, allowing the gene’s level to be finely “tuned.” The *lac* genes are so-called because, in the *E. coli* bacterium, they combine to produce a variety of proteins that allow the cell to metabolize the sugar lactose (which is most commonly found in milk, hence the derivation from the Latin word *lact*, meaning milk).

For reasons of efficiency, these proteins should only be produced (i.e., the genes be turned on) when lactose is present in the cell’s environment. Making these proteins when lactose is *absent* would be a waste of the cell’s resources, after all. However, a different sugar – glucose – will *always* be preferable to lactose, if the cell can get it, since glucose is an “easier” form of sugar to metabolize. So, the input to and output from the *lac* operon may be expressed as a truth table, with G and L standing for glucose and lactose (1 if present, 0 if absent) and O standing for the output of the operon (1 if on, 0 if off):

G	L	O
0	0	0
0	1	1
1	0	0
1	1	0

The Boolean function that the *lac* operon therefore *physically* computes is (L AND (NOT G)), since it only outputs 1 if L = 1 (lactose present) and G = 0 (glucose is absent). By showing how one gene could affect the expression of another – just like a transistor feeds into the input of another and affects its state – Jacob and Monod laid the foundations for a new way of thinking about genes, not simply in terms of protein blueprints, but of *circuits* of interacting parts, or dynamic networks of interconnected switches and logic gates. This view of the genome is now well established (Kauffman 1993a, b; McAdams and

Shapiro 1995), but in the next section, we show how it might be used to guide the *engineering* of biological systems.

## Rewiring Genetic Circuitry

A key difference between the wiring of a computer chip and the circuitry of the cell is that “electronics engineers know exactly how resistors and capacitors are wired to each other because they installed the wiring. But biologists often don’t have a complete picture. They may not know which of thousands of genes and proteins are interacting at a given moment, making it hard to predict how circuits will behave inside cells” (Ferber 2004). This makes the task of reengineering vastly more complex.

Rather than trying to assimilate the huge amounts of data currently being generated by the various genome projects, synthetic biologists are taking a novel route – simplify and build. “They create models of genetic circuits, build the circuits, see if they work, and adjust them if they don’t - learning about biology in the process. ‘I view it as a reductionist approach to systems biology,’ says biomedical engineer Jim Collins of Boston University” (Ferber 2004).

The field of *systems biology* has emerged in recent years as an alternative to the traditional reductionist way of doing science. Rather than simply focusing on a single level of description (such as individual proteins), researchers are now seeking to *integrate* information from many different layers of complexity. By studying how different biological components *interact*, rather than simply looking at their structure, systems biologists are attempting to build models of systems from the bottom up. A model is simply an abstract description of how a system operates – for example, a set of equations that describe how a disease spreads throughout a population. The point of a model is to capture the *essence* of a system’s operation, and it should therefore be as simple as possible. Crucially, a model should also be capable of making *predictions*, which can then be tested against reality using real data (e.g., if infected people are placed in quarantine for a

week, how does this affect the progress of the disease in question, or what happens if I feed this signal into the chip?). The results obtained from these tests may then feed back into further refinements of the model, in a continuous cycle of improvement.

When a model suggests a plausible structure for a synthetic genetic circuit, the next stage is to engineer it into the chosen organism, such as a bacterium. Once the structure of the DNA molecule was elucidated and the processes of transcription and translation were understood, molecular biologists were frustrated by the lack of suitable experimental techniques that would facilitate more detailed examination of the genetic material. However, in the early 1970s, several techniques were developed that allowed previously impossible experiments to be carried out (see Brown 1990; Old and Primrose 1994). These techniques quickly led to the first ever successful cloning experiments (Jackson et al. 1972; Lobban and Sutton 1973). Cloning is generally defined as “... the production of multiple identical copies of a single gene, cell, virus, or organism” (Roberts and Murrell 1998). This is achieved as follows: a specific sequence (corresponding, perhaps, to a novel gene) is inserted in a circular DNA molecule, known as a *plasmid*, or *vector*, producing a *recombinant DNA molecule*. The vector acts as a *vehicle*, transporting the sequence into a *host* cell (usually a bacterium, such as *E. coli*). Cloning single genes is well established, but is often done on an ad hoc basis. If biological computing is to succeed, it requires some degree of *standardization*, in the same way that computer manufacturers build different computers, but using a standard library of components. “Biobricks are the first example of standard biological parts,” explains Drew Endy (Jha 2005). “You will be able to use biobricks to program systems that do whatever biological systems do.” He continues, “That way, if in the future, someone asks me to make an organism that, say, counts to 3,000 and then turns left, I can grab the parts I need off the shelf, hook them together and predict how they will perform” (Gibbs 2004).

Each biobrick is a simple component, such as an AND gate, or an inverter (NOT). Put them

together one after the other, and you have a NAND (NOT-AND) gate, which is all that is needed to build any Boolean circuit (an arbitrary circuit can be translated to an equivalent circuit that uses only NAND gates. It will be much bigger than the original, but it will compute the same function. Such considerations were important in the early stages of integrated circuits, when building *different* logic gates was difficult and expensive). Just as transistors can be used together to build logic gates, and these gates then combined into circuits, there exists a hierarchy of complexity with biobricks. At the bottom are the “parts,” which generally correspond to the coding regions for proteins. Then, one level up, we have “devices,” which are built from parts – the oscillator of Elowitz and Leibler, for example, could be constructed from three inverter devices chained together, since all an inverter does is “flip” its signal from 1 to 0 (or vice versa). This circuit would be an example of the biobricks at the top of the conceptual tree – “systems” – which are collections of parts to do a significant task (like oscillating or counting).

Tom Knight at MIT made the first six biobricks, each held in a plasmid ready for use. As we have stated, plasmids can be used to insert novel DNA sequences in the genomes of bacteria, which act as the “test bed” for the biobrick circuits. “Just pour the contents of one of these vials into a standard reagent solution, and the DNA will transform itself into a functional component of the bacteria,” he explains (Brown 2004). Drew Endy was instrumental in developing this work further, one invaluable resource being the Registry of Standard Biological Parts ([Registry of Standard Biological Parts](#)), the definitive catalog of new biological component. At the start of 2006, it contained 224 “basic” parts and 459 “composite” parts, with 270 parts “under construction.” Biobricks are still at a relatively early stage, but “Eventually we’ll be able to design and build in silico and go out and have things synthesized,” says Jay Keasling, head of Lawrence Berkeley National Laboratory’s new synthetic biology department (Ferber 2004).

## Successful Implementations

Although important foundational work had been performed by Arkin and Ross as early as 1994 (Arkin and Ross 1994), the year 2000 was a particularly significant one for synthetic biology. In January two foundational papers appeared back to back in the same issue of *Nature*. “Networks of interacting biomolecules carry out many essential functions in living cells, but the ‘design principles’ underlying the functioning of such intracellular networks remain poorly understood, despite intensive efforts including quantitative analysis of relatively simple systems. Here we present a complementary approach to this problem: the design and construction of a synthetic network to implement a particular function” (Elowitz and Leibler 2000). That was the introduction to a paper that Drew Endy would call “the high-water mark of a synthetic genetic circuit that does something” (Ferber 2004). In the first of the two articles, Michael Elowitz and Stanislau Leibler (then both at Princeton) showed how to build microscopic “Christmas tree lights” using bacteria.

### Synthetic Oscillator

In physics, an *oscillator* is a system that produces a regular, periodic “output.” Familiar examples include a pendulum, a vibrating string, or a lighthouse. Linking several oscillators together in some way gives rise to *synchrony* – for example, heart cells repeatedly firing in unison or millions of fireflies blinking on and off, seemingly as one (Strogatz 2003).

Leibler actually had *two* articles published in the same high-impact issue of *Nature*. The other was a short communication, coauthored with Naama Barkai – also at Princeton but in the Department of Physics (Barkai and Leibler 2000). In their paper titled “Circadian clocks limited by noise,” Leibler and Barkai showed how a simple model of biochemical networks could oscillate reliably, even in the presence of noise. They argued that such oscillations (which might, e.g., control the internal circadian clock that tells us when to wake up and when to be tired) are based on networks of *genetic regulation*. They built a simulation of a simple regulatory network

using a *Monte Carlo* algorithm. They found that, however they perturbed the system, it still oscillated reliably, although, at the time, their results existed only in silico. The other paper by Leibler was much more applied, in the sense that they had constructed a biological circuit (Elowitz and Leibler 2000). Elowitz and Leibler had succeeded in constructing an artificial genetic oscillator in cells, using a synthetic network of repressors. They called this construction the *repressilator*.

Rather than investigating existing oscillator networks, Elowitz and Leibler decided to build one entirely from first principles. They chose three repressor-promoter pairs that had already been sequenced and characterized and first built a mathematical model in software. By running the sets of equations, they identified from their simulation results certain molecular characteristics of the components that gave rise to so-called *limit cycle* oscillations, those that are robust to perturbations. This information from the model's results led Elowitz and Leibler to select strong promoter molecules and repressor molecules that would rapidly decay. In order to implement the oscillation, they chose three genes, each of which affected one of the others by repressing it or turning it off. For the sake of illustration, we call the genes A, B, and C. The product of gene A turns off (represses) gene B. The absence of B (which represses C) allows C to turn on. C is chosen such that it turns gene A off again, and the three genes loop continuously in a “daisy chain” effect, turning on and off in a repetitive cycle. However, some form of *reporting* is necessary in order to confirm that the oscillation is occurring as planned.

Green fluorescent protein (GFP) is a molecule found occurring naturally in the jellyfish *Aequorea victoria*. Biologists find it invaluable because it has one interesting property – when placed under ultraviolet light, it *glows*. Biologists quickly sequenced the gene responsible for producing this protein, as they realized that it could have many applications as a *reporter*. By inserting the gene into an organism, you have a ready-made “status light” – when placed into bacteria, they glow brightly if the gene is turned on and look normal if it is turned off. We can think of it in

terms of a Boolean circuit – if the circuit outputs the value 1, the GFP promoter is produced to turn on the light. If the value is 0, the promoter is not produced, the GFP gene is not expressed, and the light stays off.

Elowitz and Leibler set up their gene network so that the GFP gene would be expressed whenever gene C was turned *off* – when it was turned on, the GFP would gradually decay and fade away. They synthesized the appropriate DNA sequences and inserted them into a *plasmid*, eventually yielding a population of bacteria that blinked on and off in a repetitive cycle, like miniature lighthouses. Moreover, and perhaps most significantly, the period between flashes was longer than the time taken for the cells to divide, showing that the state of the system had been passed on during reproduction.

### Synthetic Toggle Switch

Rather than modeling an existing circuit and then altering it, Elowitz and Leibler had taken a “bottom-up” approach to learning about how gene circuits operate. The other notable paper to appear in that issue was written by Timothy Gardner, Charles Cantor and Jim Collins, all from Boston University in the USA. In 2000, Gardner, Collins, and Cantor observed that genetic switching (such as that observed in the lambda phage (Ptashne 2004)) had not yet been “demonstrated in networks of non-specialized regulatory components” (Gardner et al. 2000). That is to say, at that point, nobody had been able to construct a switch out of genes that had not already been “designed” by evolution to perform that specific task. The team had a similar philosophy to that of Elowitz and Leibler, in that their main motivation was being able to test theories about the fundamental behavior of gene regulatory networks. “Owing to the difficulty of testing their predictions,” they explained, “these theories have not, in general, been verified experimentally. Here we have integrated theory and experiment by constructing and testing a synthetic, bistable [two-state] gene circuit based on the predictions of a simple mathematical model.”

“We were looking for the equivalent of a light switch to flip processes on or off in the cell,”

explained Gardner (Eisenberg 2000). “Then I realized a way to do this with genes instead of with electric circuits.” The team chose two genes that were mutually inhibitory – that is, each produced a molecule that would turn the other off. One important thing to bear in mind is that the system did not have a single input. Although the team acknowledged that bistability might be possible – in theory – using only a single promoter that regulated *itself*, they anticipated possible problems with robustness and experimental tunability if they used that approach. Instead, they decided to use a system whereby each “side” of the switch could be “pressed” by a different stimulus – the addition of a chemical on one and a change in temperature on the other. Things were set up so that if the system was in the state induced by the chemical, it would stay in that state until the temperature was changed and would only change *back* again if the chemical was reintroduced. Importantly, these stimuli did not have to be applied continuously – a “short sharp” burst was enough to cause the switch to flip over. As with the other experiments, Gardner and his colleagues used GFP as the system state reporter, so that the cells glowed in one state and looked “normal” in the other.

In line with the bottom-up approach, they first created a mathematical model of the system and made some predictions about how it would behave inside a cell. Within a year, Gardner had spliced the appropriate genes into the bacteria, and he was able to flip them – at will – from one state to the other. As McAdams and Arkin observed, synthetic “one-way” switches had been created in the mid-1980s, but “this is perhaps the first engineered design exploiting bistability to produce a switch with capability of *reversibly* switching between two ... stable states” (McAdams and Arkin 2000). The potential applications of such a bacterial switch were clear. As they state in the conclusion to their article, “As a practical device, the toggle switch ... may find applications in gene therapy and biotechnology.” They also borrowed from the language of computer programming, using an analogy between their construction and the short “applets” written in the Java language, which now allow us to download and run programs in our Web browser.

“Finally, as a cellular memory unit, the toggle forms the basis for ‘genetic applets’ - self-contained, programmable, synthetic gene circuits for the control of cell function.”

### Engineered Communication

Toward the end of his life, Alan Turing did some foundational work on pattern formation in nature, in an attempt to explain how zebras get their striped coats or leopards their spots. The study of *morphogenesis* (from the Greek words *morphe*, shape, and *genesis*, creation; “amorphous” therefore means “without shape or structure”) is concerned with how cells split to assume new roles and communicate with another to form very precise shapes, such as tissues and organs. Turing postulated that the diffusion of chemical signals both within and between cells is the main driving force behind such complex pattern formation (Turing 1952).

Although Turing’s work was mainly concerned with the processes occurring among cells inside a developing embryo, it is clear that chemical signaling also goes on between bacteria. Ron Weiss of Princeton University was particularly interested in *Vibrio fischeri*, a bacterium that has a symbiotic relationship with a variety of aquatic creatures, including the Hawaiian squid. This relationship is due mainly to the fact that the bacteria exhibit *bioluminescence* – they generate a chemical known as a *Luciferase* (coded by the *Lux* gene), a version of which is also found in fireflies and which causes them to glow when gathered together in numbers. Cells within the primitive light organs of the squid draw in bacteria from the seawater and encourage them to grow. Crucially, once enough bacteria are packed into the light organ, they produce a signal to tell the squid cells to stop attracting their colleagues, and only then do they begin to glow. The cells get a safe environment in which to grow, protected from competition, and the squid has a light source by which to navigate and catch prey. The mechanism by which the *Vibrio* “know” when to start glowing is known as *quorum sensing*, since there have to be sufficient “members” present for luminescence to occur.

The bacteria secrete an *autoinducer* molecule, known as VAI (*Vibrio* autoinducer), which diffuses

through the cell wall. The *Lux* gene (which generates the glowing chemical) needs to be activated (turned on) by a particular protein – which attracts the attention of the polymerase – but the protein can only do this with help from the VAI. Its particular 3D structure is such that it cannot fit tightly onto the gene unless it has been slightly bent out of shape. Where there is enough VAI present, it locks onto the protein and alters its conformation, so that it can turn on the gene. Thus, the concentration of VAI is absolutely crucial; once a critical threshold has been passed, the bacteria “know” that there are enough of them present, and they begin to glow.

Weiss realized that this quorum-based cell-to-cell communication mechanism could provide a powerful framework for the construction of bacterial devices – imagine, for example, a tube of solution containing engineered bacteria that can be added to a sample of seawater, causing it to glow only if the concentration of a particular pollutant exceeds a certain threshold. Crucially, as we will see shortly, it also allows the possibility of generating precise “complex patterned” development.

Weiss set up two colonies of *E. coli*, one containing “sender” and the other “receivers.” The idea was that the senders would generate a chemical signal made up of VAI, which could diffuse across a gap and then be picked up by the receivers. Once a strong enough signal was being communicated, the receivers would glow using GFP to say that it had been picked up. Weiss cloned the appropriate gene sequences (corresponding to a type of biobrick) into his bacteria and placed colonies of receiver cells on a plate, and the receivers started to glow in acknowledgment.

### Synthetic Circuit Evolution

In late 2002, Weiss and his colleagues published another paper, this time describing how rigorous engineering principles may be brought to bear on the problem of designing and building entirely new genetic circuitry. The motivation was clear – “biological circuit engineers will have to confront their inability to predict the precise behavior of even the most simple synthetic networks, a serious shortcoming and challenge for

the design and construction of more sophisticated genetic circuitry in the future” (Yokobayashi et al. 2002).

Together with colleagues Yohei Yokobayashi and Frances Arnold, Weiss proposed a two-stage strategy: first, design a circuit from the bottom up, as Elowitz and others had before, and clone it into bacteria. Such circuits are highly unlikely to work first time, “because the behavior of biological components inside living cells is highly context-dependent, the actual circuit performance will likely differ from the design predictions, often resulting in a poorly performing or nonfunctional circuit.” Rather than simply abandoning their design, Weiss and his team decided to then *tune* the circuit inside the cell itself, by applying the principles of evolution. By inducing mutations in the DNA that they had just introduced, they were able to slightly modify the behavior of the circuit that it represented. Of course, many of these changes would be catastrophic, giving even worse performance than before, but, occasionally, they observed a minor improvement. In that case, they kept the “winning” bacteria, and subjected it to another round of mutation, in a repeated cycle. In a microcosmic version of Darwinian evolution, mutation followed by selection of the fittest took an initially unpromising pool of broken circuits and transformed them into winners. “Ron is utilizing the power of evolution to design networks in ways so that they perform exactly the way you want them to,” observed Jim Collins (Gravitz 2004). In a commentary article in the same issue of the journal, Jeff Hasty called this approach “design then mutate” (Hasty 2002). The team showed how a circuit made up of three genetic gates could be fine-tuned *in vivo* to give the correct performance, and they concluded that “the approach we have outlined should serve as a robust and widely applicable route to obtaining circuits, as well as new genetic devices, that function inside living cells.”

### Pattern Formation

The next topic studied by Weiss and his team was the problem of space – specifically, how to get a population of bacteria to cover a surface with a specific density. This facility could be useful when

designing bacterial biosensors – devices that detect chemicals in the environment and produce a response. By controlling the density of the microbial components, it might be possible to tune the sensitivity of the overall device. More importantly, the ability for cells to control their own density would provide a useful “self-destruct” mechanism where these genetically modified bugs ever to be released into the environment for “real-world” applications.

In “Programmed population control by cell-cell communication and regulated killing” (You et al. 2004), Weiss and his team built on their previous results to demonstrate the ability to keep the density of an *E. coli* population artificially low – that is, below the “natural” density that could be supported by the available nutrients. They designed a genetic circuit that caused the bacteria to generate a different *Vibrio* signaling molecule; only this time, instead of making the cells glow, a sufficient concentration would flip a switch inside the cell, turning on a *killer gene*, encoding a protein that was toxic in sufficient quantities. The system behaved exactly as predicted by their mathematical model. The culture grew at an exponential rate (i.e., doubling every time step) for 7 hours, before hitting the defined density threshold. At that point, the population dropped sharply, as countless cells expired, until the population settled at a steady density significantly (ten times) lower than an unmodified “control” colony. The team concluded that “The population-control circuit lays the foundations for using cell-cell communication to programme interactions among bacterial colonies, allowing the concept of communication -regulated growth and death to be extended to engineering synthetic ecosystems” (You et al. 2004).

The next stage was to program cells to form *specific* spatial patterns in the dish. As we have already mentioned briefly, pattern formation is one characteristic of multicellular systems. This is generally achieved using some form of chemical signaling, combined with a differential response – that is, different cells, although genetically identical, may “read” the environmental signals and react in different ways, depending on their internal state. For example, one cell might be

“hungry” and choose to move toward a food source, while an identical cell might choose to remain in the same spot, since it has adequate levels of energy.

The team used a variant of the sender-receiver model, only this time adding a “distance detection” component to the receiver circuit. The senders were placed in the center of the dish, and the receivers were distributed uniformly across the surface. The receivers were constructed so that they could measure the *strength* of the signal being “beamed” from the senders, a signal which decayed over distance (a little like a radio station gradually breaking up as you move out of the reception area). The cells were engineered so that only those that were either “near” to the senders or “far” from the senders would generate a response (those in the middle region were instructed to keep quiet). These cells are genetically identical and are uniformly distributed over the surface – the differential response comes in the way that they assess the strength of the signal and make a decision on whether or not to respond. The power of the system was increased further by making the near cells glow green and those far away glow red (using a different fluorescent protein).

When the team set the system running, they observed the formation of a “dartboard” pattern, with the “bullseye” being the colony of senders (instructed to glow cyan, or light blue), which was surrounded by a green ring, which in turn was surrounded by a red ring. By placing three sender colonies in a triangle, they were also able to obtain a green heart-shaped pattern, formed by the intersection of three green circles, as well as other patterns, determined solely by the initial configuration of senders (Basu et al. 2005).

### Bacterial Camera

Rather than generating light, a different team decided to use bacteria to *detect* light – in the process – building the world’s first microbial camera. By engineering a dense bed of *E. coli*, a team of students led by Chris Voigt at Berkeley developed light-sensitive “film” capable of storing images at a resolution of 100 megapixels per square inch. *E. coli* are not normally sensitive to

light, so the group took genes coding for *photoreceptors* from blue-green algae and spliced them into their bugs (Levkaya et al. 2005). When light was shone on the cells, it turned on a genetic switch that causes a chemical inside them to permanently darken, thus generating a black “pixel.” By projecting an image onto a plate of bacteria, the team was able to obtain several monochrome images, including the *Nature* logo and the face of team member Andrew Ellington. Nobel Laureate Sir Harry Kroto, discoverer of “buckyballs,” called the team’s camera an “extremely exciting advance” (Marks 2005), going on to say that “I have always thought that the first major nanotechnology advances would involve some sort of chemical modification of biology.”

## Future Directions

Weiss and his team suggest that “the integration of such systems into higher-level organisms and with different cell functions will have practical applications in three-dimensional tissue engineering, biosensing, and biomaterial fabrication” (Basu et al. 2005). One possible use for such a system might lie in the detection of bioweapons – spread a culture of bacteria over a surface – and, with the appropriate control circuit, they will be able to accurately pinpoint the location of any pathogens. Programmed cells could eventually replace artificial tissues, or even organs – current attempts to build such constructions in the laboratory rely on cells arranging themselves around an artificial scaffold. Controlled cellular structure formation could do away with the need for such support: “The way we’re doing tissue engineering, right now, . . . is very unnatural,” argues Weiss. “Clearly cells make scaffolds themselves. If we’re able to program them to do that, we might be able to embed them in the site of injury and have them figure out for themselves what the pattern should be” (Brown 2005). In addition to building structures, others are considering engineering cells to act as miniature drug delivery systems – fighting disease or infection from the *inside*. Adam Arkin and Chris Voigt are currently investigating the use of modified *E. coli* to battle

against cancer tumors, while Jay Keasling and coworkers at Berkeley are looking at engineering circuits into the same bacteria to persuade them to generate a potent antimalarial drug that is normally found in small amounts in wormwood plants.

Clearly, bacterial computing/synthetic biology is still at a relatively early stage in its development, although the field is growing at a tremendous pace. It could be argued, with some justification, that the dominant science of the new millennium may well prove to be at the intersection of biology and computing. As biologist Roger Brent argues, “I think that synthetic biology . . . will be as important to the 21st century as [the] ability to manipulate bits was to the 20th” (Anon 2004).

## Bibliography

### Primary Literature

- Anon (2004) Roger Brent and the alpha project. ACM Ubiquity 5(3)
- Arkin A, Ross J (1994) Computational functions in biochemical reaction networks. Biophys J 67:560–578
- Barkai N, Leibler S (2000) Circadian clocks limited by noise. Nature 403:267–268
- Basu S, Gerchman Y, Collins CH, Arnold FH, Weiss R (2005) A synthetic multicellular system for programmed pattern formation. Nature 434:1130–1134
- Benner SA, Sismour M (2005) Synthetic biology. Nat Rev Genet 6:533–543
- Brown TA (1990) Gene cloning: an introduction, 2nd edn. Chapman and Hall, London
- Brown C (2004) BioBricks to help reverse-engineer life. EE Times, June 11
- Brown S (2005) Command performances. San Diego Union-Tribune, Dec 14
- Crick F (1970) Central dogma of molecular biology. Nature 227:561–563
- Eisenberg A (2000) Unlike viruses, bacteria find a welcome in the world of computing. New York Times, June 1
- Elowitz M, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403:335–338
- Ferber D (2004) Synthetic biology: microbes made to order. Science 303(5655):158–161
- Gardner T, Cantor R, Collins J (2000) Construction of a genetic toggle switch in *Escherichia coli*. Nature 403:339–342
- Geyer CR, Battersby TR, Benner SA (2003) Nucleobase pairing in expanded Watson-crick-like genetic information systems. Structure 11:1485–1498

- Gibbs WW (2004) Synthetic life. *Sci Am* 250:19
- Gravitz L (2004) 10 emerging technologies that will change your world. *MIT Technol Rev* 6:533
- Hasty J (2002) Design then mutate. *Proc Natl Acad Sci* 99(26):16516–16518
- Hopkin K (2004) Life: the next generation. *Sci* 18(19):56
- Jackson DA, Symons RH, Berg P (1972) Biochemical method for inserting new genetic information into DNA of simian virus 40: circular SV40 DNA molecules containing lambda phage genes and the galactose operon of *Escherichia coli*. *Proc Natl Acad Sci* 69:2904–2909
- Jacob F, Monod J (1961) Genetic regulatory mechanisms in the synthesis of proteins. *J Mol Biol* 3:318–356
- Jha A (2005) From the cells up. *Guardian* 91:162
- Kauffman S (1993a) Gene regulation networks: a theory for their global structure and behaviors. *Curr Top Dev Biol* 6:145–182
- Kauffman SA (1993b) The origins of order: self-organization and selection in evolution. Oxford University Press, New York
- Levskaya A, Chevalier AA, Tabor JJ, Simpson ZB, Lavery LA, Levy M, Davidson EA, Scouras A, Ellington AD, Marcotte EM, Voigt CA (2005) Engineering *Escherichia coli* to see light. *Nature* 438:441–442
- Lobban PE, Sutton CA (1973) Enzymatic end-to-end joining of DNA molecules. *J Mol Biol* 78(3):453–471
- Marks P (2005) For ultrasharp pictures, use a living camera. *New Scientist*, 28, Nov 26
- McAdams HH, Arkin A (2000) Genetic regulatory circuits: advances toward a genetic circuit engineering discipline. *Curr Biol* 10:318–320
- McAdams HH, Shapiro L (1995) Circuit simulation of genetic networks. *Science* 269(5224):650–656
- Monod J (1970) Chance and necessity. Penguin, London
- Monod J, Changeux JP, Jacob F (1963) Allosteric proteins and cellular control systems. *J Mol Biol* 6:306–329
- Morton O (2005) Life, reinvented. *Wired* 13(1):168
- Old R, Primrose S (1994) Principles of gene manipulation, an introduction to genetic engineering, 5th edn. Blackwell, Boston
- Ptashne M (2004) A genetic switch, 3rd edn. Phage lambda revisited. Cold Spring Harbor Laboratory Press, Woodbury
- Registry of standard biological parts. <http://parts.mit.edu/>
- Roberts L, Murrell C (eds) (1998) An introduction to genetic engineering. Department of Biological Sciences, University of Warwick
- Strogatz S (2003) Sync: the emerging science of spontaneous order. Penguin, London
- Turing AM (1952) The chemical basis of morphogenesis. *Phil Trans Roy Soc B* 237:37–72
- von Neumann J (1941) The general and logical theory of automata. In: Cerebral mechanisms in behavior. Wiley, New York
- Yokobayashi Y, Weiss R, Arnold FH (2002) Directed evolution of a genetic circuit. *Proc Natl Acad Sci* 99(26):16587–16591
- You L, Cox RS III, Weiss R, Arnold FH (2004) Programmed population control by cell-cell communication and regulated killing. *Nature* 428:868–871

## Books and Reviews

- Alon U (2006) An introduction to systems biology: design principles of biological circuits. Chapman and Hall/CRC
- Amos M (ed) (2004) Cellular computing, Series in Systems biology. Oxford University Press, Oxford
- Amos M (2006) Genesis machines: the new science of biocomputing. Atlantic Books, London
- Benner SA (2003) Synthetic biology: act natural. *Nature* 421:118
- Endy D (2005) Foundations for engineering biology. *Nature* 436:449–453
- Kobayashi H, Kaern M, Araki M, Chung K, Gardner TS, Cantor CR, Collins JJ (2004) Programmable cells: interfacing natural and engineered gene networks. *Proc Natl Acad Sci* 101(22):8414–8419
- Sayler GS, Simpson ML, Cox CD (2004) Emerging foundations: nano-engineering and bio-microelectronics for environmental biotechnology. *Curr Opin Microbiol* 7:267–273



## Immune Computing

Jon Timmis

Department of Electronics, Department of Computer Science, University of York, York, UK

### Article Outline

Glossary

Definition of the Subject

Introduction

What Is an Artificial Immune System?

Current Artificial Immune Systems Biology and Basic Algorithms

Alternative Immunological Theories for AIS

Emerging Methodologies in AIS

Future Directions

Bibliography

### Glossary

**Affinity** Measure or tightness of the binding between an antigen-combining site and an antigenic determinant; the stronger the binding, the higher the affinity.

**Antibody** A soluble protein molecule produced and secreted by B cells in response to an antigen. Antibodies are usually defined in terms of their specific binding to an antigen.

**Antigen** Any substance that when introduced into the body is capable of inducing an immune response.

**Antigen-presenting cells (APCs)** B cells, cells of the monocyte lineage (including macrophages as well as dendritic cells), and various other body cells that present antigen in a form that B and T cells can recognize.

**B cell** White blood cells expressing immunoglobulin molecules on its surface. Also known as B lymphocytes, they are derived

Glossary based on de Castro and Timmis (2002b).

from the bone marrow and develop into plasma cells that are the main antibody secretors.

**Clonal selection theory** A theory that states that the specificity and diversity of an immune response are the result of selection by antigen of specifically reactive clones from a large repertoire of preformed lymphocytes, each with individual specificities.

**Complex (MHC)** Cell-surface molecules (MHC class I and II) that are involved in controlling several aspects of the immune response. MHC genes code for self-markers on all body cells and play a major role in transplantation rejection.

**Dendritic cell** Set of antigen-presenting cells (APCs) present in lymph nodes and the spleen and at low levels in blood, which are particularly active in stimulating T cells.

**Lymph node** Small organs of the immune system, widely distributed throughout the body and linked by lymphatic vessels.

**Lymphocyte** White blood cell found in the blood, tissue, and lymphoid organs.

**Major histocompatibility** A group of genes encoding polymorphic.

**Pathogen** A microorganism that causes disease.

**T cell** White blood cell that orchestrates and/or directly participates in the immune defenses.

### Definition of the Subject

Immune computing, or artificial immune systems (AIS), has recently emerged as a computational intelligence approach that shows great promise. Inspired by the complexity of the immune system, computer scientists and engineers have created systems that in some way mimic or capture certain computationally appealing properties of the immune system, with the aim of building more robust and adaptable solutions. AIS have been defined by de Castro and Timmis (2002b) as:

adaptive systems, inspired by theoretical immunology and observed immune functions, principle and models, which are applied to problem solving.

However, in order to build AIS, an interdisciplinary approach is required that employs modeling of immunology (both mathematical and computational) in order to understand the underlying complexity inherent within the immune system. AIS do not rival their natural counterparts; they do not exhibit the same level of complexity or even perform the same function, but they do capture essential properties of the immune systems that are making them a competitive computational intelligence paradigm.

## Introduction

The immune system is a complex system that undertakes a myriad of tasks. The abilities of the immune system have helped to inspire computer scientists to build systems that *mimic*, in some way, various properties of the immune system. This field of research, artificial immune systems (AIS), has seen the application of immune-inspired algorithms to problems such as robotic control (Krohling et al. 2002), network intrusion detection (Forrest et al. 1997; Kim 2002), fault tolerance (Ayara 2005; Canham and Tyrrell 2002), bioinformatics (Cutello et al. 2004; Niclosia 2004), and machine learning (Kim and Bentley 2002; Knight and Timmis 2003; Watkins et al. 2004), to name a few. To many, trying to mimic how the immune system operates in a computer may seem an unusual thing to do; why then would people in computing wish to do this? The answer is that, from a computational point of view, the immune system has many desirable properties that they would like their computer systems to possess. These properties are such things as robustness, adaptability, diversity, scalability, multiple interactions on a variety of timescales, and so on.

The origins of AIS have its roots in the early theoretical immunology work of Farmer et al. (1986), Perelson (1989), Varela

et al. (1988). These works investigated a number of theoretical immune network models proposed to describe the maintenance of immune memory in the absence of antigen. While controversial from an immunological perspective, these models began to give rise to an interest from the computing community. The most influential people at crossing the divide between computing and immunology in the early days were Hugues Bersini and Stephanie Forrest. It is fair to say that some of the early work by Bersini (1991, 1992) was very well rooted in immunology, and this is also true of the early work by Forrest et al. (1994), Hightower et al. (1995). It was these works that formed the basis of a solid foundation for the area of AIS. In the case of Bersini, he concentrated on the immune network theory, examining how the immune system maintained its memory and how one might build models and algorithms mimicking that property. With regard to Forrest, her work was focused on computer security (in particular network intrusion detection) (Forrest et al. 1997; Hofmeyr and Forrest 2000) and formed the basis of a great deal of further research by the community on the application of immune-inspired techniques to computer security.

At about the same time as Forrest was undertaking her work, other researchers began to investigate the nature of learning in the immune system and how that might be used to create *machine learning* algorithms (Cooke and Hunt 1995). They had the idea that it might be possible to exploit mechanisms of the immune system (in particular the immune network) in learning systems, so they set about doing a proof of concept (Cooke and Hunt 1995). Initial results were very encouraging, and they built on their success by applying the immune ideas to the classification of DNA sequences as either promoter or non-promoter classes (Hunt and Cooke 1996) and the detection of potentially fraudulent mortgage applications (Hunt et al. 1998).

The work of Hunt and Cook spawned more work in the area of immune network-based machine learning over the next few years, notably in Timmis (2000) where the Hunt and Cook system was totally rewritten, simplified,

and applied to unsupervised learning (very similar to cluster analysis). Concurrently, similar work was carried out by de Castro and Von Zuben (2001, 2002), who developed algorithms for use in function optimization and data clustering (the details of these are described in more detail later in the entry). The work of Timmis on machine learning spawned yet more work in the unsupervised learning domain, in trying to perform dynamic clustering (where the patterns in the input data move over time). This was met with some success in works such as Neal (2002) and Wierzchon and Kuzlewska (2002). At the same time, using ideas other than the immune network theory, work by Hart and Ross (2002) used immune-inspired associative memory ideas to track moving targets in databases.

In the supervised learning domain, very little happened until work by Watkins (2001) (later augmented in Watkins et al. (2004)) developed an immune-based classifier known as AIRS. The system developed by Watkins was then adapted into a parallel and distributed learning system in Watkins (2005) and has shown itself to be one of the real success stories of immune-inspired learning (Goodman et al. 2002, 2003; Watkins et al. 2003).

In addition to the work on machine learning, there have been plenty of other activities in AIS over the years. To outline all the applications of AIS and developments over the past 10 years would take a long time, and there are some good review papers in the literature; thus, the reader is directed those (Dasgupta 1999; de Castro and Timmis 2002b; Garrett 2005; Timmis and Knight 2001). In addition to these works, Hart and Timmis (2005) investigated the application areas AIS have been applied to and considered the contribution AIS have made to these areas. Their survey of AIS is not exhaustive, but attempts to produce a picture of the general areas to which they have been applied. Of the 97 papers reviewed, 12 categories were identified to reflect the natural groupings of the papers. These were, in the order of most papers first, clustering/classification, anomaly detection (e.g., detecting faults in engineering systems),

computer security, numerical function optimization, combinatoric optimization (e.g., scheduling), learning, bioinformatics, image processing, robotics (e.g., control and navigation), adaptive control systems, virus detection, and web mining. Hart and Timmis go on to note that these categories can be summarized into three general application areas of learning, anomaly detection, and optimization.

Work in Tarakanov et al. (2003) details an alternative approach to the use of immune metaphors and presents a deterministic immune network approach that is in stark contrast to the work presented here. This work has shown to be exceptionally competitive when compared to other computational intelligence approaches (Tarakanov et al. 2005a, b). Due to a growing amount of work conducted on AIS, the International Conference on Artificial Immune Systems (ICARIS) conference series was started in 2002 (<http://www.artificial-immune-systems.org>) and has operated in subsequent years (Bersini and Carneiro 2006; Jacob et al. 2005; Nicosia et al. 2004; Timmis and Bentley 2002; Timmis et al. 2003). This is the best source of reference material to read in order to grasp the variety of application areas of AIS and also the developments in algorithms and the more theoretical side of AIS.

This remaining entry is organized as follows: section “[What Is an Artificial Immune System?](#)” first discusses the current perception of what AIS are, and we do this in terms of a simple engineering framework; in section “[Current Artificial Immune Systems Biology and Basic Algorithms](#),” we provide a simple overview of the immunology that has served to inspire the development of immune-inspired systems to date, and this is coupled with an outline of the basic algorithms that have been used in AIS to date; in section “[Alternative Immunological Theories for AIS](#),” we turn our attention to alternative immunology (away from the more classic described in the previous section) and focus on the danger theory approach which exploits a fundamental different analogy from immunology that other AIS have done to date and discuss the cognitive immune paradigm as an alternative

approach to the development of AIS; section “[Emerging Methodologies in AIS](#)” reviews new approaches to the development of AIS in the context of a conceptual framework; and finally in section “[Future Directions](#),” we review some of the possible challenges and directions that AIS might follow.

## What Is an Artificial Immune System?

In an attempt to create a common basis for AIS, work in de Castro and Timmis ([2002b](#)) proposed the idea of a framework for engineering AIS. They argued the case for such a framework as the existence of similar frameworks in other biologically inspired approaches, such as artificial neural networks (ANN) and evolutionary algorithms (EAs), has helped considerably with the understanding and construction of such systems. For example, de Castro and Timmis ([2002b](#)) consider a set of artificial neurons, which can be arranged together to form an artificial neural network. In order to acquire knowledge, these neural networks undergo an adaptive process, known as learning or training, which alters (some of) the parameters within the network. Therefore, they argued that in a simplified form, a framework to design an ANN is composed of a set of artificial neurons, a pattern of interconnection for these neurons, and a learning algorithm. Similarly, they argued that in evolutionary algorithms, there is a set of artificial chromosomes representing a population of individuals that iteratively suffer a process of reproduction, genetic

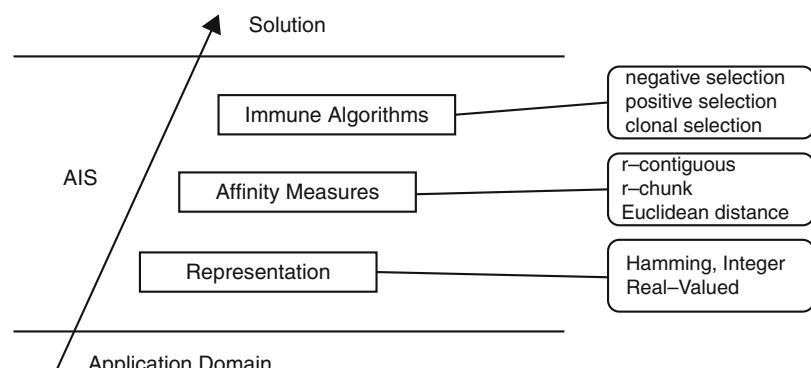
variation, and selection. As a result of this process, a population of evolved artificial individuals arises. A framework, in this case, would correspond to the genetic representation of the individuals of the population, plus the procedures for reproduction, genetic variation, and selection. Therefore, they proposed that a framework to design a biologically inspired algorithm requires, at least, the following basic elements:

- A representation for the components of the system
- A set of mechanisms to evaluate the interaction of individuals with the environment and each other. The environment is usually simulated by a set of input stimuli, one or more fitness function(s), or other means
- Procedures of adaptation that govern the dynamics of the system, i.e., how its behavior varies over time

This framework can be thought of as a layered approach such as the specific framework for engineering AIS of de Castro and Timmis ([2002b](#)) shown in Fig. 1. This framework follows the three basic elements for designing a biologically inspired algorithm just described, where the set of mechanisms for evaluation are the affinity measures and the procedures of adaptation are the immune algorithms. In order to build a system such as an AIS, one typically requires an application domain or target function. From this basis, the way in which the components of the system will be represented is considered. For example, the representation of network traffic may well be

### Immune Computing,

**Fig. 1** AIS layered framework (Adapted from de Castro and Timmis ([2002b](#)))



different than the representation of a real-time embedded system. In AIS, the way in which something is represented is known as *shape space*. There are many kinds of shape space, such as Hamming, real valued, and so on, each of which carries its own bias and should be selected with care (Freitas and Timmis 2003). Once the representation has been chosen, one or more affinity measures are used to quantify the interactions of the elements of the system. There are many possible affinity measures (which are partially dependent upon the representation adopted), such as Hamming and Euclidean distance metrics. Again, each of these has its own bias, and the affinity function must be selected with great care, as it can affect the overall performance (and ultimately the result) of the system (Freitas and Timmis 2003). This was also recently shown experimentally in the case of immune networks, where the affinity function affected the overall outcome of the shape of the network (Hart 2005; Hart and Ross 2004). The final layer involves the use of algorithms, which govern the behavior (dynamics) of the system. Such algorithms include those based on the following immune processes: negative and positive selection, clonal selection, bone marrow, and immune network algorithms.

## Current Artificial Immune Systems Biology and Basic Algorithms

The main developments within AIS have focused on three main immunological theories: clonal selection, immune networks, and negative selection. Researchers in AIS have concentrated, for the most part, on the *learning* and *memory* mechanisms of the immune system inherent in clonal selection and immune networks and the negative selection principle for the generation of *detectors* that are capable of classifying changes in *self*. In this section, we review the immunology that has been capitalized on by the AIS community. We outline the three main immunological theories noted above that have acted as a source of inspiration. At each stage, we review a simple AIS approach that has extracted some feature from

that theory. It is worth noting that, although not covered here, a large effort is currently being made in the AIS community into exploring other immune ideas and mechanisms such as danger theory and innate immunity.

### Immunity

The vertebrate immune system (the one which has been used to inspire the vast majority of AIS to date) is composed of diverse sets of cells and molecules. These work in collaboration with other systems, such as the neural and endocrine, to maintain a steady state of operation within the host: this is termed *homeostasis*. The role of the immune system is typically viewed as one of the protections from infectious agents such as viruses, bacteria, fungi, and other parasites. On the surface of these agents are antigens that allow the identification of the invading agents (pathogens) by the immune cells and molecules, which in turn provoke an immune response. There are two basic types of immunity, innate and adaptive. Innate immunity is not directed toward specific pathogens, but against any pathogen that enters the body. The innate immune system plays a vital role in the initiation and regulation of immune responses, including adaptive immune responses. Specialized cells of the innate immune system evolved so as to recognize and bind to common molecular patterns found only in microorganisms. However, the innate immune system is by no means a complete solution to protecting the body.

Adaptive, or acquired immunity, is directed against specific invaders, with adaptive immune cells being modified by exposure to such invaders. The adaptive immune system mainly consists of lymphocytes, which are white blood cells, more specifically B and T cells. These cells aid in the process of recognizing and destroying specific substances. Any substance that is capable of generating such a response from the lymphocytes is called an antigen or immunogen. Antigens are not the invading microorganisms themselves; they are substances such as toxins or enzymes in the microorganisms that the immune system considers foreign. Adaptive immune responses are normally directed against

the antigen that provoked them and are said to be antigen specific.

### Natural Clonal Selection

The clonal selection theory (CST) (Burnet 1959) is the theory used to explain the basic response of the adaptive immune system to an antigenic stimulus. It establishes the idea that only those cells capable of recognizing an antigenic stimulus will proliferate, thus being selected against those that do not. Clonal selection operates on both T cells and B cells. In the case of B cells, when their antigen receptors (antibodies) bind with an antigen, the B cell becomes activated and begins to proliferate producing new B-cell clones that are an exact copy of the parent B cell. The clones then undergo somatic hypermutation and produce antibodies that are specific to the invading antigen (Berek and Ziegner 1993). After proliferation, B cells differentiate into *plasma cells* or long-lived B *memory cells*. Plasma cells produce large amounts of *antibodies* which will attach themselves to the antigen and act as a type of *tag* for other immune cells to pick up on and remove from the system. This whole process is known as *affinity maturation*.

Memory cells help the immune system to be protective over periods of time. In the normal course of the evolution of the immune system, an organism would be expected to encounter a given antigen repeatedly during its lifetime. The initial exposure to an antigen that stimulates an adaptive immune response is handled by a small number of B cells, each producing antibodies of different affinity. Storing some high-affinity antibody-producing cells (memory cells) from the first infection, so as to form a large initial specific B-cell subpopulation for subsequent encounters, considerably enhances the effectiveness of the immune response to secondary encounters. Such a strategy ensures that both the speed and accuracy of the immune response become successively stronger after each infection.

Autoimmunity is the term used to describe the existence of antigen receptors that recognize the body's own molecules or self-antigens. According to the CST, immune specificity is a

property of immune receptors. When a non-self-antigen is detected, a suitable immune response is elicited and the antigen is destroyed. Thus, the recognition of self-antigen is forbidden, and self-reacting receptors must be deleted.

### Artificial Clonal Selection

Work in de Castro and Von Zuben (2000, 2002) proposes an optimization algorithm, known as CLONALG, inspired by the clonal selection process, as outlined in the previous section. Given a function  $F$ , a population of candidate solutions (antibodies) are evolved to either minimize or maximize the function. Each member of this population is a vector, in a certain shape space, which maps values to the parameters of the function  $F$ . CLONALG exploits the cloning, mutation, and selection mechanisms of clonal selection, to effectively evolve a set of memory cells that contain candidate solutions to the function  $F$ .

CLONALG operates via the following procedure. A population  $P$  is initialized with random vectors, where  $P$  is the set of candidate solutions for the given function. Each member of  $P$  is evaluated against the function, and the highest affinity  $n$  number is selected for cloning, where affinity can be measured as the distance to the optimal value. Clones are produced at a rate proportional to the affinity (so the better the affinity, the more clones are produced). Each clone is subject to a mutation rate, which is inversely proportional to the affinity. These clones are added to  $P$ , and then the  $n$  highest affinity is selected to remain in the population. A number of low-affinity members are then removed from the population and replaced with the same number of randomly generated members. This process is repeated until some convergence criteria are satisfied or a fixed number of iterations have been performed.

Experimentally, CLONALG has been shown to perform well on standard benchmark tests for optimization problems (de Castro and Von Zuben 2002). However, it has not been reported in the literature that CLONALG itself outperforms any well-known technique. Other algorithms similar to CLONALG exist in the literature, such as Kelsey and Timmis (2003) and Cutello

et al. (2004), with comparative studies showing that while CLONALG is effective, better results can be obtained with more specialized versions of the algorithm (Cutello et al. 2004; Nicosia 2004). Indeed, recent work by Cutello et al. (2007a) has shown that a clonal selection-based algorithm using a special aging operator can perform as well as the state of art on certain protein-folding problems.

Clonal selection-based algorithms have also been developed for dynamic environments, reporting good performance (Gaspar and Hirsbrunner 2002; Kelsey et al. 2003; Kim and Bentley 2002). CLONALG has also been adapted for simple pattern recognition problems, but the results from that work are less conclusive (Whitesides and Boncheva 2002). It has also been adapted for more sophisticated learning systems where results are very encouraging indeed for static learning (Goodman et al. 2002; Watkins and Timmis 2004; Watkins et al. 2003) and for dynamic learning (Secker et al. 2003).

### Immune Networks

In a landmark paper for its time, Jerne (1974) proposed that the immune system is capable of achieving immunological memory by the existence of a mutually reinforcing network of B cells. This network of B cells occurs due to the ability of paratopes (molecular portions of an antibody) located on B cells, to match against idiotopes (other molecular portions of an antibody) on other B cells. The binding between idiotopes and paratopes has the effect of stimulating the B cells. This is because the paratopes on B cells react to the idiotopes on similar B cells, as it would an antigen. However, to counter the reaction, there is a certain amount of suppression between B cells which acts as a regulatory mechanism. This interaction of B cells due to the network was said to contribute to a *stable* memory structure and account for the retention of memory cells, even in the absence of antigen. This theory was refined and formalized in successive works by Farmer et al. (1986) and Perelson (1989) and combined with work by Bersini and Varela (1994) was very influential in the development of the immune network-based AIS such

as Hunt and Cooke (1996), Neal (2002), Timmis and Neal (2001), and Timmis et al. (2000).

### Artificial Immune Networks

Based on the work of CLONALG, an algorithm known as aiNET was proposed in de Castro and Von Zuben (2001). aiNET is a simple extension of CLONALG (described above), but exploits interactions between B cells according to the immune network theory. The main difference between the two approaches is that after new clones are integrated into the population, a network suppression function is employed throughout the population to remove cells that have similar affinities (It should be noted that this is a slight departure from the immune network theory, where both suppression and stimulation occur between cells) this facilitates the maintenance of diversity within the population. Recent work by Stibor and Timmis (2007) has shown that aiNET performs better on data that has a more uniform underlying distribution, due to the nature of how aiNET performs the suppression. However, this work did focus on the initial version of aiNET, and there are recent variants of aiNET that seem to perform much better.

aiNET was initially designed for data clustering, but has been extended over the years, as a hierarchical clustering tool in de Castro and Timmis (2002c) and through hybridization with fuzzy systems methods by Bezerra et al. (2005). In the last paper, aiNET was augmented to take into account an adaptive radius measure instead of a fixed radius for B-cell matching. This leads to a much improved version of aiNET, being able to achieve better separation of the data, forming clusters in less time. Work by de Castro and Timmis (2002a) adapted aiNET for multimodal function optimization. In that paper, aiNET was also modified to be applied to the same optimization problems as CLONALG and was shown to have greatly improved performance over CLONALG, but this is not as comparable to other clonal selection-based systems (Timmis and Edmonds 2004; Timmis et al. 2004). However, it was recently identified that if careful thought was given to the optimization problem, the basic aiNET algorithm can be augmented to

give significant gains in performance (Andrews and Timmis 2005b).

### Negative Selection

Negative selection is a process of *selection* that takes place in the thymus gland. T cells are produced in the bone marrow and, before they are released into the lymphatic system, undergo a maturation process in the thymus gland. The maturation of the T cells is conceptually very simple. T cells are exposed to self-proteins in a binding process. If this binding activates the T cell, then the T cell is killed; otherwise, it is allowed into the lymphatic system. This process of *censoring* prevents cells that are reactive to *self* from entering the lymph system, thus endowing (in part) the host's immune system with the ability to distinguish between self and nonself agents.

### Artificial Negative Selection

The negative selection principle inspired Forrest et al. (1994) to propose a negative selection algorithm to detect data manipulation caused by computer viruses. The basic idea is to generate a number of detectors in the complementary space and then to apply these detectors to classify new (unseen) data as self (no data manipulation) or nonself (data manipulation). In the negative selection algorithm as proposed by Forrest et al., we can define self as a set  $S$  of elements of length  $l$  in shape space and then generate a set  $D$  of detectors, such that each fails to match any element in  $S$ . With these detectors, we monitor a continual data stream for any changes, by continually matching the detectors in  $D$  against the stream. This work spawned a great deal of investigations into the use of negative selection for intrusion detection, with early work meeting with some success (Forrest et al. 1997) and this being built on in later years (Balthrop et al. 2002; Esponda et al. 2004; Hofmeyr and Forrest 1999, 2000). The work on negative selection has been dominating in AIS. A great deal of work has gone into investigating various alternatives of representations (Dasgupta and Nino 2000; González and Dagupta 2003; González et al. 2003), techniques for estimating detector coverage (González et al. 2002; Ji and Dasgupta 2004a, b,

2005), and applications of the said technique (Ayara et al. 2002; Dasgupta and Forrest 1995; Dasgupta and Majumdar 2002; Ebner et al. 2002; Ji and Dasgupta 2006; Ji et al. 2006; Kim and Bentley 2001a, b; Singh 2002) to name only a few. However, later works began to highlight certain limitations of the approach (Stibor et al. 2004, 2005a, b, 2006) with regard to scalability issues and applicability of the technique to classification. In addition, work in Freitas and Timmis (2003) outlines the need to consider carefully the application domain when developing AIS, and they give particular attention to negative selection. They review the role AIS have played in the development of a number of machine learning tasks, including that of negative selection. However, Freitas and Timmis point out that there is a lack of appreciation for possible inductive bias within algorithms and positional bias within the choice of representation and affinity measures that comes from not carefully applying not only negative selection, but other algorithms as well.

### Alternative Immunological Theories for AIS

#### Danger Theory

In recent years there has been a growing interest in the mechanisms of innate immune system in immunology (Germain 2004). Up to this point, AIS had concentrated solely on the adaptive immune systems, but the danger theory proposed by Matzinger (1997, 2002) has caught the interest of the AIS practitioner in recent years as a compliment to the adaptive.

The danger theory attempts to explain the nature and workings of the immune response in a way different to the more traditional clonal selection view. Matzinger criticizes this idea, as she states that observations demonstrate that it may sometimes be necessary for the body to attack itself and conversely the immune system may not attack cells it knows to be foreign (this is not possible under the classical clonal selection theory). Matzinger argues a more plausible way to describe the triggering of an immune response

is a reaction to a stimulus the body considers harmful. This might be seen as a very small change, but in reality this is a real shift in thinking about how the immune system responds to pathogens. In essence, this model allows for foreign and immune cells to exist together, a situation impossible in the traditional standpoint. When under attack, cells dying unnaturally may release a danger signal that disperses to cover a small area around that cell: a danger area. It is within this and only within this area that the immune system becomes active and will concentrate its attack against any antigen within it. There is still much debate in the immunological world as to whether the danger theory is a plausible explanation for observed immune function, but it is proving to be an interesting theory nonetheless, and it affects the thinking in the AIS world. As we have discussed in the previous section, the idea of using the innate immune systems, in particular the danger theory, has started to become popular. The first to propose the idea was Aickelin et al. (2003) in the context of using such danger theory ideas in the context of network intrusion detection. Here the authors discussed how one might introduce the notion of *danger areas* in networks which might be indicated by unusual behavior without having to define a priori what those behaviors were (a large departure from the current way of network intrusion detection using AIS, where unusual behavior was used to train the system). This work was then extended notably in Greensmith et al. (2005, 2006) where the dendritic cell algorithm was proposed and Bentley et al. (2005) where the idea of artificial tissue was proposed. Work in Greensmith et al. (2006) describes how dendritic cell (DC), which is a cell considered to be part of the innate immune system, performs a function that in effect controls the adaptive immune response when under attack. What a DC does depends on the signals that it receives within the tissue: these might be danger signals, PAMPs (pathogenic-associated molecular patterns), safe signals, and inflammatory cytokines. The DC will mature into different states depending on the concentration of these signals, and the state of the DC influences the response of the T cell to which the DC is

presenting the antigen. Based on these ideas, a DC-inspired algorithm has been developed and tested within a tissue environment developed in Twycross and Aickelin (2006). The application area was that of anomaly detection, and the DC had to identify if certain types of behavior on a computer were anomalous or not: no predefined knowledge of what constitutes anomalous is required, but what is required is a definition of what constitutes dangerous behaviors for various variables. Results reported in Greensmith et al. (2006) would seem to indicate that the DC algorithm is capable of identifying anomalous behavior (processes that were considered not to be normal) over time. However, this is still quite preliminary work, and the system needs to be baselined against a state-of-the-art-type system in order to fully see the contribution this approach can bring.

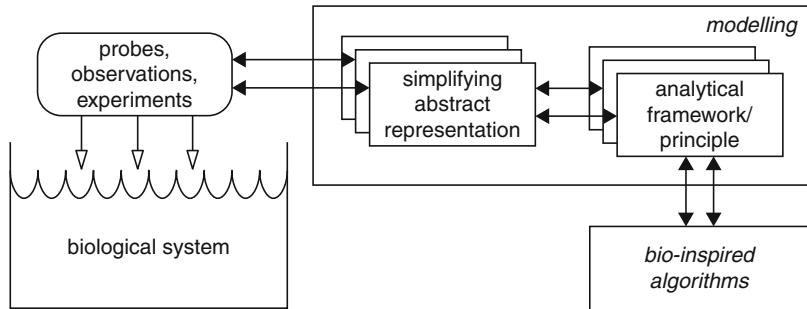
### Cognitive Immunology

Much like the paradigm shift of danger theory, within AIS, recent attention has been paid to the cognitive immune paradigm proposed by Cohen (2000). Notably, work in Andrews and Timmis (2005a) discusses how Cohen views the immune system as a cognitive system, capable of detection, cognition, and decision making, and argues that the primary role of the immune system is not protection (as is considered by say the clonal selection theory), but one of body maintenance. Andrews and Timmis (2005a) say that in Cohen's view, removal of pathogen is beneficial to the health of the body, and thus defense against pathogen is considered to be just a special case of body maintenance. In order to carry out body maintenance, the immune system must be able to detect the current state of the body's tissues and elicit an appropriate response.

According to the clonal selection theory, immune specificity is a property of the somatically generated immune receptors of the T and B cells, which both initiates and regulates the immune response. Initiation is achieved via the binding between an antigen and a receptor that is specific to it. As stated by Andrews and Timmis (2005a), Cohen, however, points out that immune receptors are intrinsically degenerate, i.e., they

**Immune Computing,**

**Fig. 2** Conceptual framework for the development of AIS  
(Adapted from Stepney et al. (2006))



can bind more than one ligand. Immune specificity, therefore, cannot be purely dependent on molecular binding as no one receptor can be specific to a single antigen. Instead, affinity, the strength of binding between a receptor and its ligand, is a matter of degree. Indeed, as a follow-up to their work, Andrews and Timmis (2006) outline a simple computational model that demonstrates degenerate recognition in the context of a lymph node, and work in Mendoz et al. (2007) has also investigated degeneracy in the context of pattern recognition and begun to develop high-level AIS as a result.

### Emerging Methodologies in AIS

The methodology in which AIS are built has been addressed by work in Stepney et al. (2006). This paper proposes a conceptual framework that allows for the development of more biologically grounded AIS, through the adoption of an interdisciplinary approach. As will be clear from the article, metaphors that have been employed have typically been simple, but somewhat effective. However, as proposed in Stepney et al. (2006), through greater interaction between computer scientists, engineers, biologists, and mathematicians, better insights into the workings of the immune system and the applicability (or otherwise) of the AIS paradigm will be gained. These interactions should be rooted in a sound methodology in order to fully exploit the synergy.

These interactions should be rooted in a sound methodology in order to fully exploit the synergy.

The authors argue that rather than going straight from observing the biology and then to the development of an algorithm, a more principled approach is required to adequately capture the required properties of the biological system in the engineered counterpart. The methodology is one of abstraction, as seen in Fig. 2. The first step is to observe the biological system through a means of experimentation and analysis. From there it is possible to create a mathematical model of the biosystem: a relatively detailed model of the system. However, these mathematical models may be too complex to solve, and therefore another level of abstraction is required to gain further insight into the interactions within the system and overall systems dynamics. Therefore, these mathematical models are then used to derive a more abstract computational model: the model can be executed and analyzed for properties that are desired in the engineered system we wish to construct, these can be encompassed into an analytical framework and design principles and high-level abstraction of algorithms, and systems can be developed abstract from any application area. This is then instantiated in the application area, being tailored to the specific requirement of that application area. The result is a well-grounded bio-inspired algorithm that is understood better on a theoretical level and captures the *relevant* biological properties for the required application. As part of this process, the authors suggest a second stage of development, and that is to create *meta-frameworks* which cut across a number of frameworks that have arisen as part of the initial development. At this stage it is possible to ask unifying questions across these

systems that are concerned with complexity of the system:

1. Openness: How much openness is required in the system? Biological systems do not stop computing; therefore, should our computations stop?
2. Diversity: How much diversity is required in the system to attain the performance we require, and how many different types of actors are needed?
3. Interaction: How should these agents interact? At what timescale and what should they communicate?
4. Structure: Biological systems operate on a number of levels. How many levels are needed in our artificial systems and are suitable levels of a hierarchy required?
5. Scale: Biological systems operate on vast scales, rather different from a typical immune-inspired algorithm. How many actors are required in the system to achieve the desired complexity?

- They will be required to perform *life-long learning*. (Hart and Timmis 2005)

It is apparent that, despite the success of some applications of AIS, all AIS to date fail to fully capture the complex operation of the immune system. What has changed is the increased scope of immunological theories that those working with AIS take inspiration from. For example, in their summaries of the future for AIS, both Garrett (2005) and Hart and Timmis (2005) point toward an increased emphasis on the innate and homeostatic functions of the immune system as possible areas for AIS exploitation. In addition to the increased scope of AIS, there has been a recent and healthy rise in investigating the theoretical workings of various immune algorithms (Clark et al. 2005; Cutello et al. 2007b; Stibor et al. 2005a). In a recent position paper, Timmis (2007) argues that the area of AIS has reached something of an impasse. They discuss a number of challenges to the AIS community which they believe will stimulate discussion and help move the area forward:

## Future Directions

With this may come a better understanding of how to apply AIS and not fall into the traps highlighted by Freitas and Timmis (2003). A recent paper by Hart and Timmis (2005) highlights the fact that to date, the development of AIS has been *scattergun*, i.e., many applications have been tried without a great deal of thought. Indeed, this paper provides a detailed overview of the many application areas that AIS have tried, and this will not be repeated here: the interested reader should consult that paper. The authors go on to propose a number of properties that they feel any AIS should have and that these properties may help guide the type of application they could be applied to:

- They will exhibit *homeostasis*.
- They will benefit from interactions between *innate* and *adaptive* immune models.
- They will consist of *multiple, interacting, communicating* components.
- Components can be easily and naturally *distributed*.

**Challenge 1: To Develop Novel and Accurate Metaphors and Be a Benefit to Immunology.** Typically naive approaches to extracting metaphors from the immune system have been taken. This has occurred as an accident of history, and AIS has slowly drifted away from its immunological roots. Time is now ripe for greater interaction with immunologists and mathematicians to undertake specific experimentation and create useful models, all of which can be used as a basis for abstraction into powerful algorithms.

**Challenge 2: To Develop a Theoretical Basis for AIS.** Much work on AIS has concentrated on simple extraction of metaphors and direct application. Despite the creation of a framework for developing AIS, it still lacks significant formal and theoretical underpinning. AIS have been applied to a wide variety of problem domains, but a significant effort is still required to understand the nature of AIS and where they are best applied. For this, a more theoretical understanding is required.

**Challenge 3: To Consider the Application of AIS.** Work to date in the realm of AIS has mainly concentrated on what other paradigms do, such as simple optimization, learning and the like. This has happened as an accident of history and whilst productive, the time is here to look for the killer application of AIS, or, if not that

radical, then applications where the benefit of adopting the immune approach is clear.

**Challenge 4: To Consider the Integration of Immune and Other Systems.** The immune system does not work in isolation. Therefore, attention should not only be paid to the potential of the immune system as inspiration, but also other systems with which the immune system interacts, in particular the neural and endocrine systems. This will pave the way for a greater understanding of the role and function of the immune system and develop a new breed of immune inspired algorithms. (Timmis 2007)

To be sure, some of these challenges are now being addressed within the community. The area of AIS is a dynamic and vibrant area of research, it is inherently interdisciplinary in nature and great lessons can be learnt between various communities, and they can all benefit from successful interactions.

## Bibliography

### Primary Literature

- Aickelin U, Bentley P, Cayzer S, Kim J, McLeod J (2003) Danger theory: the link between AIS and IDS? In: Timmis J, Bentley P, Hart E (eds) Proceedings of the 2nd international conference on artificial immune systems (ICARIS). LNCS, vol 2787. Springer, Berlin, pp 147–155
- Andrews PS, Timmis J (2005a) Inspiration for the next generation of artificial immune systems. In: Jacob C, Pilat M, Bentley P, Timmis J (eds) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin, pp 126–138
- Andrews PS, Timmis J (2005b) On diversity and artificial immune systems: incorporating a diversity operator into aiNet. In: Proceedings of the international conference on natural and artificial immune systems (NAIS05). LNCS, vol 391. Springer, Berlin, pp 293–306
- Andrews PS, Timmis J (2006) A computational model of degeneracy in a lymph node. In: Bersini H, Carneiro J (eds) Proceedings of 5th international conference on artificial immune systems. LNCS. Springer, Berlin, pp 164–177
- Ayara M (2005) An immune inspired solution for adaptable error detection in embedded systems. PhD thesis, University of Kent
- Ayara M, Timmis J, de Lemos R, de Castro L, Duncan R (2002) Negative selection: how to generate detectors. In: Proceedings of the 1st international conference on artificial immune systems (ICARIS-2002). University of Kent, Canterbury, pp 89–98
- Balthrop J, Forrest S, Glickman M (2002) Revisiting lisys: parameters and normal behavior. In: Proceedings of congress on evolutionary computation (CEC). IEEE Press, pp 1045–1050
- Bentley PJ, Greensmith J, Ujjin S (2005) Two ways to grow tissue for artificial immune systems. In: Jacob C, Pilat M, Bentley P, Timmis J (eds) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin, pp 139–152
- Berek C, Ziegner M (1993) The maturation of the immune response. *Immunol Today* 14:200–402
- Bersini H (1991) Immune network and adaptive control. In: Proceedings of the 1st European conference on artificial life (ECAL). MIT Press, Cambridge, pp 217–226
- Bersini H (1992) Reinforcement and recruitment learning for adaptive process control. In: Proceedings of the international Fuzzy association conference (IFAC/IFIP/IMACS) on artificial intelligence in real time control, pp 331–337
- Bersini H, Carneiro J (eds) (2006) Proceedings of 5th international conference on artificial immune systems. LNCS, vol 4163. Springer, Berlin
- Bersini H, Varela F (1994) The immune learning mechanisms: recruitment, reinforcement and their applications. Chapman Hall, Austin
- Bezerra G, Barra T, de Castro LN, Von Zuben F (2005) Adaptive radius immune algorithm for data clustering. In: Jacob C, Pilat M, Bentley P, Timmis J (eds) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin, pp 290–303
- Burnet FM (1959) The clonal selection theory of acquired immunity. Cambridge University Press, Cambridge
- Canham RO, Tyrrell AM (2002) A multilayered immune system for hardware fault tolerance within an embryonic array. In: Timmis J, Bentley P (eds) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury, pp 3–11
- Clark E, Hone A, Timmis J (2005) A Markov chain model of the B-cell algorithm. In: Jacob C, Pilat M, Bentley P, Timmis J (eds) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin, pp 318–330
- Cohen IR (2000) Tending Adam's garden: evolving the cognitive immune self. Elsevier Academic, London
- Cooke D, Hunt J (1995) Recognising promoter sequences using an artificial immune system. In: Proceedings of intelligent systems in molecular biology. AAAI Press, pp 89–97
- Cutello V, Nicosia G, Parvone M (2004) Exploring the capability of immune algorithms: a characterisation of hypermutation operators. In: Nicosia G, Cutello V, Bentley P, Timmis J (eds) Proceedings of the 3rd

- international conference on artificial immune systems (ICARIS). LNCS, vol 3239. Springer, Berlin, pp 263–276
- Cutello V, Nicosia G, Pavone M, Timmis J (2007a) An immune algorithm for protein structure prediction on lattice models. *IEEE Trans Evol Comput* 11(1):101–117
- Cutello V, Nicosia G, Oliveto P, Romeo M (2007b) On the convergence of immune algorithms. In: Proceedings of foundations of computational intelligence. IEEE Press, pp 409–416
- Dasgupta D (1999) Artificial immune systems and their applications. Springer, Berlin
- Dasgupta D, Forrest S (1995) Tool breakage detection in milling operations using a negative selection algorithm. Technical report no CS95-5. Department of Computer Science, University of New Mexico
- Dasgupta D, Majumdar NS (2002) Anomaly detection in multidimensional data using negative selection algorithm. In: Proceedings of congress on evolutionary computation (CEC). IEEE Press, Honolulu, pp 1039–1044
- Dasgupta D, Nino F (2000) A comparison of negative and positive selection algorithms in novel pattern detection. In: Proceedings of the IEEE international conference on systems, man and cybernetics (SMC), Nashville, 8–11 Oct
- de Castro LN, Timmis J (2002a) An artificial immune network for multi modal optimisation. In: Proceedings of the world congress on computational intelligence WCCI. IEEE Press, Honolulu, pp 699–704
- de Castro LN, Timmis J (2002b) Artificial immune systems: a new computational intelligence approach. Springer, Berlin
- de Castro LN, Timmis J (2002c) Hierarchy and convergence of immune networks: basic ideas and preliminary results. In: Timmis J, Bentley P (eds) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury, pp 231–240
- de Castro LN, Von Zuben FJ (2000) The clonal selection algorithm with engineering applications. In: GECCO workshop on artificial immune systems and their applications, pp 36–37
- de Castro LN, Von Zuben FJ (2001) aiNet: an artificial immune network for data analysis. Idea Group Publishing, pp 231–259
- de Castro LN, Von Zuben FJ (2002) Learning and optimization using the clonal selection principle. *IEEE Trans Evol Comput* 6(3):239–251
- Ebner M, Breunig H-G, Albert J (2002) On the use of negative selection in an artificial immune system. In: Proceedings of genetic and evolutionary computation conference (GECCO). Morgan Kaufman Publishers, San Francisco, pp 957–964
- Esponda F, Forrest S, Helman P (2004) A formal framework for positive and negative detection schemes. *IEEE Trans Syst Man Cybern B* 34(1):357–373
- Farmer JD, Packard NH, Perelson AS (1986) The immune system, adaptation, and machine learning. *Physica D* 22:187–204
- Forrest S, Perelson AS, Allen L, Cherukuri R (1994) Self-nonself discrimination in a computer. In: Proceedings of the IEEE symposium on research security and privacy. IEEE Press, pp 202–212
- Forrest S, Hofmeyr S, Somayaji A (1997) Computer immunology. *Commun ACM* 40(10):88–96
- Freitas A, Timmis J (2003) Revisiting the foundations of artificial immune systems: a problem oriented perspective. In: Timmis J, Bentley P, Hart E (eds) Proceedings of the 2nd international conference on artificial immune systems (ICARIS). LNCS, vol 2787. Springer, Berlin, pp 229–241
- Garrett SM (2005) How do we evaluate artificial immune systems? *Evol Comput* 13(2):145–177
- Gaspar A, Hirsbrunner B (2002) From optimization to learning in learning in changing environments: The pittsburgh immune classifier system. In: Timmis J, Bentley P (eds) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury, pp 190–199
- Germain RN (2004) An innately interesting decade of research in immunology. *Nat Med* 10:1307–1320
- González F, Dagupta D (2003) Anomaly detection using real-valued negative selection. *Genet Program Evolvable Mach* 4(4):383–403
- González F, Dasgupta D, Kozma R (2002) Combining negative selection and classification techniques for anomaly detection. In: IEEE congress on evolutionary computation. IEEE, pp 705–710
- González F, Dasgupta D, Gómez J (2003) The effect of binary matching rules in negative selection. In: Genetic and evolutionary computation – GECCO-2003. Lecture notes in computer science, vol 2723. Springer, Chicago, pp 195–206
- Goodman D, Boggess L, Watkins A (2002) Artificial immune system classification of multiple-class problems. In: Proceedings of intelligent engineering systems. ASME, pp 179–184
- Goodman D, Boggess L, Watkins A (2003) An investigation into the source of power for AIRS, an artificial immune classification system. In: Proceedings of the international joint conference on neural networks. IEEE, pp 1678–1683
- Greensmith J, Aickelin U, Cayzer S (2005) Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. In: Jacob C, Pilat M, Bentley P, Timmis J (eds) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin, pp 153–167
- Greensmith J, Aickelin U, Twycross J (2006) Articulation and clarification of the dendritic cell algorithm. In: Bersini H, Coutinho A (eds) Proceedings of the 5th international conference on artificial immune systems. LNCS, vol 4163. Springer, Berlin
- Hart E (2005) Not all balls are round: an investigation of alternative recognition-region shapes. In: Jacob C,

- Pilat M, Bentley P, Timmis J (eds) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin, pp 29–42
- Hart E, Ross P (2002) Exploiting the analogy between immunology and sparse distributed memories: a system for clustering non-stationary data. In: Timmis J, Bentley P (eds) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury, pp 49–58
- Hart E, Ross P (2004) Studies on the implications of shape-space models for idiotypic networks. In: Nicosia G, Cutello V, Bentley P, Timmis J (eds) Proceedings of the 3rd international conference on artificial immune systems (ICARIS). LNCS, vol 3239. Springer, Berlin, pp 413–426
- Hart E, Timmis J (2005) Application areas of AIS: the past, the present and the future. In: Jacob C, Pilat M, Bentley P, Timmis J (eds) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin, pp 483–497
- Hightower RR, Forrest SA, Perelson AS (1995) The evolution of emergent organization in immune system gene libraries. In: Proceedings of the 6th international conference on genetic algorithms. Morgan Kaufmann, pp 344–350
- Hofmeyr S, Forrest S (1999) Immunity by design: an artificial immune system. In: Proceedings of genetic and evolutionary computation conference (GECCO), pp 1289–1296
- Hofmeyr S, Forrest S (2000) Architecture for an artificial immune system. *Evol Comput* 7(1):1289–1296
- Hunt J, Cooke D (1996) Learning using an artificial immune system. *J Netw Comput Appl* 19:189–212
- Hunt J, Timmis J, Cooke D, Neal M, King C (1998) JISYS: development of an artificial immune system for real-world applications. In: Dasgupta D - (ed) Artificial immune systems and their applications. Springer, Berlin, pp 157–186
- Jacob C, Pilat M, Bentley P, Timmis J (eds) (2005) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin
- Jerne NK (1974) Towards a network theory of the immune system. *Ann Immunol (Inst Pasteur)* 125C:373–389
- Ji Z, Dasgupta D (2004a) Augmented negative selection algorithm with variable-coverage detectors. In: IEEE congress on evolutionary computation. IEEE, pp 1081–1088
- Ji Z, Dasgupta D (2004b) Real-valued negative selection algorithm with variable-sized detectors. In: Genetic and evolutionary computation – GECCO-2004, part I. Lecture notes in computer science, vol 3102. Springer, Seattle, pp 287–298
- Ji Z, Dasgupta D (2005) Estimating the detector coverage in a negative selection algorithm. In: Proceedings of genetic and evolutionary computation conference (GECCO). ACM Press, pp 281–288
- Ji Z, Dasgupta D (2006) Applicability issues of the real-valued negative selection algorithms. In: Proceedings of genetic and evolutionary computation conference (GECCO). ACM Press, pp 111–118
- Ji Z, Dasgupta D, Yang Z, Teng H (2006) Analysis of dental images using artificial immune systems. In: Proceedings of congress on evolutionary computation (CEC). IEEE Press, pp 528–535
- Kelsey J, Timmis J (2003) Immune inspired somatic contiguous hypermutation for function optimisation. In: Proceedings of genetic and evolutionary computation conference (GECCO). LNCS, vol 2723. Springer, Berlin, pp 207–218
- Kelsey J, Timmis J, Hone A (2003) Chasing chaos. In: Proceedings of congress on evolutionary computation (CEC). IEEE, Canberra, pp 89–98. <http://www.cs.ukc.ac.uk/pubs/2002/1504>
- Kim J (2002) Integrating artificial immune algorithms for intrusion detection. PhD thesis, UCL
- Kim J, Bentley PJ (2001a) An evaluation of negative selection in an artificial immune system for network intrusion detection. In: Proceedings of genetic and evolutionary computation conference (GECCO). Morgan Kaufmann, San Francisco, pp 1330–1337
- Kim J, Bentley PJ (2001b) Towards an artificial immune system for network intrusion detection: an investigation of clonal selection with negative selection operator. In: Proceedings of congress on evolutionary computation (CEC). Morgan Kaufmann, Seoul, pp 1244–1252
- Kim J, Bentley PJ (2002) Immune memory in the dynamic clonal selection algorithm. In: Timmis J, Bentley P (eds) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury, pp 59–67
- Knight T, Timmis J (2003) A multi-layered immune inspired machine learning algorithm. In: Lotfi A, Garibaldi M (eds) Applications and science in soft computing. Springer, Berlin, pp 195–202. <http://www.cs.kent.ac.uk/pubs/2003/1760>
- Krohling R, Zhou Y, Tyrrell A (2002) Evolving FPGA-based robot controllers using an evolutionary algorithm. In: Timmis J, Bentley P (eds) (2002) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury, pp 41–46
- Matzinger P (1997) An innate sense of danger. *Semin Immunol* 10(5):399–415
- Matzinger P (2002) The danger model: a renewed sense of self. *Science* 296:301–305
- Mendoza M, Timmis J, Andrews PS, Davies M (2007) The immune system in pieces: computational lessons from degeneracy in the immune system. In: Fogel DB (ed) Proceedings of foundations of computational intelligence. IEEE Press, pp 394–400

- Neal M (2002) An artificial immune system for continuous analysis of time-varying data. In: Timmis J, Bentley P (eds) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury, pp 76–85
- Nicosia G (2004) Immune algorithms for optimization and protein structure prediction. PhD thesis, University of Catania
- Nicosia G, Cutello V, Bentley P, Timmis J (eds) (2004) Proceedings of the 3rd international conference on artificial immune systems (ICARIS). LNCS, vol 3239. Springer, Berlin
- Perelson AS (1989) Immune network theory. *Immunol Rev* 110:5–36
- Secker A, Freitas A, Timmis J (2003) AISEC: an artificial immune system for email classification. In: Proceedings of congress on evolutionary computation (CEC). IEEE Press, pp 131–139
- Singh S (2002) Anomaly detection using negative selection based on the r-contiguous matching rule. In: Timmis J, Bentley PJ (eds) Proceedings of the 1st international conference on artificial immune systems ICARIS. University of Kent at Canterbury Printing Unit, University of Kent at Canterbury, pp 99–106. <http://www.aber.ac.uk/icaris-2002>
- Stepney S, Smith R, Timmis J, Tyrrell A, Neal M, Hone A (2006) Conceptual frameworks for artificial immune systems. *Int J Unconv Comput* 1(3):315–338
- Stibor T, Timmis J (2007) An investigation into the compression quality of ainet. In: Fogel D (ed) Proceedings of foundations of computational intelligence. IEEE Press
- Stibor T, Bayarou KM, Eckert C (2004) An investigation of R-chunk detector generation on higher alphabets. In: Proceedings of genetic and evolutionary computation conference (GECCO). LNCS, vol 3102. Springer, Berlin, pp 299–307
- Stibor T, Timmis J, Eckert C (2005a) A comparative study of real-valued negative selection to statistical anomaly detection techniques. In: Jacob C, Pilat M, Bentley P, Timmis J (eds) Proceedings of the 4th international conference on artificial immune systems (ICARIS). LNCS, vol 3627. Springer, Berlin, pp 262–275
- Stibor T, Mohr P, Timmis J, Eckert C (2005b) Is negative selection appropriate for anomaly detection? In: Proceedings of genetic and evolutionary computation conference (GECCO). ACM Press
- Stibor T, Timmis J, Eckert C (2006) Generalization regions in hamming negative selection. In: Intelligent information processing and web mining. Advances in soft computing. Springer, Berlin, pp 447–456
- Tarakanov AO, Skormin VA, Sokolova SP (2003) Immunocomputing: principles and applications. Springer, New York
- Tarakanov AO, Goncharova LB, Tarakanov OA (2005a) A cytokine formal immune network. In: Advances in artificial life, 8th European conference, ECAL 2005, Canterbury, 5–9 Sept 2005, pp 510–519
- Tarakanov AO, Kvachev SV, Sukhorukov AV (2005b) A formal immune network and its implementation for on-line intrusion detection. In: MMM-ACNS, pp 394–405
- Timmis J (2000) Artificial immune systems: a novel data analysis technique inspired by the immune system. PhD thesis, University of Wales
- Timmis J (2007) Artificial immune systems: today and tomorrow. *Nat Comput* 6(1):1–18
- Timmis J, Bentley P (eds) (2002) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury
- Timmis J, Edmonds C (2004) A comment on opt-AINet: an immune network algorithm for optimisation. In: Proceedings of genetic and evolutionary computation conference (GECCO). LNCS, vol 3102. Springer, Berlin, pp 308–317
- Timmis J, Knight T (2001) Artificial immune systems: using the immune system as inspiration for data mining. In: Abbas H, Ruhul A, Sarker A, Newton S (eds) Data mining: a heuristic approach. Idea Group, pp 209–230
- Timmis J, Neal M (2001) A resource limited artificial immune system for data analysis. *Knowl Based Syst* 14(3–4):121–130
- Timmis J, Neal M, Hunt J (2000) An artificial immune system for data analysis. *Biosystems* 55(1/3):143–150
- Timmis J, Bentley P, Hart E (eds) (2003) Proceedings of the 2nd international conference on artificial immune systems (ICARIS). LNCS, vol 2787. Springer, Berlin
- Timmis J, Edmonds C, Kelsey J (2004) Assessing the performance of two immune inspired algorithms and a hybrid genetic algorithm for function optimisation. In: Proceedings of congress on evolutionary computation (CEC), vol 1. IEEE, pp 1044–1051
- Twycross J, Aickelin U (2006) Libtissue: implementing innate immunity. In: Proceedings of the congress on evolutionary computation. IEEE Press, pp 499–506
- Varela F, Coutinho A, Dupire B, Vaz N (1988) Cognitive networks: immune, neural and otherwise. *J Theor Immunol* 2:359–375
- Watkins A (2001) AIRS: a resource limited artificial immune classifier. Master's thesis, Mississippi State University
- Watkins A (2005) Exploiting immunological metaphors in the development of serial, parallel and distributed learning algorithms. PhD thesis, University of Kent
- Watkins A, Timmis J (2004) Exploiting parallelism inherent in AIRS, an artificial immune classifier. In:

- Nicosia G, Cutello V, Bentley P, Timmis J (eds) Proceedings of the 3rd international conference on artificial immune systems (ICARIS). LNCS, vol 3239. Springer, Berlin, pp 427–438
- Watkins A, Xintong B, Phadke A (2003) Parallelizing an immune-inspired algorithm for efficient pattern recognition. In: Intelligent engineering systems through artificial neural networks: smart engineering system design: neural networks, fuzzy logic, evolutionary programming, complex systems and artificial life. ASME Press, pp 224–230
- Watkins A, Timmis J, Boggess L (2004) Artificial immune recognition system (AIRS): an immune inspired supervised machine learning algorithm. *Genet Program Evolvable* 5(3):291–318. <http://www.cs.kent.ac.uk/pubs/2004/1634>
- Whitesides GM, Boncheva M (2002) Beyond molecules: self-assembly of mesoscopic and macroscopic components. *Proc Natl Acad Sci U S A* 99(8):4769–4774
- Wierzchon S, Kuzelewska U (2002) Stable clusters formation in an artificial immune system. In: Timmis J, Bentley P (eds) Proceedings of the 1st international conference on artificial immune systems (ICARIS). University of Kent, Canterbury, pp 68–75

### Books and Reviews

- Cohen I, Segal L (2001) Design principles for the immune system and other distributed autonomous systems. SFT. Oxford University Press, New York
- Ishida Y (2004) Immunity-based systems: a design perspective. Springer, New York



---

## Membrane Computing: Power and Complexity

Marian Gheorghe<sup>1</sup>, Andrei Păun<sup>2</sup>, Sergey Verlan<sup>3</sup> and Gexiang Zhang<sup>4,5,6</sup>

<sup>1</sup>School of Electrical Engineering and Computer Science, University of Bradford, Bradford, West Yorkshire, UK

<sup>2</sup>Department of Computer Science, University of Bucharest, Bucharest, Romania

<sup>3</sup>LACL, Université Paris Est Créteil, Créteil, France

<sup>4</sup>Robotics Research Center, Xihua University, Chengdu, Sichuan, China

<sup>5</sup>Key Laboratory of Fluid and Power Machinery, Xihua University, Ministry of Education, Chengdu, Sichuan, China

<sup>6</sup>School of Electrical Engineering, Southwest Jiaotong University, Chengdu, Sichuan, China

### Article Outline

Glossary  
Introduction  
Types of P Systems  
General Functioning of P Systems  
Examples of P Systems  
Computing Power  
Computational Efficiency  
Future Directions  
Bibliography

### Glossary

**Computational completeness** A computing model which is equivalent in power with Turing machines (the Standardabweichung model of algorithmic computing) is said to be computationally complete or *Turing complete*. In the case of an accepting (resp. generating) model like Turing machines (resp. formal

grammars), it is also said that such a model can accept (resp. generate) all *recursively enumerable* languages.

**Membrane structure** In biology, the cells are separated from the environment by a membrane; the internal compartments of a cell (nucleus, mitochondria, Golgi apparatus, vesicles, etc.) are also delimited by membranes. A membrane is a *separator* of a compartment, and it also has *filtering* properties (only certain substances, in certain conditions, can pass through a membrane). In a cell, the membranes are hierarchically arranged, and they delimit “protected reactors,” compartments where specific chemicals evolve according to specific reactions. In membrane computing there is a notion of space which consists of such cell-like arrangement of membranes and that is called *membrane structure*. From a theoretical point of view, this corresponds to a topological space (a partition of the space where the actual size and geometrical form of membranes are not important, and only the “include” or “neighbor” relationships between them are considered). Moreover, a cell-like (hence hierarchical) membrane structure corresponds to a tree; hence, a natural representation is by means of a tree or any mathematical representation of a tree.

**Multiset** A *multiset* is a set with multiplicities associated with its elements. For instance,  $\{(a, 2), (b, 3)\}$  is the multiset consisting of two copies of element  $a$  and three copies of element  $b$ . Mathematically, a multiset is identified with a mapping  $\mu$  from a *support set*  $U$  (an alphabet) to  $\mathbb{N}$ , the set of natural numbers,  $\mu : U \rightarrow \mathbb{N}$ . A multiset can be compactly represented by a string  $w \in U^*$ , where the number of occurrences of a symbol in  $w$  is the multiplicity of that element in the multiset represented by  $w$  ( $a^2b^3$  or  $ab^2ab$  for the example above).

**Multiset rewriting** A multiset of objects  $M$  evolves by means of (multiset) rewriting rules, of form  $u \rightarrow v$ , where  $u$  and  $v$  are

multisets. Such a rule is applicable if  $M$  contains  $u$ . The result of its application is the multiset  $M - u + v$ , i.e.,  $u$  is taken off from  $M$ , followed by adding  $v$  to the result.

**Register machine** A *register machine* is a computing device working with a finite number of registers that can hold natural numbers. It has a program that is composed of three types of instructions: add one to the value of some specified counter, subtract one from the value of some counter, and the if instruction that is performing a zero test on some counter and based on its result transfers the control to another instruction of the program. The register machines are known to be universal with only two counters. Moreover, it is possible to construct universal machines having a small number of instructions.

**Regulated rewriting** The *regulated rewriting* is the area of formal language theory that studies (context-free) grammars enriched with control mechanisms that restrict possible rule applications at each step. There are two types of restrictions: context-based conditions and control-based conditions. The first type of conditions specifies the properties of the word that need to be satisfied in order for a rule to be applicable to it. Typical examples are permitting (resp. forbidding) conditions that list symbols that should (resp. should not) be present in the word in order for a rule to be applicable to it. The control-based conditions specify the properties of the *control language* (the language consisting of all sequences of applied rules corresponding to a successful computation) that need to be satisfied in order to apply a rule. Typical examples are graph control (all possible sequences of rules are described by a finite automaton) and matrix control (graph control where the automaton is a finite union of loops starting and ending in the initial state).

**Sympport/antiport** An important way of selectively passing chemicals across biological membranes is the coupled transport through protein channels. The process of moving two or more objects in the same direction is called *sympport*; the process of simultaneously moving two or more objects across a membrane, in

opposite directions, is called *antiport*. For uniformity, the particular case of moving only one object is called *uniport*.

**Universality** The universality is a property of a class of computing (programmable) devices that states that there exists a concrete device that can “simulate” the computation of any device in the corresponding class by taking as additional input the “code” of the device to be simulated. Such “simulation” generally requires the input to be encoded and the output to be decoded using some recursive functions. The typical example for the universality is the class of Turing machines, proved to be universal in Turing (1937). As a consequence, any computationally complete model is universal.

## Introduction

Membrane computing is a branch of natural computing initiated in Păun (2000) which abstracts computing models from the architecture and the functioning of living cells, as well as from the organization of cells in tissues, organs (the brain included), or other higher-order structures. The initial goal of membrane computing was to learn from the cell biology something possibly useful to computer science and the area fast developed in this direction. Several classes of computing models called *P systems* (or membrane systems) were defined in this context, inspired from biological facts or motivated from mathematical or computer science points of view. A series of applications were reported in the last years; we refer to Ciobanu et al. (2006a) and to Zhang et al. (2017) for a comprehensive overview.

The main ingredients of a *P system* are (i) the membrane structure, (ii) the multisets of objects placed in the compartments of the membrane structure, and (iii) the rules for processing the objects and the membranes. Thus, membrane computing can be defined as a framework for devising computing models, which process multisets in compartments defined by means of membranes that are arranged in a topological space,

usually in a cell-like or tissue-like manner. These models are (in general) distributed and parallel.

Since in many cases the information processing in P systems can be seen as distributed multiset rewriting, there are strong connections between P systems and other multiset rewriting-based models like Petri nets or vector addition systems. Generally, it gives a different point of view on the corresponding problem. There are two main particularities of using P system-based approach for the investigation of multiset rewriting: (a) the explicit notion of (topological) space and of the location of rules and objects in this space and (b) the integration of notions from regulated rewriting area allowing many powerful features. The first point is extremely important as in many problems there is a notion of space and co-location of symbols and rules that is made explicit by using a P system and which shall be encoded and deduced using other multiset-based models. The second point makes the P systems area to perform the most advanced study of the principles of functioning of multiset-based models.

When a P system is considered as a computing device, hence it is investigated in terms of (theoretical) computer science; the main issues investigated concern the *computing power* (in comparison with standard models from computability theory, especially Turing machines and their restrictions) and the *computing efficiency* (the possibility of using the parallelism for solving computationally hard problems in a feasible time). Computationally and mathematically oriented ways of using the rules and of defining the result of a computation are considered in this case (e.g., maximal or minimal parallelism, halting, counting objects). When a P system is constructed as a model of a biochemical process, then it is examined in terms of dynamical systems, with the evolution in time being the issue of interest, not a specific output.

From a theoretical point of view, P systems are both powerful (most classes are Turing complete, even when using ingredients of a reduced complexity – a small number of membranes, rules of simple forms, ways of controlling the use of rules directly inspired from biology) and efficient (many classes of P systems, especially those with an

enhanced parallelism, can solve computationally hard problems, typically NP-complete problems, but also harder problems, in a feasible time, typically polynomial). Then, as a modeling framework, membrane computing is rather adequate for handling discrete (biological) processes, having many attractive features: easy understandability, scalability and programmability, inherent compartmentalization and nonlinearity, etc. Ideas from cell biology as captured by membrane computing proved to be rather useful in handling various computer science topics – computer graphics, robot control, membrane evolutionary algorithms, used for solving optimization problems, etc.

The literature of membrane computing has grown very fast (already in 2003, Thompson Institute for Scientific Information (ISI) has qualified the initial paper as “fast breaking” and the domain as “emergent research front in computer science” – see <http://esi-topics.com>), while the bibliography of the field counts, at the middle of 2016, more than 3000 titles (see the Web site from <http://ppage.psystems.eu>). Moreover, the domain is now very diverse, as a consequence of the many motivations of introducing new variants of P systems: to be biologically oriented/realistic, mathematically elegant, computationally powerful, and efficient. That is why it is possible to give here only a few basic notions and only a few (types of) results and of applications. The reader interested in details should consult the monograph Păun (2002), the volume Ciobanu et al. (2006b) where a friendly introduction to membrane computing can be found in the first chapter, the Handbook Păun et al. (2010), the book Zhang et al. (2017), and the comprehensive bibliography from the above mentioned Web page.

## Types of P Systems

The field started by looking to the cell in order to learn something possibly useful to computer science, but then the research also considered cell organization in tissues (in general, populations of cells, such as colonies of bacteria) and, recently, also neuron organization in the brain. Thus, at the moment, there are four main types of P systems: (i) cell-like

P systems, (ii) tissue-like P systems, (iii) neural-like P systems, and (iv) numerical P systems.

The **cell-like P systems** imitate the (eukaryotic) cell. Their basic ingredient is the *membrane structure*, a hierarchical arrangement of membranes (understood as three-dimensional vesicles), delimiting compartments where multisets of symbol objects are placed; rules for evolving these multisets as well as the membranes are provided and also localized, acting in specified compartments or on specified membranes. The objects not only evolve, but they also pass through membranes (we say that they are “communicated” among compartments). The rules can have several forms, and their use can be controlled in various ways: promoters, inhibitors, priorities, etc.

In **tissue-like P systems**, several one-membrane cells are considered as evolving in a common environment. They contain multisets of objects, while also the environment contains objects. Certain cells can communicate directly (channels are provided between them), and all cells can communicate through the environment. The channels can be given in advance or they can be dynamically established – this latter case appears in so-called population P systems.

**Numerical P systems** are based on a cell-like membrane structure, but in its compartment evolve numerical variables rather than (biochemical) objects like in a cell-like P system. These variables evolve by means of *programs*, composed of a production function and a *repartition protocol*.

There are two types of **neural-like P systems**. One of them is similar to tissue-like P systems in the fact that the cells (neurons) are placed in the nodes of an arbitrary graph and they contain multisets of objects, but they also have a *state* which controls the evolution. Another variant called *spiking neural P systems*, where one uses only one type of objects, the *spike*, and the main information one works with is the time distance between consecutive spikes.

The cell-like P systems were introduced first and their theory is now very well developed; tissue-like P systems have also attracted a considerable interest, while the neural-like systems, mainly under the form of spiking neural P systems, are only recently investigated.

Another important distinction between variants P systems is made based on the possibility to evolve in time the membrane structure (add/delete/move membranes, change links, etc.). If the structure can be changed, then corresponding models are called P systems with *dynamically evolving structure* or P systems with *active membranes* (in the case of cell-like systems). If the structure cannot evolve, corresponding systems are called *static* P systems. It is worth to note that the deletion and the bounded (limited in number) creation of membranes yield a finite set of possible membrane structures, so corresponding models are variants of static P systems.

Any static P system using multiset rewriting rules operating on objects in membranes can be *flattened*, e.g., reduced to one membrane (Freund et al. 2013). This makes a strong link between P systems and multiset rewriting, as well as other multiset-based models like Petri nets, vector addition systems, and register machines. Basically this means that any result for any model cited above can be easily transcribed in terms of another model. However, obtained constructions could use some nonstandard or nonmainstream features (e.g., max-step semantics in Petri nets).

While the flattening is a useful tool for the comparison between different models of P systems (as well as linking them to related models), it makes the analysis of the system more difficult as it hides the localization of objects and rules (so, the possible collocations). Hence, in general, the description/definition of P systems makes a particular accent on the underlying structure of the system, making rules and objects to be spatially located in the system.

Many classes of P systems can be obtained by considering various possibilities for the various ingredients. We enumerate here several of these possibilities, without exhausting the list:

- Objects: symbols, strings of symbols, spikes, arrays, trees, numerical variables, other data structures, combinations
- Data structure: multisets, sets (languages in the case of strings), fuzzy sets, fuzzy multisets, (algebraic) groups

- Place of objects: in compartments, on membranes, combined
- Forms of rules: multiset rewriting, symport/antiport, communication rules, boundary rules, with active membranes, combined, string rewriting, array/trees processing, spike processing
- Controls on rules: catalysts, priority, promoters, inhibitors, activators, sequencing, energy
- Form of membrane structure: cell-like (tree), tissue-like (arbitrary graph)
- Type of membrane structure: static, dynamic, precomputed (arbitrarily large)
- Timing: synchronized, non-synchronized, local synchronization, time-free
- Ways of using the rules: maximal parallelism, minimal parallelism, bounded parallelism, sequential parallelism
- Successful computations: global halting, local halting, with specified events signaling the end of a computation, non-halting
- Modes of using a system: generative, accepting, computing an input-output function, deciding
- Types of evolution: deterministic, non-deterministic, confluent, probabilistic
- Ways to define the output: internal, external, traces, tree of membrane structure, spike train
- Types of results: set of numbers, set of vectors of numbers, languages, set of arrays, yes/no

We refer to the literature for details, and we only add here the fact that when using P systems as models of biological systems/processes, we have to apply the rules in ways suggested by biochemistry, according to reaction rates or probabilities; in many cases, these rates are computed dynamically, depending on the current population of objects in the system.

## General Functioning of P Systems

In short, a P system consists of an (hierarchical) arrangement of *membranes*, which delimit *compartments*, where *multisets* (sets with multiplicities associated with their elements) of abstract *objects* are placed. These objects correspond to

the chemicals from the compartments of a cell; the chemicals swim in water (many of them are bound on membranes, but we do not consider this case here), and their multiplicity matters – that is why the data structure most adequate to this situation is the multiset (a multiset can be seen as a string modulo permutation that is why in membrane computing, one usually represents the multisets by strings). In what follows, the objects are supposed to be unstructured; hence, we represent them by symbols from a given alphabet.

The objects evolve according to *rules* which are also associated with the regions. The rules say both how the objects are changed and how they can be moved (*communicated*) across membranes. In many cases the rules can be seen as particular cases of multiset rewriting rules enriched with communication. A particular interest in the area is to restrict possible types of rules, especially following some biological motivation. As example we cite catalytic rules (of form  $ca \rightarrow cu$ ) or symport/antiport rules (of form  $(u, in; v, out)$ ) motivated by corresponding biological phenomena.

There also are rules which only move objects across membranes, as well as rules for evolving the membranes themselves (e.g., by destroying, creating, dividing, or merging membranes). By using these rules, we can change the configuration of a system (the multisets from their compartments as well as the membrane structure); we say that we get a *transition* among system configurations.

The rules can be applied in many ways. The basic mode imitates the biological way chemical reactions are performed – in parallel – with the mathematical additional restriction to have a *maximal parallelism*: one applies a bunch of rules which are maximal; no further object can evolve at the same time by any rule. There might be several such groups (more precisely multisets) of rules, so, in general case, the evolution is performed in a *nondeterministic* manner.

Besides the maximally parallel mode, there were considered several others: sequential (one rule is used in each step), bounded parallelism (the number of membranes to evolve and/or the number of rules to be used in any step is bounded

in advance), set-maximal (or flat) parallelism (at most one rule of each kind can be used in a maximally parallel manner), minimal parallelism (in each compartment where a rule *can* be used, at least one rule *must* be used), etc.

A sequence of transitions forms a *computation*, and with computations which *halt* (reach a configuration where no rule is applicable), we associate a *result*, for instance, in the form of the multiset of objects present in the halting configuration in a specified membrane.

This way of using a P system, starting from an initial configuration and computing a number, is a grammar-like (generative) one. We can also work in an automata style: an input is introduced in the system, for instance, in the form of a number represented by the multiplicity of an object placed in a specified membrane, and we start computing; the input number is accepted if and only if the computation halts. A combination of the two modes leads to a functional behavior: an input is introduced in the system (at the beginning, or symbol by symbol during the computation), and also an output is produced. In particular, we can have a decidability case, where the input encodes a decision problem and the output is one of two special objects representing the answers *yes* and *no* to the problem.

The generalization of this approach is obvious. We start from the cell, but the abstract model deals with very general notions: membranes interpreted as separators of regions with filtering capabilities, objects, and rules assigned to regions; the basic data structure is the multiset. Thus, membrane computing can be interpreted as a *bio-inspired framework for distributed parallel processing of multisets*. If the reader is interested by technical details, we suggest to consult Verlan (2013) as starting point that gives a good insight on the overall structure and the semantics of P systems.

As briefly introduced above, the P systems are synchronous systems, and this feature is useful for theoretical investigations (e.g., for obtaining universality results or results related to the computational complexity of P systems). Also non-synchronized systems were considered asynchronous in the standard sense or even time-free, or clock-free (e.g., generating the same output,

irrespective of the duration associated with the evolution rules). Similarly, in applications to biology, specific strategies of evolution are considered. We do not enter here into details; rather we refer the reader to the bibliography given below.

## Examples of P Systems

In what follows, in order to let the reader get a flavor of membrane computing, we will discuss in some detail only basic cell-like P systems and spiking neural P systems, and we refer to the area literature for other classes.

### Basic Cell-Like P Systems

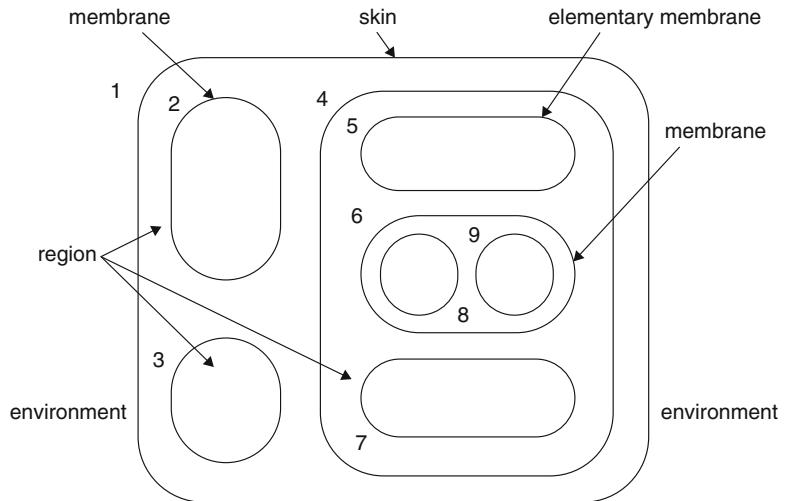
Because in this section we only consider cell-like P systems, they will be simply called P systems.

As said above, we look to the cell structure and functioning, trying to get suggestions for an abstract computing model. The fundamental feature of a cell is its compartmentalization through membranes. Accordingly, the main particularity of a cell-like P system is the *membrane structure* in the form of a hierarchical arrangement of membranes (thus corresponding to a tree). Figure 1 illustrates this notion and the related terminology.

We distinguish the external membrane (corresponding to the plasma membrane and usually called the *skin* membrane) and several internal membranes; a membrane without any other membrane inside is said to be *elementary*. Each membrane determines a compartment, also called *region*, the space delimited from above by it and from below by the membranes placed directly inside, if any exists. The correspondence membrane-region is one to one, so that we identify by the same label a membrane and its associated region.

In the basic class of P systems, each region contains a multiset of symbol objects, described by symbols from a given alphabet.

The objects evolve by means of evolution rules, which are also localized, associated with the regions of the membrane structure. The typical form of such a rule is  $cd \rightarrow (a, \text{here}) (b, \text{out}) (b, \text{in})$ , with the following meaning: one copy of object *c* and one copy of object *d* react, and the reaction produces one copy of *a* and two copies of

**Membrane Computing:  
Power and Complexity,**
**Fig. 1** A membrane structure


*b*; the newly produced copy of *a* remains in the same region (indication *here*); one of the copies of *b* exits the compartment, going to the surrounding region (indication *out*); and the other enters one of the directly inner membranes (indication *in*). We say that the objects *a*, *b*, and *b* are *communicated* as indicated by the commands associated with them in the right-hand member of the rule. When an object exits the skin membrane, it is “lost” in the environment, and it possibly never comes back into the system. If no inner membrane exists (i.e., the rule is associated with an elementary membrane), then the indication *in* cannot be followed, and the rule cannot be applied.

As discussed in the previous section, membrane structure and the multisets of objects from its compartments identify a *configuration* of a P system. By a nondeterministic maximally parallel use of rules as suggested above, we pass to another configuration; such a step is called a *transition*. A sequence of transitions constitutes a *computation*. A computation is successful if it halts, and it reaches a configuration where no rule can be applied to the existing objects. With a halting computation, we can associate a *result* in various ways. The simplest possibility is to count the objects present in the halting configuration in a specified elementary membrane; this is called *internal output*. We can also count the objects which leave the system during the computation, and this is called *external output*. In both cases,

the result is a number. If we distinguish among different objects, then we can have as the result a vector of natural numbers.

The objects which leave the system can also be arranged in a sequence according to the moments when they exit the skin membrane, and in this case the result is a string.

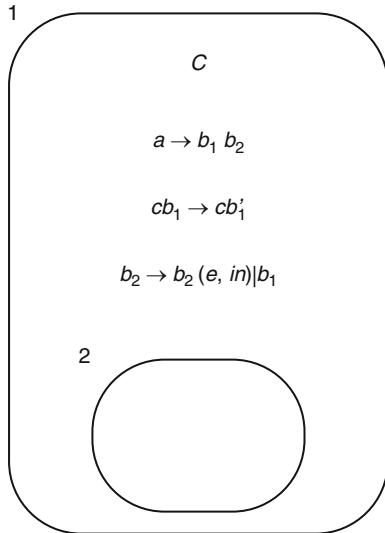
Because of the non-determinism of the application of rules, starting from an initial configuration, we can get several successful computations, hence several results. Thus, a P system *computes* (one also uses to say *generates*) a set of numbers, or a set of vectors of numbers, or a language.

In general, a (cell-like) P system is formalized as a construct

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, i_0),$$

where *O* is the alphabet of objects (sometimes it can be split into several alphabets according to specific restrictions),  $\mu$  is the membrane structure (with *m* membranes),  $w_1, \dots, w_m$  are multisets of objects present in the *m* regions of  $\mu$  at the beginning of a computation,  $R_1, \dots, R_m$  are finite sets of evolution rules associated with the regions of  $\mu$ , and  $i_0$  is the label of a membrane, used as the output membrane.

We end this section with a simple example, illustrating the architecture and the functioning of a (cell-like) P system. Figure 2 indicates the initial configuration (the rules included) of a system which computes a function, namely,  $n \rightarrow n^2$ ,



**Membrane Computing: Power and Complexity,**  
**Fig. 2** A P system with catalysts and promoters

for any natural number  $n \geq 1$ . Besides catalytic (rules of type  $ca \rightarrow cv$ ) and noncooperative (context-free) rules, the system also contains a rule with promoters (permitting context conditions),  $b_2 \rightarrow b_2(e, in)|_{b_1}$ : the object  $b_2$  evolves to  $b_2e$  only if at least one copy of object  $b_1$  is present in the same region.

In symbols, the system is given as follows:

$$\begin{aligned} \Pi &= (O, C, \mu, \omega_1, \omega_2, R_1, R_2, i_0), \text{ where} \\ O &= \{a, b_1, b'_1, b_2, c, e\} \text{ (the set of objects)} \\ C &= \{c\} \text{ (the set of catalysts)} \\ \mu &= [1[2]_2]_1 \text{ (membrane structure)} \\ \omega_1 &= c \text{ (initial objects in region 1)} \\ \omega_2 &= c \text{ (initial objects in region 2)} \\ R_1 &= \{a \rightarrow b_1 b_2, cb_1 \rightarrow cb'_1, b_2 \rightarrow b_2(e, in)|_{b_1}\} \\ i_0 &= 2 \text{ (the output region)} \\ R_2 &= \emptyset \text{ (rules in region 2)} \end{aligned}$$

We start with only one object in the system, the catalyst  $c$ . If we want to compute the square of a number  $n$ , then we have to input  $n$  copies of the object  $a$  in the skin region of the system. In that moment, the system starts working, by using the rule  $a \rightarrow b_1 b_2$ , which has to be applied in parallel to all copies of  $a$ ; hence, in one step, all objects  $a$  are replaced by  $n$  copies of  $b_1$  and  $n$  copies of  $b_2$ . From now on, the other two

rules from region 1 can be used. The catalytic rule  $cb_1 \rightarrow cb'_1$  can be used only once in each step, because the catalyst is present in only one copy. This means that in each step, one copy of  $b_1$  gets primed. Simultaneously (because of the maximal parallelism), the rule  $b_2 \rightarrow b_2(e, in)|_{b_1}$  should be applied as many times as possible, and this means  $n$  times, because we have  $n$  copies of  $b_2$ . Note the important difference between the promoter  $b_1$ , which allows using the rule  $b_2 \rightarrow b_2(e, in)|_{b_1}$ , and the catalyst  $c$ : the catalyst is involved in the rule and it is counted when applying the rule, while the promoter makes possible the use of the rule, but it is not counted; the same (copy of an) object can promote any number of rules. Moreover, the promoter can evolve at the same time by means of another rule (the catalyst is never changed).

In this way, in each step we change one  $b_1$  to  $b_1'$  and we produce  $n$  copies of  $e$  (one for each copy of  $b_2$ ); the copies of  $e$  are sent to membrane 2 (the indication  $in$  from the rule  $b_2 \rightarrow b_2(e, in)|_{b_1}$ ). The computation should continue as long as there are applicable rules. This means exactly  $n$  steps: in  $n$  steps, the rule  $cb_1 \rightarrow cb'_1$  will exhaust the objects  $b_1$ , and in this way neither this rule can be applied nor  $b_2 \rightarrow b_2(e, in)|_{b_1}$  because its promoter does no longer exist. Consequently, in membrane 2, considered as the output membrane, we get  $n$  copies of object  $e$ .

Note that the computation is deterministic, always the next configuration of the system is unique, and that changing the rule  $b_2 \rightarrow b_2(e, in)|_{b_1}$  with  $b_2 \rightarrow b_2(e, in)|_{b_1'}$ , the  $n^2$  copies of  $e$  will be sent to the environment; hence, we can read the result of the computation outside the system, and in this case membrane 2 is useless.

### Spiking Neural P Systems

Spiking neural P systems (SN P systems) were introduced in Ionescu et al. (2006) with the aim of defining P systems based on ideas specific to spiking neurons, much investigated in neural computing.

Very shortly, an SN P system consists of a set of *neurons* (cells consisting of only one

membrane) placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol  $a$ ) along *synapses* (arcs of the graph). Thus, the architecture is that of a tissue-like P system, with only one kind of object present in the cells. The objects evolve by means of *spiking rules*, which are of the form  $E|a^c \rightarrow a; d$ , where  $E$  is a regular expression over  $\{a\}$  and  $c, d$  are natural numbers,  $c \geq 1, d \geq 0$ . The meaning is that a neuron containing  $k$  spikes such that  $a^k \in L(E)$ ,  $k > c$  can consume  $c$  spikes and produce one spike, after a delay of  $d$  steps.

This spike is sent to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*, of the form  $a^s \rightarrow \lambda$ , with the meaning that  $s \geq 1$  spikes are forgotten, provided that the neuron contains exactly  $s$  spikes. We say that the rules “cover” the neuron, and all spikes are taken into consideration when using a rule. The system works in a synchronized manner, i.e., in each time unit, each neuron which can use a rule should do it, but the work of the system is sequential in each neuron: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output neuron*, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1; the other moments are marked with 0. This binary sequence is called the *spike train* of the system – it might be infinite if the computation does not stop. In the spirit of spiking neurons, the result of a computation is encoded in the distance between consecutive spikes sent into the environment by the (output neuron of the) system. For example, we can consider only the distance between the first two spikes of a spike train, the distance between the first  $k$  spikes, or the distances between all consecutive spikes, taking into account all intervals or only intervals that alternate, all computations or only halting computations, etc.

An SN P system can also be used in the accepting mode: a neuron is designated as the *input neuron* and two spikes are introduced in it, at an interval of  $n$  steps; the number  $n$  is accepted if the computation halts.

Another possibility is to consider the spike train itself as the result of a computation, and

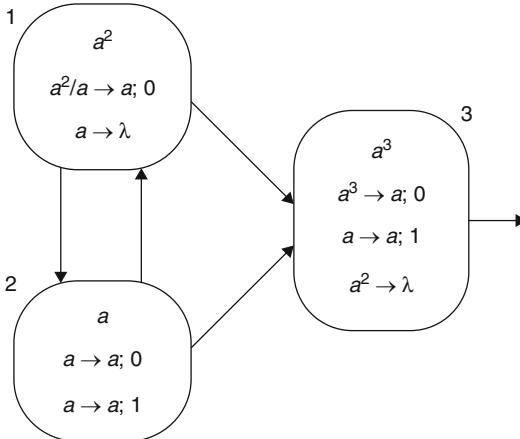
then we obtain a (binary) *language-generating device*. We can also consider input neurons and then an SN P system can work as a transducer. Languages on arbitrary alphabets can be obtained by generalizing the form of rules: take rules of the form  $E|a^c \rightarrow a^p; d$ , with the meaning that, provided that the neuron is covered by  $E$ ,  $c$  spikes are consumed and  $p$  spikes are produced and sent to all connected neurons after  $d$  steps (such rules are called extended). Then, with a step when the system sends out  $i$  spikes, we associate a symbol  $b_i$ , and thus we get a language over an alphabet with as many symbols as the number of spikes simultaneously produced. Another natural extension is to consider several output neurons, thus producing vectors of numbers, not only single numbers.

Also for SN P systems, we skip the technical details, but we consider a simple example. We give it first in a formal manner (if a rule  $E|a^c \rightarrow a; d$  has  $L(E) = \{a^c\}$ , and then we write it in the simplified form  $a^c \rightarrow a; d$ ):

$$\begin{aligned} \Pi_1 &= (O, \sigma_1, \sigma_2, \sigma_3, \text{syn}, \text{out}), \text{with} \\ O &= \{a\} \text{ (alphabet, with only one object, the spike)} \\ \sigma_1 &= (2, \{a^2|a \rightarrow a; 0, a \rightarrow \lambda\}) \text{ (First neuron: initial spikes, rules)} \\ \sigma_2 &= (1, \{a \rightarrow a; 0, a \rightarrow a; 1\}) \text{ (Second neuron: initial spikes, rules)} \\ \sigma_3 &= (3, \{a^3 \rightarrow a; 0, a \rightarrow a; 1, a^2 \rightarrow \lambda\}) \text{ (Third neuron: initial spikes, rules)} \\ \text{syn} &= \{(1, 2), (2, 1), (1, 3), (2, 3)\} \text{ synapses} \\ \text{out} &= 3 \text{ (output neuron)} \end{aligned}$$

This system is represented in a graphical form in Fig. 3 and it functions as follows. All neurons can fire in the first step, with neuron  $\sigma_2$  choosing nondeterministically between its two rules. Note that neuron  $\sigma_1$  can fire only if it contains two spikes; one spike is consumed and the other remains available for the next step.

Both neurons  $\sigma_1$  and  $\sigma_2$  send a spike to the output neuron,  $\sigma_3$ ; these two spikes are forgotten in the next step. Neurons  $\sigma_1$  and  $\sigma_2$  also exchange their spikes; thus, as long as neuron  $\sigma_2$  uses the rule  $a \rightarrow a; 0$ , the first neuron receives one spike, thus completing the needed two spikes for firing again.



**Membrane Computing: Power and Complexity,**  
**Fig. 3** An SN P system generating all natural numbers greater than 1

However, at any moment, starting with the first step of the computation, neuron  $\sigma_2$  can choose to use the rule  $a \rightarrow a; 1$ . On the one hand, this means that the spike of neuron  $\sigma_1$  cannot enter neuron  $\sigma_2$ , and it only goes to neuron  $\sigma_3$ ; in this way, neuron  $\sigma_2$  will never work again because it remains empty. On the other hand, in the next step, neuron  $\sigma_1$  has to use its forgetting rule  $a \rightarrow \lambda$ , while neuron  $\sigma_3$  fires, using the rule  $a \rightarrow a; 1$ . Simultaneously, neuron  $\sigma_2$  emits its spike, but it cannot enter neuron  $\sigma_3$  (it is closed this moment); the spike enters neuron  $\sigma_1$ , but it is forgotten in the next step. In this way, no spike remains in the system. The computation ends with the expelling of the spike from neuron  $\sigma_3$ . Because of the waiting moment imposed by the rule  $a \rightarrow a; 1$  from neuron  $\sigma_3$ , the two spikes of this neuron cannot be consecutive, but at least two steps must exist in between.

Thus, we conclude that  $\Pi_1$  computes/generates all natural numbers greater than or equal to 2.

### Generalized Communicating P Systems

Generalized communicating P systems were introduced in Verlan et al. (2008) as the generalization of minimal symport/antiport operations. The main idea is to model a conservative system where the objects cannot be transformed (rewritten), but only are moved through the membranes of the system. Moreover, this

movement is performed in a particular manner which can be seen as a pairwise synchronization of objects. More precisely, the system structure is a so-called network of cells, which is a list of cells having an identifier and a multiset content. The generalized communication rule is written as  $(a, i)(b, j) \rightarrow (a, k)(b, l)$ , which indicates that if an object  $a$  is present in cell (membrane) with the id (number)  $i$  and the object  $b$  is present in cell  $j$ , then they can synchronize, and as the result of the application of the rule, object  $a$  moves to cell  $k$  and object  $b$  moves to cell  $l$ . In case there are several copies of  $a$  and  $b$  in corresponding cells, only one of them is moved using one rule (this is called 1: 1 mode). In the 1: *all* mode, one copy of  $a$  and all copies of  $b$  are moved to the corresponding cells. In the general case, indexes  $i, j, k, l$  can be pairwise different; however if some of them coincide, then restricted variants of the model are obtained (in particular symport/antiport). We remark that rules of the system induce a structure corresponding to a hypergraph.

The evolution mode is maximally parallel; this means that the whole system can be seen as a (maximally) parallel evolution of signals in a network, where only pairwise synchronization of signals is permitted. The result of the computation is the number of objects in some designated output cell.

Since no objects are rewritten or produced, the system is finite. In order to increase the computational power, a special cell, the *environment*, labeled by 0 is introduced. This cell contains an infinite supply of some particular objects.

We will consider an example of the computation of the  $n^2$  function. We start by introducing a graphical notation for the generalized communicating rules as shown in Fig. 4.

Figure 5 gives the description of the system. We will skip the textual representation and the discussion about using a single mode and refer to Verlan et al. (2008) for the missing details. We only remark that we depicted cell 0 three times in order to simplify the picture – it should be read as the same cell in all three cases.

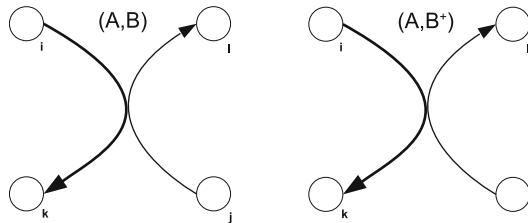
The initial value is given as the number of symbols  $A$  in cell 1. The system is performing

the following algorithm (with the initial values 1 for  $B$  and 0 for  $C$ ):

```

1 while (A>0) {
2   A=A-1
3   C=C+B
4   B=B+2
5 }
```

Clearly, at the end of the above algorithm, we have that  $C = A^2$ . Technically, the three steps of the inner loop are made using the additional symbol  $S$  (used as instruction pointer to sequence the operations). The first rule corresponds to line 2 of the algorithm. Indeed, it decreases the number of symbols  $A$  in cell 1, and the instruction pointer  $S$  moves to the next instruction.



**Membrane Computing: Power and Complexity,**  
**Fig. 4** Graphical notation for the rule  $(a, i) (b, j) \rightarrow (a, k) (b, l)$  in 1: 1 mode (left) and 1: all mode (right)

#### Membrane Computing: Power and Complexity,

**Fig. 5** Generalized  
communicating P system  
computing  $n^2$ . The rules are  
numbered for convenience

Rule 2 operates in 1: *all* mode and moves all symbols  $B$  from cell 5 to cell 6. Further, rule 3 is applied in a maximally parallel manner, which means that for every  $B$  moved to cell 7, a symbol  $C$  is moved from the environment (cell 0) to cell 8. We recall that the environment contains an unbounded number of objects  $B$  and  $C$ . The overall action of rules 2 and 3 corresponds to the execution of the line 3 of the algorithm.

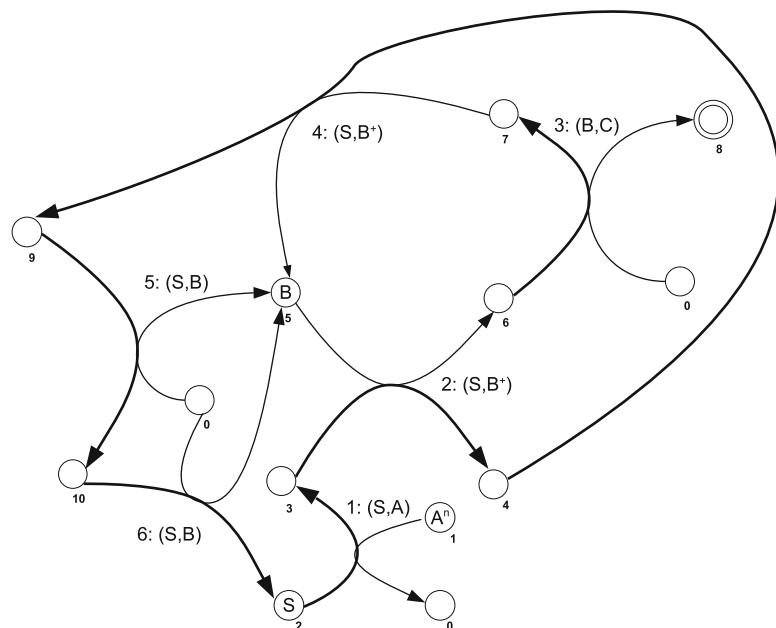
Next, rule 4 is executed, operating in 1: *all* mode. This allows to move all objects  $B$  from cell 7 to cell 5. The next two rules (5 and 6) allow to increment by 2 the number of objects  $B$  in cell 5, corresponding to line 4 of the algorithm.

Now object  $S$  returned to cell 2 (and objects  $B$  to cell 5), so a new iteration can start.

The system stops when there are no more objects  $A$ . It is easy to see that in this case, no rule is applicable, so the halting condition occurs.

#### Computing Power

As we have mentioned before, many classes of P systems, combining various ingredients (as described above or similar), are able of simulating register machines; hence they are



*computationally complete.* Always, the proofs of results of this type are constructive, and this has an important consequence from the computability point of view: there are universal (hence *programmable*) P systems. In short, starting from a universal Turing machine (or an equivalent universal device), we get an equivalent universal P system. Among others, this implies that in the case of Turing complete classes of P systems, the hierarchy on the number of membranes always collapses (at most at the level of the universal P systems). Actually, the number of membranes sufficient in order to characterize the power of Turing machines by means of P systems is always rather small (one to two in most of the cases). We only mention here four of the most interesting (types of) universality results for cell-like P systems:

1. P systems with symbol objects with catalytic rules, using only two catalysts and one membrane, are computationally complete.
2. P systems with minimal symport/antiport rules (where at most two objects are involved in a rule) using two membranes are computationally complete.
3. P systems with symport/antiport rules (of arbitrary size), using only two objects and seven membranes, are computationally complete.
4. P systems with symport/antiport rules with inhibitors (forbidding conditions), one membrane and having only 16 rules, are computationally complete.

There are several other similar results, improvements, or extensions of them. Many results are also known for tissue-like P systems. Details can be found, e.g., in the proceedings of the yearly Conference on Membrane Computing and Asian Conference on Membrane Computing mentioned in the bibliography of this entry. Most universality results were obtained in the deterministic case, but there also are situations where the deterministic systems are strictly less powerful than the nondeterministic ones. This is proven in Ibarra and Yen (2006), for the accepting catalytic P systems.

The hierarchy on the number of membranes collapses in many cases also for nonuniversal classes of P systems, but there also are cases when “the number of membrane matters,” to cite the title of Ibarra (2003), where two classes of P systems were defined for which the hierarchies on the number of membranes are infinite.

Also various classes of SN P systems are computationally complete as devices which generate or accept sets of numbers. This is true when no bound is imposed on the number of spikes present in any neuron; if such a bound exists, then the sets of numbers generated (or accepted) are semilinear.

## Computational Efficiency

The computational power (the “competence”) is only one of the important questions to be dealt with when defining a new (bio-inspired) computing model. The other fundamental question concerns the computing *efficiency*. Because P systems are parallel computing devices, it is expected that they can solve hard problems in an efficient manner – and this expectation is confirmed for systems provided with ways for producing an exponential workspace in a linear time. However, we would like to remark that there are no physical implementations of P systems able to produce an exponential workspace in a linear time (and going beyond toy examples), so the corresponding research has rather theoretical importance.

Three main such biologically inspired possibilities have been considered so far in the literature, and *all of them were proven to lead to polynomial solutions to NP-complete problems*. These three ideas are *membrane division*, *membrane creation*, and *string replication*. The standard problems addressed in this framework were decidability problems, starting with SAT, the Hamiltonian path problem, and the node-covering problem, but also other types of problems were considered, such as the problem of inverting one-way functions or the subset-sum and the knapsack problems (note that the last two are numerical problems, where the answer

is not of the yes/no type, as in decidability problems).

Roughly speaking, the framework for dealing with complexity matters is that of *accepting P systems with input*: a family of P systems of a given type is constructed starting from a given problem, and an instance of the problem is introduced as an input in such systems; working in a deterministic mode (or a *confluent* mode: some non-determinism is allowed, provided that the branching converges after a while to a unique configuration, or, in the weak confluent case, all computations halt and all of them provide the same result), in a given time, one of the answers yes/no is obtained, in the form of specific objects sent to the environment. The family of systems should be constructed in a uniform mode by a Turing machine, working a polynomial time.

This direction of research is very active at the present moment. More and more problems are considered, the membrane computing complexity classes are refined, characterizations of the  $P \neq NP$  conjecture were obtained in this framework, and several characterizations of the class P, even problems which are PSPACE complete, were proven to be solvable in polynomial time by means of membrane systems provided with membrane division or membrane creation. An important (and difficult) problem is that of finding the borderline between efficiency and non-efficiency: which ingredients should be used in order to be able to solve hard problems in a polynomial time? Many results in this respect were reported by M.J. Pérez-Jimenez and his co-workers (see “Bibliography”), but still many problems remain open in this respect.

## Future Directions

Although so much developed in less than 18 years since the investigations were initiated, membrane computing still has a large number of open problems and research topics which wait for research efforts.

A general class of theoretical questions concerns the borderline between universality and

nonuniversality or between efficiency and non-efficiency, i.e., concerning the succinctness of P systems able to compute at the level of Turing (or register) machines or to solve hard problems in polynomial time, respectively. Then, because universality implies undecidability of all non-trivial questions, an important issue is that of finding classes of P systems with decidable properties.

This is also related to the use of membrane computing as a modeling framework: if no insights can be obtained in an analytical manner, algorithmically, then what remains is to simulate the system on a computer. To this aim, better programs are still needed, maybe parallel implementations, able to handle real-life questions (for instance, in the quorum sensing area, existing applications deal with hundreds of bacteria, but biologists would need simulations at the level of thousands of bacteria in order to get convincing results).

We give below some interesting topics that arise last years; however we recommend the reading of the Bulletin of the International Membrane Computing Society as well as the proceedings of the recent conferences on Membrane Computing to stay up to date with the newest research trends in the area.

At the time of writing this material, the following topics attract a lot of research effort (we concentrate on the theoretical topics; other questions related to applications are discussed in another chapter of this book):

- Introduction and the investigation of different variants of spiking neural P systems. The topics vary from the introduction of new (biologically motivated) ingredients to the computational completeness, universality, and efficiency.
- Investigation of P colonies, which is a model of P systems using extremely simple rules and resources.
- Investigation of P systems where the multiset structure and operations are replaced by generalized variants. Simplest versions are fuzzy and rough multisets; recent development makes use of generalized multisets

- which are a function from an alphabet to an abelian group.
- Investigation of different new properties of P systems, like new derivation modes and halting conditions and new types of objects and rules with special semantics.
  - P system simulator design and questions related to efficient implementations.
  - Verification frameworks using P systems, especially based on kernel P systems.

**Acknowledgments** The work of G. Zhang was supported by the National Natural Science Foundation of China (61373047 and 61672437) and the Research Project of Key Laboratory of Fluid and Power Machinery (Xihua University), Ministry of Education, P.R. China (JYBFXYQ-1).

## Bibliography

### Primary Literature

- Ciobanu G, Păun G, Pérez-Jiménez MJ (2006a) Applications of membrane computing, vol 17. Springer, Berlin
- Ciobanu G, Pérez-Jiménez MJ, Păun Gh (2006b) Applications of membrane computing. Natural computing series. Springer, Berlin
- Freund R, Leporati A, Mauri G, Porreca AE, Verlan S, Zandron C (2013) Flattening in (tissue) P systems. In: Alhazov A, Cojocaru S, Gheorghe M, Rogozhin Y, Rozenberg G, Salomaa A (eds) Membrane computing – 14th international conference, CMC 2013, Chișinău, 20–23 Aug 2013, Revised Selected Papers, Springer, Lecture notes in computer science, vol 8340, pp 173–188
- Ibarra OH (2003) The number of membranes matters. In: Martín-Vide C, Mauri G, Paun G, Rozenberg G, Salomaa A (eds) Membrane computing, international workshop, WMC 2003, Tarragona, 17–22 July 2003, Revised Papers, Springer, Lecture notes in computer science, vol 2933, pp 218–231. [https://doi.org/10.1007/978-3-540-24619-0\\_16](https://doi.org/10.1007/978-3-540-24619-0_16)
- Ibarra OH, Yen H (2006) Deterministic catalytic systems are not universal. *Theor Comput Sci* 363(2):149–161. <https://doi.org/10.1016/j.tcs.2006.07.029>
- Ionescu M, Păun G, Yokomori T (2006) Spiking neural P systems. *Fundam Informaticae* 71(2–3):279–308
- Păun G (2000) Computing with membranes. *J Comput Syst Sci* 61(1):108–143. <https://doi.org/10.1006/jcss.1999.1693>, also Turku Center for Computer Science Report TUCS 208, Nov 1998
- Păun Gh (2002) Membrane computing: an introduction. Natural Computing Series. Springer-Verlag Berlin Heidelberg
- Păun G, Rozenberg G, Salomaa A (eds) (2010) The Oxford handbook of membrane computing. Oxford University Press, Oxford
- Turing AM (1937) On computable numbers, with an application to the Entscheidungsproblem. *Proc London Math Soc s2-42(1):230–265*. <https://doi.org/10.1112/plms/s2-42.1.230>
- Verlan S (2013) Using the formal framework for P systems. In: Alhazov A, Cojocaru S, Gheorghe M, Rogozhin Y, Rozenberg G, Salomaa A (eds) Membrane computing – 14th international conference, CMC 2013, Chișinău, 20–23 Aug 2013, Revised Selected Papers, Springer, Lecture notes in computer science, vol 8340, pp 56–79
- Verlan S, Bernardini F, Gheorghe M, Margenstern M (2008) Generalized communicating P systems. *Theor Comput Sci* 404(1–2):170–184
- Zhang G, Păl'rez-Jiménez MJ, Gheorghe M (2017) Real-life applications with membrane computing. Emergence, complexity and computation, vol 25. Springer International Publishing, Cham

### Books and Reviews

- Alhazov A, Cojocaru S, Gheorghe M, Rogozhin Y, Rozenberg G, Salomaa A (eds) (2014) Membrane computing – 14th international conference, CMC 2013, Chișinău, 20–23 Aug 2013, Revised Selected Papers, Lecture notes in computer science, vol 8340, Springer. <https://doi.org/10.1007/978-3-642-54239-8>
- Corne DW, Frisco P, Păun Gh, Rozenberg G, Salomaa A (eds) (2009) Membrane computing – 9th international workshop, WMC 2008, Edinburgh, 28–31 July 2008, Revised Selected and Invited Papers, Lecture notes in computer science, vol 5391, Springer. <https://doi.org/10.1007/978-3-540-95885-7>
- Csuha-Jarai E, Gheorghe M, Rozenberg G, Salomaa A, Vaszil G (eds) (2013) Membrane computing – 13th international conference, CMC 2012, Budapest, 28–31 Aug 2012, Revised Selected Papers, Lecture notes in computer science, vol 7762, Springer. <https://doi.org/10.1007/978-3-642-36751-9>
- Eleftherakis G, Kefalas P, Păun Gh, Rozenberg G, Salomaa A (eds) (2007) Membrane computing, 8th international workshop, WMC 2007, Thessaloniki, 25–28 June 2007 Revised Selected and Invited Papers, Lecture notes in computer science, vol 4860, Springer
- Freund R, Păun Gh, Rozenberg G, Salomaa A (eds) (2006) Membrane computing, 6th international workshop, WMC 2005, Vienna, 18–21 July 2005, Revised Selected and Invited Papers, Lecture notes in computer science, vol 3850, Springer
- Frisco P (2009) Computing with cells: advances in membrane computing. OUP, Oxford
- Gheorghe M, Hinze T, Păun Gh, Rozenberg G, Salomaa A (eds) (2011) Membrane computing – 11th international conference, CMC 2010, Jena, 24–27 Aug 2010. Revised Selected Papers, Lecture notes in computer science, vol 6501, Springer. <https://doi.org/10.1007/978-3-642-18123-8>
- Gheorghe M, Păun Gh, Rozenberg G, Salomaa A, Verlan S (eds) (2012) Membrane computing – 12th international conference, CMC 2011, Fontainebleau, 23–26 Aug 2011, Revised Selected Papers, Lecture notes in

- computer science, vol 7184, Springer. <https://doi.org/10.1007/978-3-642-28024-5>
- Gheorghe M, Rozenberg G, Salomaa A, Sosík P, Zandron C (eds) (2014) Membrane computing – 15th international conference, CMC 2014, Prague, 20–22 Aug 2014, Revised Selected Papers, Lecture notes in computer science, vol 8961, Springer. <https://doi.org/10.1007/978-3-319-14370-5>
- Hoogeboom HJ, Păun Gh, Rozenberg G, Salomaa A (eds) (2006) Membrane computing, 7th international workshop, WMC 2006, Leiden, 17–21 July 2006, Revised, Selected, and Invited Papers, Lecture notes in computer science, vol 4361, Springer. <https://doi.org/10.1007/11963516>
- Leporati A, Rozenberg G, Salomaa A, Zandron C (eds) (2017) Membrane computing – 17th international conference, CMC 2016, Milan, 25–29 July 2016, Revised Selected Papers, Lecture notes in computer science, vol 10105, Springer. <https://doi.org/10.1007/978-3-319-54072-6>
- Martín-Vide C, Mauri G, Păun Gh, Rozenberg G, Salomaa A (eds) (2004) Membrane computing, international workshop, WMC 2003, Tarragona, 17–22 July 2003, Revised Papers, Lecture notes in computer science, vol 2933, Springer
- Mauri G, Păun Gh, Pérez-Jiménez MJ, Rozenberg G, Salomaa A (eds) (2005) Membrane computing, 5th international workshop, WMC 2004, Milan, 14–16 June 2004, Revised Selected and Invited Papers, Lecture notes in computer science, vol 3365, Springer
- Păun Gh, Pérez-Jiménez MJ, Riscos-Núñez A, Rozenberg G, Salomaa A (eds) (2010) Membrane computing, 10th international workshop, WMC 2009, Curtea de Arges, 24–27 Aug 2009. Revised Selected and Invited Papers, Lecture notes in computer science, vol 5957, Springer. <https://doi.org/10.1007/978-3-642-11467-0>
- Rozenberg G, Salomaa A, Sempere JM, Zandron C (eds) (2015) Membrane computing – 16th international conference, CMC 2015, Valencia, 17–21 Aug 2015, Revised Selected Papers, Lecture notes in computer science, vol 9504, Springer. <https://doi.org/10.1007/978-3-319-28475-0>



---

## Applications of P Systems

Marian Gheorghe<sup>1</sup>, Andrei Păun<sup>2</sup>, Sergey Verlan<sup>3</sup> and Gexiang Zhang<sup>4,5,6</sup>

<sup>1</sup>School of Electrical Engineering and Computer Science, University of Bradford, Bradford, West Yorkshire, UK

<sup>2</sup>Department of Computer Science, University of Bucharest, Bucharest, Romania

<sup>3</sup>LACL, Université Paris Est Créteil, Créteil, France

<sup>4</sup>Robotics Research Center, Xihua University, Chengdu, Sichuan, China

<sup>5</sup>Key Laboratory of Fluid and Power Machinery, Xihua University, Ministry of Education, Chengdu, Sichuan, China

<sup>6</sup>School of Electrical Engineering, Southwest Jiaotong University, Chengdu, Sichuan, China

### Article Outline

Glossary

Introduction

Applications in Systems in Synthetic Biology

Applications to Real-Life Complex Problems

Other Applications

Future Directions

Bibliography

### Glossary

**Computational model** A computational model is a concept introduced in computer science with the aim of defining an algorithm that is executed on an abstract machine. It is built for different purposes and makes use of various notations and formalisms. Some of the most widely used computational models are finite state machines, Turing machines, formal grammars, Boolean networks, Petri nets, cellular automata, and process calculi.

### Execution strategy of a P system

Every P system is executed in steps. In each step and each compartment, a number of rules are selected to be applied to the multiset contained in the compartment. The most utilized execution strategies are maximal parallelism (in each compartment after the rules are selected, no more objects are available to be processed by the existing rules), sequential execution (only one rule per compartment is applied), and stochastic behavior (the rules are selected in accordance with the probabilities associated to them). In most of the cases, maximal parallelism leads to nondeterministic behavior as there might be different ways of selecting the rules applied to the multisets contained in compartments. One of the most used stochastic behavior is based on Gillespie algorithm.

### Fuzzy reasoning spiking neural P system (FRSNP system)

This is a special type of P system combining the features of a spiking neural P system (SNP system) and some fuzzy theory concepts. The SNP system consists of an alphabet with only one element, called *spike*, neurons – each of them with its multiset and rules; a set of links between neurons, called *synapses*; and designated input and output neurons. Fuzzy values are associated with rules defining the potential values of spikes and fuzzy truth values for rules. These values control both the amount of spikes consumed and the rules selected to be executed.

**Membrane algorithms** A distributed evolutionary approach using the regions defined by the membranes to host instances of specific meta-heuristic search algorithms. It makes use of different topologies of the membrane structure and the types of rules involved in the meta-heuristic search algorithms. A broad spectrum of meta-heuristic search algorithms have been used in defining new membrane algorithms.

**Model checking** Model checking is a method for automatically verifying the correctness of properties of a system. Given a model of a

system and a specification, both formulated in a precise mathematical language, model checking automatically checks in an exhaustive manner that the model meets the specification. The model is a state machine, and the specification is given by a temporal logic formula.

**P system model** A P system consists of a set of compartments. The most common P system models have their compartments linked in such a way that they form a network (tissue P systems) or a tree – a hierarchical structure (cell P systems). Each compartment consists of a multiset of elements which are transformed through various rules.

**Rules** The most basic rules used by a P system are rewriting (evolution or transformation) and communication between neighbor compartments (those directly connected). For stochastic P systems, each rule has a probability or a kinetic constant associated with. The kinetic constants are utilized in computing the rule probabilities.

**Standard membrane controller (MeC)** A construct based on the concept of numerical P system. It uses a membrane structure; a set of variables (replacing the usual multisets), with some initial values; and a set of programs (instead of usual rewriting and communication rules) associated to compartments. The programs in each compartment define polynomial production functions, stating the values consumed from the local variables, and repartition protocols, indicating the values distributed to variables from the same compartment and its neighbors.

## Introduction

Membrane computing was initiated in Păun (2000) as a new research topic investigating computing models inspired by the architecture and the functioning of the living cells, as individual biological entities as well as parts of higher-order structures, such as tissues or more complex organs. The models are called **membrane systems** or **P systems**. Membrane computing is a branch of natural computing, and the main developments in this area are summarized in the handbook of natural computing (see Păun (2012)).

A thorough account of the main theoretical topics and significant applications is provided in Păun et al. (2010a).

Membrane computing models have been used in various applications in linguistics, computer graphics, economics, approximate optimizations, cryptography, and other computer science areas, in Ciobanu et al. (2006). The most prominent applications are related to systems and synthetic biology (see Frisco et al. (2014)) and, more recently, to real-life complex problems, presented in Zhang et al. (2017).

Simple or more complex biological systems have been modeled for a long time using mathematical approaches, such as differential equations and statistical models. In recent years, especially after launching the concept of “executable biology” in Fisher and Henzinger (2007 Nov), computational models have started being used more often in applications in biology. These are attractive modeling approaches as they provide a plethora of formal methods from computer science that allow not only to efficiently simulate the systems, but they also provide tools that are able to investigate the behavior of the systems for various scenarios and input conditions and check properties of the associated models. More well-established computational models, such as process algebras, Petri nets, Boolean networks, statecharts, or newly developed nature-inspired models – brane calculi, membrane systems, and kappa rule-based system – are utilized, according to Bartocci and Lió (2016 Jan), in applications in biology. In Bartocci and Lió (2016 Jan), it is shown that these computational models come with formal analysis methods such as static analysis, runtime verification and monitoring, model checking, and tools supporting them.

P systems, as presented in Ciobanu et al. (2006), have been used initially as computational models for various applications primarily in biology. However, the applicability area has been expanded due to the power and flexibility of the computational devices introduced and studied. A relatively new and very promising application, the controller design for mobile robots, using the characteristics of numerical P systems, was

introduced in Buiu et al. (2012) and Wang et al. (2015c). P system concepts have been also combined with methods and principles developed in the area of soft computing, especially evolutionary algorithms and fuzzy systems, with the aim of solving a broad spectrum of optimization problems. In Zhang et al. (2014b) and Zhang et al. (2017), a comprehensive survey was presented on the combination of P systems with meta-heuristic algorithms, such as local search approaches, evolutionary algorithms, and swarm intelligence algorithms giving rise to membrane algorithms or membrane-inspired evolutionary algorithms, involved in solving optimization problems. In Zhang et al. (2014c), a novel way of designing a P system for directly obtaining the approximate solutions of combinatorial optimization problems and discrete engineering optimization problems without the aid of meta-heuristic algorithms was presented. In Wang et al. (2015a) and Zhang et al. (2017), the combination of P systems with fuzzy logic theory to solve fault diagnosis problems was reviewed in a systematic manner.

In the sequel we show how various instances of the membrane computing models are used in applications. We start with applications of membrane systems in modeling systems and synthetic biology problems, then continue presenting the use of membrane algorithms for solving optimization problems, and finally describe briefly the use of membrane systems in modeling problems from computer science, graphics, and economy and mention some of the tools developed to support the simulation of various P system models.

## Applications in Systems in Synthetic Biology

There are many features of membrane computing models which make them attractive for applications in several disciplines, especially biology. Membrane computing uses a set of concepts derived directly from cell biology: **compartments**, which are independent entities; their **objects**, corresponding to simpler or more complex biochemical elements, such as simple molecules or complex proteins and DNA molecules,

interacting locally via evolution (or transformation or rewriting); and communication **rules**, corresponding to various biochemical interactions—complex formation or its reverse transformation, membrane interaction, or transport. The behavior of such systems may be quite complex, nonlinear, and emerging from local interactions within compartments or inter-compartment. Both qualitative and quantitative aspects of their behavior are expressed with nondeterministic and stochastic models, respectively. Such models are presented in Ciobanu et al. (2006) and Frisco et al. (2014) covering a broad spectrum of applications.

In the early monograph on applications of membrane computing (Ciobanu et al. (2006)), there are chapters dedicated to modeling problems in biology. In these applications membrane computing is used as a **formal language** allowing to describe in a rigorous and precise way the biochemical entities and their interactions. I. I. Ardelean, D. Besozzi, M. H. Garzon, G. Mauri, and S. Roy present a P system model for mechanosensitive channels; L. Bianco, F. Fontana, G. Franco, and V. Manca discuss P systems for biological dynamics; M. Cavaliere and I. I. Ardelean introduce a P system model to simulate respiration and photosynthesis interaction in cyanobacteria; G. Cioban models cell-mediated immunity and T. Y. Nishida photosynthesis; a multi-set processing model of p53 signaling pathway is presented by Y. Suzuki and H. Tanaka.

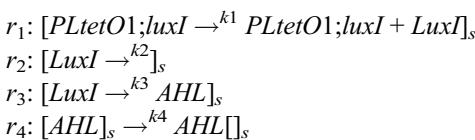
Most of the membrane systems presented in Frisco et al. (2014) and used for systems and synthetic biology applications are stochastic, each rule having associated a kinetic constant based on which a probability is calculated. The execution strategy, the model semantics, is based on Gillespie algorithm. In accordance with this semantics, the probabilities of the rules are evaluated at every step of execution as these are computed taking into account the kinetic constants and the current concentrations of the reactants involved.

Firstly, we present a stochastic P system model for a *pulse generator* example following Frisco et al. (2014), and then we introduce a formal verification approach for this system. Finally, we describe some other P system models used for various systems and synthetic biology problems, presented in Frisco et al. (2014).

The pulse generator (by J. Blakes, J. Twycross, S. Konur, F. J. Romero-Campero, N. Krasnogor, and M. Gheorghe, in Frisco et al. (2014)), a synthetic bacterial colony, consists of two different bacterial strains, *sender cells* and *pulsing cells*:

- *Sender cells* contain the gene *luxI* from *Vibrio fischeri*, codifying the enzyme *LuxI* which synthesizes the molecular signal *3OC6-HSL* (*AHL*). The *luxI* gene is expressed by the promoter *PLtetO1* from the tetracycline resistance transposon.
- *Pulsing cells* contain the *lux R* gene from the same organism, *Vibrio fischeri*, that codifies the *3OC6-HSL* receptor protein *LuxR*. This gene is regulated by the promoter *PluxL*. These cells also contain the gene *c I* from lambda phage codifying the repressor *C I* under the regulation of the promoter *PluxR*, activated by the binding of the transcription factor *LuxR\_3OC6\_2*. These bacterial strains carry the gene *gfp* that codifies the green fluorescent protein under the regulation of the synthetic promoter *PluxPR* combining the *Plux* promoter (activated by the transcription factor *Lux R\_3OC6\_2*) and the *PR* promoter from lambda phage (repressed by the transcription factor *C I*).

The rules describing the interactions within sending cells are:



The rules appear within a compartment, denoted *s*, expressing interactions within the compartment: *luxI* gene expressed by the promoter *PLtetO1* produces the enzyme *LuxI* (*r*<sub>1</sub>); *LuxI* might degrade (*r*<sub>2</sub>); *LuxI* synthesizes the signal *3OC6-HSL* (*AHL*) (*r*<sub>3</sub>); and this signal is communicated to a neighbor (*r*<sub>4</sub>). Following the description of a pulsing cell, one can derive in a similar manner the rules for such a cell.

The pulse generator system consists of a lattice of cells distributed in accordance with a specific spatial arrangement. At one end there are located

the sender cells and the rest are pulsing cells. The sender cells will start producing the molecular signal *3OC6-HSL* (*AHL*). This will get initially accumulated in sending cells and then through the communication rules, *r*<sub>4</sub>, will start propagating *3OC6-HSL* toward the pulsing cells. These in turn, in the presence of *3OC6-HSL*, will start producing the green fluorescent protein, *GFP*. The model, described in Chap. 1 of Frisco et al. (2014), consists of a two-dimensional lattice, each cell being a stochastic P system, corresponding to either a sending cell or a pulsing cell. A simulation has been made for a lattice with 341 compartments (11 × 31) and 28 molecular species.

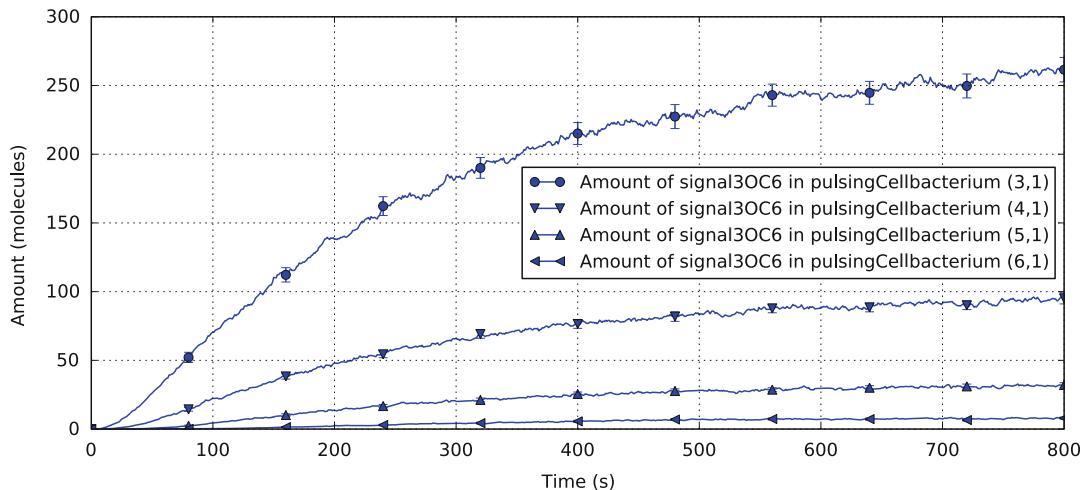
Figure 1 shows the signal molecule *3OC6-HSL* amount over time. The figure, initially presented in Chap. 1 of Frisco et al. (2014), suggests that the further away the pulsing cells are from the sender cells, the less likely they are to produce a pulse, the green fluorescent protein.

Figure 1 shows the result of one single simulation. As this is a stochastic system, then several simulations need to be performed in order to compute an average behavior. By using some of the analysis methods associated with computational models, in the case of stochastic P systems, the model checking tool, PRISM, one can get more precise information regarding the behavior of the pulse generator. Let us consider the concentration of *GFP* at row 3 within 50 s. This can be formulated as “probability that *GFP* concentration at row 3 exceeds 100 within 50 s,” and in PRISM language, this becomes

$$P_{=?}[true \ U^{<50} GFP\_pulsing\_3 \geq 100].$$

This confirms the behavior shown in Fig. 1, but it also returns a high probability of this event, 0.87, as presented in Frisco et al. (2014).

This example from synthetic biology shows the capabilities of membrane systems to model biological systems using features such as compartments, biochemical interactions, and inter-compartment communications, a direct way of writing biochemical reactions through evolution and communication rules. The stochastic behavior of the system is studied through simulations.



**Applications of P Systems, Fig. 1** Signal 3OC6–HSL level over time (From Frisco et al. (2014))

Properties of the system can be systematically investigated and emergent behavior predicted. Other applications have been considered in Frisco et al. (2014) and adequate membrane system models developed in this respect.

- A biological system linked to breast cancer is modeled as a membrane system with peripheral proteins (M. Cavalieri, T. Mazza and S. Sedwards). This model allows to simulate the behavior of the system, revealing the role of estrogen in cellular mitosis and DNA damage. The use of a stochastic model checker facilitates the investigation of the appropriate time-dependent dosage of antagonist that is meant to be used to minimize the random replication of abnormal cells.
- An application of membrane systems to the study of intracellular diffusive processes is presented by P. Cazzaniga, D. Besozzi, D. Pescini, and G. Mauri. A signal transduction pathway of bacterial chemotaxis is investigated by considering first a single-volume module revealing the effects of different perturbations on the system dynamics and then a multivolume module, extending the former, that looks at diffusive processes leading to the formation of concentration gradients within the bacterial cytoplasm. The membrane computing model is also a stochastic one, called  $\tau$ -DPP

model. The simulation of the model helps analyzing the bacterial chemotaxis by showing the stochastic fluctuations of various proteins and the distribution of flagellar motors over the cell surface.

- A stochastic membrane computing model suitable for real ecosystem applications, called population dynamics P systems (PDP systems, for short) is considered by M. A. Colomer-Cugat, M. García-Quismondo, L. F. Macías-Ramos, M. A. Martínez-del-Amor, I. Pérez-Hurtado, M. J. Pérez-Jiménez, A. Riscos-Núñez, and L. Valencia-Cabrera. The formal definition of the syntax and semantics of the PDP systems together with a simulation platform, *MeCoSim* (standing for membrane computing simulator), are provided. The PDP systems are applied to modeling pandemic dynamics of three populations.
- T. Hinze, J. Behre, C. Bodenstein, G. Escuela, G. Grünert, P. Hofstedt, P. Sauer, S. Hayat, and P. Dittrich present three chronobiological studies modeled with different types of membrane systems capturing spatiotemporal aspects of the biological systems. Using a KaiABC core oscillator, a cell signaling network representing a posttranslational prototype for generation of circadian rhythms is investigated. The second study is dedicated to the circadian clockwork able to adapt its

oscillation to an external stimulus. The third system investigated is a bistable toggle switch resulting from mutual gene regulation. The P systems used as models of these systems are specified and analyzed with a software package, called SRSim. The software package allows the use of spatial interaction rules and provides powerful visualization capabilities.

- A membrane system using an execution strategy for its rules based on a new algorithm, called nondeterministic waiting time method, is introduced by J. Jack, A. Păun, and M. Păun. Some case studies from system biology are used to compare the method with stochastic membrane systems and ordinary differential equations. A FAS-induced apoptosis is then modeled and studied with these membrane systems.
- The class of MP systems is introduced together with a method based on regression analysis that synthesizes a MP model from time series of observed dynamics (L. Marchetti, V. Manca, R. Pagliarini, and A. Bollig-Fischer). This model is applied to two systems biology case studies. The first case study is about the glucose/insulin interactions related to the intravenous glucose tolerance test. The second one presents gene expression networks involved in breast cancer.
- The behavior of *E. coli* bacterium with respect to different levels of oxygen in the environment is investigated using an agent-based approach (A. Turcanu, L. Mierlă, F. Ipate, A. Stefanescu, H. Bai, M. Holcombe, and S. Coakley). The behavior of the system is not only studied through simulations of the agent-based model, but certain properties, formulated in a temporal logic formalism, are investigated through two model checkers, Rodin and Spin. The translation of the agent-based system is not directly into these model checkers, but it comes via a specific class of membrane systems, called kernel P systems. These membrane systems allow a direct translation of their transformation rules into transitions of the model checkers.

## Applications to Real-Life Complex Problems

The real-life complex problems discussed in this section are of three types: engineering optimizations, fault diagnosis, and robot control.

### Membrane Algorithms

Until now two kinds of membrane algorithms, also called membrane-inspired evolutionary algorithms, have been discussed. One is a proper hybridization of P system features – the hierarchical or network membrane structures, evolution rules and computational procedures, and various well-established meta-heuristic methods such as local search approaches, evolutionary algorithms, and swarm intelligence algorithms. The other type includes optimization spiking neural P systems (OSNPs), which were designed by using spiking neural P systems for directly obtaining the approximate solutions of optimization problems without the aid of meta-heuristic algorithms. In what follows, we start from the first kind of membrane algorithms and then describe the second one.

A very promising direction of research, namely, applying membrane computing in devising approximate algorithms for solving hard optimization problems, was initiated by Nishida, in Nishida (2004), who proposed *membrane algorithms* as a new class of distributed evolutionary algorithms. These algorithms can be considered as a high-level (distributed and dynamically evolving their structure during the computation) evolutionary algorithms. In short, candidate solutions evolve in compartments of a (dynamical) membrane structure according to local algorithms, with better solutions migrating through the membrane structure; after a specified halting condition is met, the current best solution is extracted as the result of the algorithm.

Nishida has checked this strategy for the traveling salesman problem, and the results were more than encouraging for a series of benchmark problems: the convergence is very fast, the number of membranes is rather influential on the quality of the solution, the method is reliable, and both the average quality and the worst solutions were good enough and always better than the average and the worst solutions given by simulated annealing.

This combination of two nature-inspired computational approaches has led to many variants of such models and applications of various types. A survey of the results obtained is published in Zhang et al. (2014b). Real-life applications of this approach have been reported in Zhang et al. (2017). A membrane algorithm is a successful instance of a model linking membrane computing and evolutionary algorithms. In Zhang et al. (2014a), the role played by P systems in membrane algorithms was discussed. Dynamic behaviors of membrane algorithms by introducing a set of population diversity and convergence measures were analyzed. This is the first attempt to obtain deep insights into the search capabilities of membrane algorithms. The analysis is performed on the membrane algorithm, QEPS (a quantum-inspired evolutionary algorithm based on membrane computing), and its counterpart algorithm, QIEA (a quantum-inspired evolutionary algorithm), using a comparative approach in an experimental context to better understand their characteristics and performances. Experiments are performed on the knapsack problems. The comparison with respect to the diversity measure, Hamming distance between the best and worst binary individuals ( $D_{hbw}$ ) in a population between QEPS and QIEA, is shown in Fig. 2. The best fitness convergence ( $C_{fb}$ ) comparison is shown in

**Applications of P Systems, Fig. 2**  $D_{hbw}$  of QEPS and QIEA on the knapsack problem with 800 items (From Zhang et al. (2017))

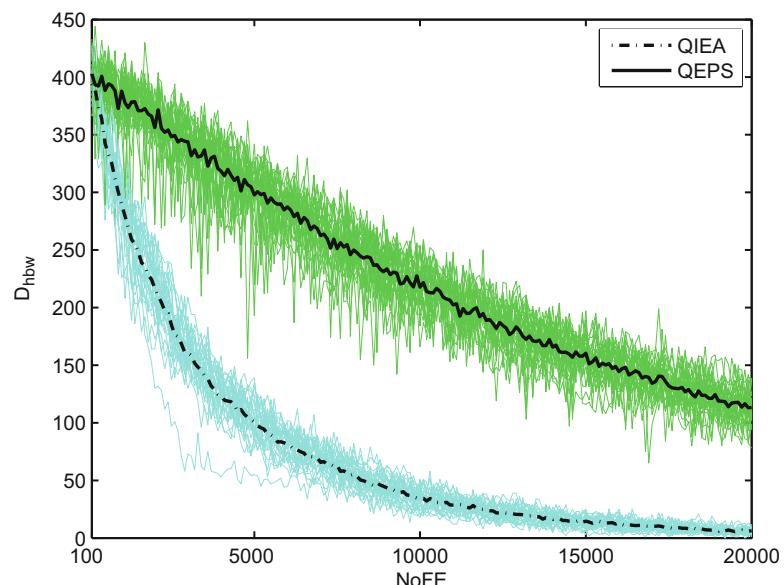
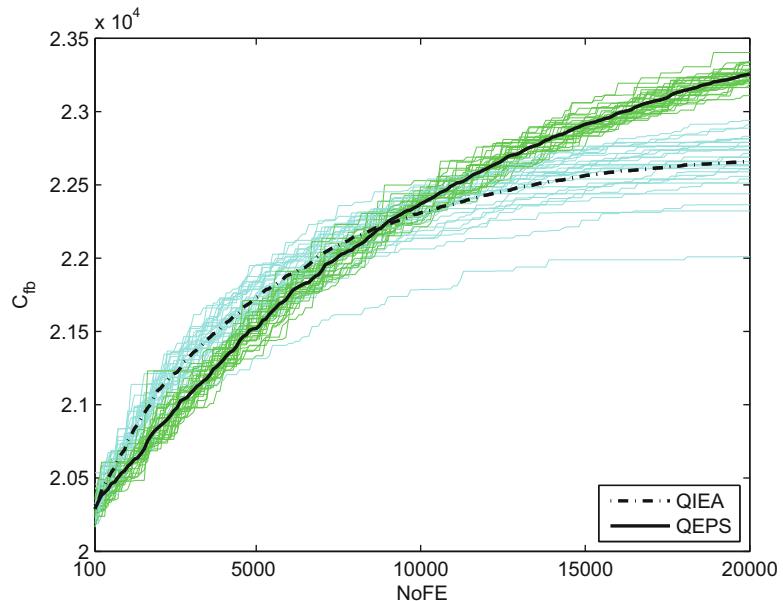


Fig. 3. NoFE represents the number of function evaluations. It is shown that QEPS can achieve better balance between convergence and diversity than QIEA, which indicates QEPS has a stronger capacity of balancing exploration and exploitation than QIEA in order to prevent premature convergence that might occur.

In the sequel we use these references for presenting several types of membrane algorithms and then, using the later reference, describe some applications dealing with complex industrial problems of optimization.

Now this area has reached a certain level of maturity, and the focus is on developing new variants of meta-heuristics by using different membrane structures, types of evolution rules and other computational capabilities of membrane systems, and various evolutionary computation methods. Four types of hierarchical structures and two network structures of the membrane systems are used Zhang et al. (2014b). The membrane algorithms using hierarchical membrane systems are membrane algorithms with nested membranes (NMS, for short), one-level membrane structure (OLMS), hybrid membrane structure (HMS), and dynamic membrane structure (DMS). The membrane algorithms using network membrane structures are membrane algorithms with static network structure (SNS) and dynamic

**Applications of P Systems, Fig. 3**  $C_{fb}$  of QEPS and QIEA on the knapsack problem with 800 items (From Zhang et al. (2017))



network structure (DNS). In each of these cases, various meta-heuristic approaches have been considered, including tabu search, genetic algorithms, quantum-inspired evolutionary algorithms, ant colony optimization, differential evolution, particle swarm optimization, and so on.

The network structure considered for membrane systems is called tissue, and the models are called tissue P systems or neural P systems. In the sequel we refer to membrane algorithms designed with a tissue P system structure and differential evolution (DETPS) approach presented in Zhang et al. (2013).

In Zhang et al. (2017), experiments with five constrained problems having a mixture of linear or quadratic objective functions and constraints are presented. DETPS results are compared with those obtained by using hybrid immune-hill climbing algorithm (HIHC), genetic algorithm with an artificial immune system (GAIS), genetic algorithm based on immune network modeling (GAINM), teaching-learning-based optimization (TLBO), multi-membered evolutionary strategy (M-ES), particle evolutionary swarm optimization (PESO), cultural differential evolution (CDE), coevolutionary differential evolution (CoDE), and artificial bee colony (ABC).

We present from Zhang et al. (2017) the problem below with nine linear constraints and a quadratic objective function. Two types of experiments are performed with different halting criteria and using different sets of optimization algorithms. In both cases DETPS is used. In addition to this, in the first case, DETPS, HIHC, GAIS, and GAINM are used, whereas in the second one, TLBO, M-ES, PESO, VDE, CoDE, and ABC are used.

#### Problem:

$$\min f(x) = 5 \sum_{i=1}^4 x_i - 5 \sum_{i=1}^4 x_i^2 - \sum_{i=5}^{13} x_i \quad (1)$$

subject to

$$\begin{cases} g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\ g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\ g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\ g_4(x) = -8x_1 + x_{10} \leq 0 \\ g_5(x) = -8x_2 + x_{11} \leq 0 \\ g_6(x) = -8x_3 + x_{12} \leq 0 \\ g_7(x) = -2x_4 - x_5 + x_{10} \leq 0 \\ g_8(x) = -2x_6 - x_7 + x_{11} \leq 0 \\ g_9(x) = -2x_8 - x_9 + x_{12} \leq 0 \end{cases}$$

where  $0 \leq x_i \leq 1$ ,  $i = 1, 2, 3, \dots, 9$ ;  $0 \leq x_i \leq 100$ ,  $i = 10, 11, 12$ ; and  $0 \leq x_{13} \leq 1$ . The optimal

solution is  $f(x^*) = -15$  at  $x^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ .

In Table 1, the results of using the first set of optimization algorithms are presented, and in Table 2, the results of running the second set are presented. One can observe that DETPS produces similar results to the other algorithms (Table 2) if not better (Table 1). In both sets of experiments, DETPS arrives at the solution with less function evaluations (Table 1) or significantly less (Table 2).

Membrane algorithms are successfully applied to optimization problems in engineering in Zhang et al. (2017). An OLMS with a quantum-inspired evolutionary algorithm is applied to analyze radar signals and solve the image sparse decomposition problem. OLMS with a particle swarm optimization algorithm is used to optimize the design of a proportional integral derivative controller and mobile robot path planning.

DETPS, presented earlier, is applied for manufacturing parameter optimization problem. A DNS based on population P systems, a variant of tissue P systems with dynamic structure, with a quantum-inspired evolutionary algorithm is used to solve the distribution network reconfiguration problem in power systems. A SNS based on

neural P systems algorithm is used to solve the power system fault diagnosis problem.

The synthesis of various classes of P systems solving different problems has been investigated with evolutionary methods. A survey of this work is presented in Zhang et al. (2014b).

The membrane algorithms discussed above were designed by combining P systems with meta-heuristic techniques. In what follows, an optimization algorithm, optimization spiking neural P system (OSNPS), is directly derived from membrane computing models, spiking neural P systems (SN P systems) in Zhang et al. (2014c).

Inspired by the behavior of the language generating SN P systems, OSNPS, as shown in Fig. 4, was constructed by using a family of extended SN P systems (ESNPS) and a guider to adaptively adjust rule probabilities. ESNPS, as shown in Fig. 5, is designed by introducing the probabilistic selection of evolution rules and the output collection from multiple neurons. In each compartment there are rewriting ( $r_i^1$ ) and forgetting ( $r_i^2$ ) rules,  $1 \leq i \leq m$ .

The viability and effectiveness of OSNPS were tested on knapsack problems by considering genetic quantum algorithm (GQA), quantum-inspired evolutionary algorithm (QEA), novel

**Applications of P Systems, Table 1** Statistical results of DETPS, HIHC, GAIS, and GAINM to test Problem. Best, mean, worst, and SD represent the best solution,

mean best solution, worst solution, and standard deviation over independent 30 runs, respectively

Methods	Best	Mean	Worst	SD	Function evaluations
DETPS	-15.0000	-15.0000	-15.0000	2.2362e-6	64,156
HIHC	-15	-14.8266	-14.3417	0.145	120,000
GAIS	-14.7841	-14.5266	-13.8417	0.2335	150,000
GAINM	-5.2735	-3.7435	-2.4255	0.9696	150,000

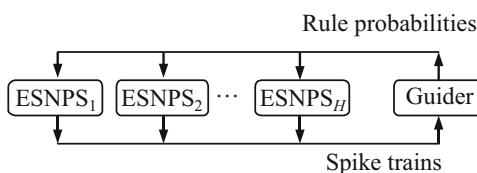
**Applications of P Systems, Table 2** Statistical results of seven algorithms to test the problem. Best, mean, and worst represent the best solution, mean best solution, and worst solution over independent 30 runs, respectively

Methods	Best	Mean	Worst	Function evaluations
DETPS	-15.0	-15.0	-15.0	20,875
TLBO	-15.0	-15.0	-15.0	25,000
M-ES	-15.0	-15.0	-15.0	240,000
PESO	-15.0	-15.0	-15.0	350,000
CDE	-15.0	-15.0	-15.0	100,100
CoDE	-15.0	-15.0	-15.0	248,000
ABC	-15.0	-15.0	-15.0	240,000

quantum evolutionary algorithm (NQEA), quantum-inspired evolutionary algorithm based on P systems (QEPS), and two membrane-inspired evolutionary algorithms with quantum-inspired subalgorithms (MAQIS<sub>1</sub> and MAQIS<sub>2</sub>) as benchmark algorithms in the experiments. Eleven knapsack problems, labeled 1–11, with 1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400, 2600, 2800, and 3000 items are used. GQA, which obtained the worst performance among the seven algorithms, is regarded as benchmarks to illustrate the percentage of the improvements of QEA, NQEA, QEPS, OSNPS, MAQIS<sub>1</sub>, and MAQIS<sub>2</sub>. The experimental results show that OSNPS is superior or competitive to the other six optimization approaches, GQA, QEA, NQEA, QEPS, MAQIS<sub>1</sub>, and MAQIS<sub>2</sub>, with respect to the best solutions over 11 problems and 30 independent runs.

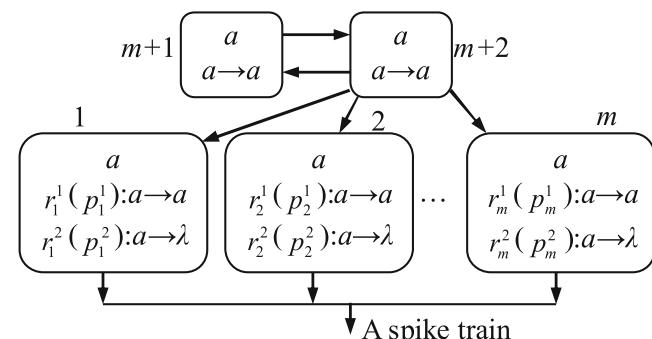
### Fault Diagnosis and Robot Control

Fault diagnosis problems, especially in electric power systems, can be solved by using fuzzy reasoning spiking neural P systems (short for FRSNP), a graphic modeling approach combining spiking neural P systems and fuzzy logic theory in Peng et al. (2013), Wang et al. (2015b), and Zhang et al.



**Applications of P Systems, Fig. 4** OSNPS (From Zhang et al. (2017))

**Applications of P Systems, Fig. 5** ESNPS structure (From Zhang et al. (2017))



(2017)). FRSNP models offer an intuitive description of a system based on fuzzy logics, good fault tolerance principles, a complete set of relationships between protective devices and faults, and an understandable diagnosis model-building process. In what follows, we will refer to FRSNP models with trapezoidal fuzzy numbers.

In Wang et al. (2015b) and Zhang et al. (2017), the FRSNP model with trapezoidal fuzzy numbers is used in experiments carried out on seven cases of the local system in an electric power system. It is shown that this FRSNP model obtains better or competitive results, compared with four methods reported in the literature, fuzzy logic, fuzzy Petri nets, genetic algorithm-tabu search, and genetic algorithm.

Enzymatic numerical P systems have been considered in modeling the robot controller in Zhang et al. (2017). These classes of P systems introduce significantly new features to the standard membrane computing framework. Membrane controller P systems change the multiset rewriting mechanism, widely utilized by the transformation rules, with a computational procedure based on a distributed way of calculating different polynomials.

Membrane controllers for mobile robots use a hierarchical cognitive architecture and are designed by using the syntax (such as the membrane structure, initial multisets, and evolution rules) and semantics of numerical P systems (NPS) or enzymatic numerical P systems (ENPS). In NPS and ENPS, the values of the variables received from various sensors are real-valued numbers, and the computation is performed on real-valued variables.

Algorithm 1 shows the generic structure of membrane controllers in Buiu et al. (2012). In the first step, the robot parameters are set to initial values. In the second step, real-time data are read from the sensors. The NPS corresponding to the current membrane computing process is executed in the third step. Finally, the motors' speeds are set. Then the execution of the controller goes back to the second step.

### Algorithm 1 Generic structure of membrane controllers

```
Require: parameters =
initialiseBehaviourParameters(robot,
behaviour)
While (True)
    sensors = readSensors(robot)
    # simulate Numerical P System on the
    network server
    query = constructQuery(robot,
    behaviour, sensors, parameters)
    response = queryWebApp(address,
    query)
    speed = extractContent(response)
    setSpeed(robot, speed)
End While
```

Suppose that  $u$  and  $y$  are the inputs and outputs of a membrane controller and  $r$  is the predefined setpoints. A standard membrane controller (MeC), defined in Buiu et al. (2012), is used as a P system model for designing robot controllers.

In Buiu et al. (2012) and Wang et al. (2015c), several examples using robot controllers designed with NPS or ENPS models are presented. They deal with features such as obstacle avoidance, wall following, following another robot, and trajectory tracking.

## Other Applications

In Ciobanu et al. (2006) applications of membrane systems in computer science are presented. Different sorting algorithms (A. Alhazov and D. Sburlan) are considered, and their solutions make use of the massive parallelism of the

membrane system models; a public key protocol (O. Michel and F. Jaquemard) is described in the framework of membrane systems. Applications were also reported in computer graphics (A. Georgiou, M. Gheorghe, and F. Bernardini) – where the compartmentalization seems to add a significant efficiency to well-known techniques based on L systems, linguistics, both as a representation language for various concepts related to language evolution, dialog, and semantics (G. Bel Enguix, M. D. Jiménez-Lopez) and as a parsing tool utilizing a new concept of automata-based translation where the parallelism of the model is exploited (R. Gramatovici and G. Bel Enguix).

The additional bibliography consists of papers referring to other applications of membrane systems. Some of the most significant are in economics (where many biochemical metaphors find a natural counterpart, with the mentioning that the “reactions” which take place in economics, for instance, in market-like frameworks, are not driven only by probabilities/stoichiometric calculations but also by psychological influences, which make the modeling still more difficult than in engineering or biological applications). R. Niculescu and his collaborators have utilized in a series of papers various extensions of the membrane systems to solve problems of synchronization, of broadcasting, or of representing complex data structures with compact and efficient membrane systems-based notations.

For most of the applications presented so far, there have been developed software tools helping analyzing the systems investigated. Simulators for the models built for different types of P systems and, in some cases, tools revealing properties of the systems are provided. Here there will be mentioned the most established software tools.

P-Lingua tool ([http://www.p-lingua.org/wiki/index.php/Main\\_Page](http://www.p-lingua.org/wiki/index.php/Main_Page)) provides a domain-specific language, with the same name, allowing to describe models for a large set of P systems, including those presented in Chap. 4 of Frisco et al. (2014). The P-Lingua language is supported by a dashboard allowing to handle input and output data associated with the models and data visualization. This is integrated with a tool, MeCoSim

(<http://www.p-lingua.org/mecosim/>), that provides connections to various run time platforms.

Stochastic P systems presented in Chap. 1 of Frisco et al. (2014) can be described in a specific format and then simulated within the Infobiotics software platform (<http://infobiotics.org>). This allows to specify and verify properties of the system expressed as probabilistic temporal logic formulas.

Deterministic models written as metabolic P systems, presented in Chap. 7 of Frisco et al. (2014), are simulated with MetaPlab (<http://mplab.sci.univr.it/index.html>). The tool provides support for data visualization but also calibrates the model based on observed time series referring to the system investigated.

SRSim simulator (<http://www.biosys.uni-jena.de/members/gerd+gruenert/srsim.html>) allows to analyze P systems with spatiotemporal features, as those presented in Chap. 5 of Frisco et al. (2014).

Some more details about these tools, including the domain-specific languages they use, the simulators employed, and other features helping the analysis of the investigated system, are presented in the additional bibliography.

## Future Directions

We present future developments of membrane computing applications, membrane algorithms, and finally FRSNP models and robot control models, as reflected in Gheorghe et al. (2013) and Zhang et al. (2017).

In Gheorghe et al. (2013) some future developments of the membrane computing pointing out to applications of P systems as well are presented. The most important developments with respect to applications are:

1. The analysis methods for P system models – formal verification and testing, causality, and semantics – and the efficient and robust development of tools related to numerical P system models will be considered.
2. The tool development will continue, especially with an emphasis on parallel implementations

and their integration with existing tools. New parallelization algorithms should be considered, given the complexity of the models and the systems investigated.

3. More complex and diverse problems, requiring both modeling and optimization aspects, are expected to be approached with the methods and tools developed so far or with new ones.

Future developments regarding membrane algorithms, as listed in Gheorghe et al. (2013), are as follows:

1. Further combinations of a larger set of evolutionary algorithms with various classes of membrane computing models such as cell P systems with active membranes, tissue P systems, and population P systems are expected to be studied.
2. Usually, in a membrane algorithm, an evolutionary algorithm is placed inside a membrane. Given the distributed structure of a P system, it is natural to consider several different types of evolutionary operators, or several distinct kinds of evolutionary mechanisms, such as a genetic algorithm, evolutionary programming, evolution strategy, differential evolution, and particle swarm optimization acting in parallel across the model. Furthermore, the flexible communication rules can be used at the level of genes, instead of at the level of individuals.
3. The single-objective problems are usually involved in the investigations reported so far in the literature. It is worth investigating how multi-objective, dynamic, peaked optimization problems might lead to better solutions by using the P systems environment.
4. More real-world application problems, such as power system optimization, software/hardware co-design, and vehicle route plan, might benefit from a membrane algorithm-based approach.
5. A deeper and fine grain performance analysis and evaluation of membrane algorithms starting from the work in Zhang et al. (2014a) is requested in order to reveal more precisely the role played by P systems in the hybrid optimization algorithms.

New research developments regarding FRSNP models are identified in Zhang et al. (2017). New variants of SN P systems and their reasoning algorithms are expected to be investigated in connection with more complex fault diagnosis problems, such as online diagnosis, fast fault diagnosis, high-precision diagnosis, as well as power supply systems for urban rail transit, mechanical fault diagnosis, and power systems with new energies. Another promising research avenue is the usage of FRSNP models with learning abilities.

The future work with respect to the use of P systems in supporting robot controller design is reported in Zhang et al. (2017):

1. The design of membrane controllers for more complex behaviors of a larger class of wheeled robots
2. The applicability of numerical P systems to other advanced robot control strategies with applications in various engineering areas
3. The use of P systems, including numerical P systems, in constructing cognitive and executable architectures (planning, learning, execution) of autonomous robots, as stated in Buiu et al. (2012), whereby P systems-based cognitive architectures at higher levels in a control system can communicate with lower level membrane controllers

**Acknowledgments** The work of G. Zhang was supported by the National Natural Science Foundation of China (61373047 and 61672437) and the Research Project of Key Laboratory of Fluid and Power Machinery (Xihua University), Ministry of Education, P. R. China (JYBFXYQ-1).

## Bibliography

### Primary Literature

- Bartocci E, Lió (2016) Computational modeling, formal analysis, and tools for systems biology. *PLoS Comput Biol* 21(1):e1004,591
- Buiu C, Vasile CI, Arsene O (2012) Development of membrane controllers for mobile robots. *Inf Sci* 187:33–51
- Ciobanu G, Păun Gh, Pérez-Jiménez MJ (eds) (2006) Applications of membrane computing. Natural computing series. Springer, Berlin

- Fisher J, Henzinger T (2007) Executable cell biology. *Nat Biotechnol* 25(11):1239–1249
- Frisco P, Gheorghe M, Pérez-Jiménez MJ (eds) (2014) Applications of membrane computing in systems and synthetic biology. Emergence, complexity and computation. Springer, Cham
- Gheorghe M, Păun G, Pérez-Jiménez MJ, Rozenberg G (2013) Research frontiers of membrane computing: open problems and research topics. *Int J Found Comput Sci* 24(5):547–624
- Nishida TY (2004) An application of P system: a new algorithm for NP-complete optimization problems. In: Proceedings of the 8th world multi-conference on systems, cybernetics and informatics, vol 5, pp 109–112
- Păun G (2000) Computing with membranes. *J Comput Syst Sci* 61(1):108–143. also Turku Center for Computer Science Report TUCS 208, Nov 1998
- Păun GH (2012) Membrane computing. In: Rozenberg G, Bäck T, Kok JN (eds) Handbook of natural computing. Springer, Berlin, pp 1355–1377
- Păun G, Rozenberg G, Salomaa A (eds) (2010a) The Oxford handbook of membrane computing. Oxford University Press, Oxford
- Peng H, Wang J, Pérez-Jiménez MJ, Wang H, Shao J, Wang T (2013) Fuzzy reasoning spiking neural P system for fault diagnosis. *Inf Sci* 235:106–116
- Wang T, Zhang G, Pérez-Jiménez MJ (2015a) Fuzzy membrane computing: theory and applications. *Int J Comput Commun* 10:904–935
- Wang T, Zhang G, Zhao J, He Z, Zhao J, Wang J, Pérez-Jiménez MJ (2015b) Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems. *IEEE T Power Syst* 30(3):1182–1194
- Wang X, Zhang G, Neri F, Jiang T, Zhao J, Gheorghe M, Ipaté F, Lepticaru R (2015c) Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots. *Integr Comput Aided Eng* 23(1):15–30
- Zhang G, Cheng J, Gheorghe M, Meng Q (2013) A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems. *Appl Soft Comput* 13(3):1528–1542
- Zhang G, Cheng J, Gheorghe M (2014a) Dynamic behavior analysis of membrane-inspired evolutionary algorithms. *Int J Comput Commun* 9(2):227–242
- Zhang G, Gheorghe M, Pan L, Pérez-Jiménez MJ (2014b) Evolutionary membrane computing: a comprehensive survey and new results. *Inf Sci* 279:528–551
- Zhang G, Rong H, Neri F, Pérez-Jiménez MJ (2014c) An optimization spiking neural P system for approximately solving combinatorial optimization problems. *Int J Neural Syst* 24(5):1–16
- Zhang G, Pérez-Jiménez MJ, Gheorghe M (2017) Real-life applications with membrane computing. Emergence, complexity and computation. Springer, Cham

**Books and Reviews**

- del Amor MAM, García-Quismondo M, Macías-Ramos LF, Valencia-Cabrera L, Nez ARN, Pérez-Jiménez MJ (2015) Simulating P systems on GPU devices: a survey. *Fundam Inform* 136(3):269–284
- Dinneen MJ, Kim YB, Nicolescu R (2012) Faster synchronization in P systems. *Nat Comput* 11(1):107–115
- Freund R, Păun G, Rozenberg G, Salomaa A (eds) (2006) Membrane computing, 6th international workshop, WMC 2005, Vienna, 18–21 July 2005, Revised selected and invited papers, Lecture notes in computer science, vol 3850, Springer
- Leporati A, Rozenberg G, Salomaa A, Zandron C (eds) (2017) Membrane computing – 17th international conference, CMC 2016, Milan, 25–29 July 2016, Revised selected papers, Lecture notes in computer science, vol 10105, Springer
- Macías-Ramos LF, Valencia-Cabrera L, Song B, Song T, Pan L, Pérez-Jiménez MJ (2015) A P\_Lingua based simulator for P systems with symport/antiport rules. *Fundam Inform* 139(2):211–227
- Manca V (2013) Infobiotics – information in biotic systems. Emergence, complexity and computation. Springer, Heidelberg
- Nicolescu R, Istrate F, Wu H (2013) Programming P systems with complex objects. In: Alhazov A, Cojocaru S, Gheorghe M, Rogohzin Y, Rozenberg G, Salomaa A (eds) Membrane computing, international conference, CMC 2013, Chișinău, 20–23 Aug 2013, Revised papers, Springer, Lecture notes in computer science, vol 8340, pp 280–300
- Păun GH, Pérez-Jiménez MJ, Riscos-Núñez A, Rozenberg G, Salomaa A (eds) (2010b) Membrane computing, 10th international workshop, WMC 2009, Curtea de Arges, 24–27 Aug 2009. Revised selected and invited papers, Lecture notes in computer science, vol 5957, Springer



## Social Algorithms

Xin-She Yang  
School of Science and Technology, Middlesex University, London, UK  
Department of Engineering, University of Cambridge, Cambridge, UK

### Article Outline

Glossary  
Introduction  
Algorithms and Optimization  
Social Algorithms  
Algorithm Analysis and Insight  
Future Directions  
Bibliography

### Glossary

**Algorithm** An algorithm is a step-by-step, computational procedure or a set of rules to be followed by a computer in calculations or computing an answer to a problem.

**Ant colony optimization** Ant colony optimization (ACO) is an algorithm for solving optimization problems such as routing problems using multiple agents. ACO mimics the local interactions of social ant colonies and the use of chemical messenger – pheromone to mark paths. No centralized control is used and the system evolves according to simple local interaction rules.

**Bat algorithm** Bat algorithm (BA) is an algorithm for optimization, which uses frequency-tuning to mimic the basic behavior of echolocation of microbats. BA also uses the variations of loudness and pulse emission rates and a solution vector to a problem corresponds to a position vector of a bat in the search space. Evolution of solutions follow two algorithmic equations for positions and frequencies.

**Bees-inspired algorithms** Bees-inspired algorithms are a class of algorithms for optimization using the foraging characteristics of honeybees and their labor division to carry out search. Pheromone may also be used in some variants of bees-inspired algorithms.

**Cuckoo search** Cuckoo search (CS) is an optimization algorithm that mimics the brood parasitism of some cuckoo species. A solution to a problem is considered as an egg laid by a cuckoo. The evolution of solutions is carried out by Lévy flights and the similarity of solutions controlled by a switch probability.

**Firefly algorithm** Firefly algorithm is an optimization inspired by the flashing patterns of tropical fireflies. The location of a firefly is equivalent to a solution vector to a problem, and the evolution of fireflies follows a non-linear equation to simulate the attraction between fireflies of different brightness that is linked to the objective landscape of the problem.

**Metaheuristic** Metaheuristic or metaheuristic algorithms are a class of optimization algorithms designed by drawing inspiration from nature. They are thus mostly nature-inspired algorithms, and examples of such metaheuristic algorithms are ant colony optimization, firefly algorithm, and particle swarm optimization. These algorithms are often swarm intelligence-based algorithms.

**Nature-inspired algorithms** Nature-inspired algorithms are a much wider class of algorithms that have been developed by drawing inspiration from nature. These algorithms are almost all population-based algorithms. For example, ant colony optimization, bat algorithm, cuckoo search, and particle swarm optimization are all nature-inspired algorithms.

**Nature-inspired computation** Nature-inspired computation is an area of computer science, concerning the development and application of nature-inspired metaheuristic algorithms for optimization, data mining, machine learning, and computational intelligence.

**Objective** An objective function is the function to be optimized in an optimization problem. Objective functions are also called cost functions, loss functions, utility functions, or fitness functions.

**Optimization** Optimization concerns a broad area in mathematics, computer science, operations research, and engineering designs. For example, mathematical programming or mathematical optimization is traditionally an integrated part of operations research. Nowadays, optimization is relevant to almost every area of sciences and engineering. Optimization problems are formulated with one or more objective functions subject to various constraints. Objectives can be either minimized or maximized, depending on the formulations. Optimization can subdivide into linear programming and nonlinear programming.

**Particle swarm optimization** Particle swarm optimization (PSO) is an optimization algorithm that mimics the basic swarming behavior of fish and birds. Each particle has a velocity and a position that corresponds to a solution to a problem. The evolution of the particles is governed by two equations with the use of the best solution found in the swarm.

**Population-based algorithm** A population-based algorithm is an algorithm using a group of multiple agents such as particles, ants, and fireflies to carry out search for optimal solutions. The initialization of the population is usually done randomly and the evolution of the population is governed by the main governing equations in an algorithm in an iterative manner. All social algorithms are population-based algorithms.

**Social algorithms** Social algorithms are a class of nature-inspired algorithms that use some of characteristics of social swarms such as social insects (e.g., ants, bees, bats, and fireflies) and reproduction strategies such as cuckoo-host species coevolution. These algorithms tend to be swarm intelligence-based algorithms. Examples are ant colony optimization, particle swarm optimization, and cuckoo search.

**Swarm intelligence** Swarm intelligence is the emerging behavior of multiagent systems

where multiple agents interact and exchange information, according to simple local rules. There is no centralized control, and each agent follows local rules such as following pheromone trail and deposit pheromone. These rules can often be expressed as simple dynamic equations and the system then evolves iteratively. Under certain conditions, emergent behavior such as self-organization may occur, and the system may show higher-level structures or behavior that is often more complex than that of individuals.

## Introduction

To find solutions to problems commonly used in science and engineering, algorithms are required. An algorithm is a step-by-step computational procedure or a set of rules to be followed by a computer. One of the oldest algorithms is the Euclidean algorithm for finding the greatest common divisor (gcd) of two integers such as 12,345 and 125, and this algorithm was first given in detail in Euclid's Elements about 2300 years ago (Chabert 1999). Modern computing involves a large set of different algorithms from fast Fourier transform (FFT) to image processing techniques and from conjugate gradient methods to finite element methods.

Optimization problems in particular require specialized optimization techniques, ranging from the simple Newton-Raphson's method to more sophisticated simplex methods for linear programming. Modern trends tend to use a combination of traditional techniques in combination with contemporary stochastic metaheuristic algorithms, such as genetic algorithms, firefly algorithm, and particle swarm optimization.

This work concerns a special class of algorithms for solving optimization problems and these algorithms fall into a category: social algorithms, which can in turn belong to swarm intelligence in general. Social algorithms use multiple agents and the "social" interactions to design rules for algorithms so that such social algorithms can mimic certain successful characteristics of the social/biological systems, such as ants, bees,

birds, and animals. Therefore, our focus will solely be on such social algorithms.

It is worth pointing out that the social algorithms in the present context do not include the algorithms for social media, even though algorithms for social media analysis are sometimes simply referred to as “social algorithm” (Lazer 2015). The social algorithms in this work are mainly nature-inspired, population-based algorithms for optimization, which share many similarities with swarm intelligence.

Social algorithms belong to a wider class of metaheuristic algorithms. Alan Turing, the pioneer of artificial intelligence, was the first to use heuristic in his Enigma-decoding work during the Second World War and connectionism (the essence of neural networks) as outlined in his National Physical Laboratory report *Intelligent Machinery* (Turing 1948).

The initiation of nondeterministic algorithms was in the 1960s when evolutionary strategy and genetic algorithm started to appear, which attempted to simulate the key feature of Darwinian evolution of biological systems. For example, genetic algorithm (GA) was developed by John Holland in the 1960s (Holland 1975), which uses crossover, mutation, and selection as basic genetic operators for algorithm operations. At about the same period, Ingo Rechenberg and H. P. Schwefel, developed the evolutionary strategy for constructing automatic experimenter using simple rules of mutation and selection, though crossover was not used. In around 1966, L. J. Fogel and colleagues used simulated evolution as a learning tool to study artificial intelligence, which leads to the development of evolutionary programming. All these algorithms now evolved into a much wider discipline, called evolutionary algorithms or evolutionary computation (Fogel et al. 1966).

Then, simulated annealing was developed in 1983 by Kirkpatrick et al. (1983), which simulated the annealing process of metals for the optimization purpose, and the Tabu search was developed by Fred Glover in 1986 (Glover 1986) that uses memory and history to enhance the search efficiency. In fact, it was Fred Glover who coined the word “metaheuristic” in his 1986 paper.

The major development in the context of social algorithms started in the 1990s. First, Marco Dorigo developed the ant colony optimization (ACO) in his PhD work (Dorigo 1992), and ACO uses the key characteristics of social ants to design procedure for optimization. Local interactions using pheromone and rules are used in ACO. Then, in 1995, particle swarm optimization was developed by James Kennedy and Russell C. Eberhardt, inspired by the swarming behavior of fish and birds (Kennedy and Eberhart 1995). Though developed in 1997, differential evolution (DE) is not a social algorithm; however, DE has used vectorized mutation which forms a basis for many later algorithms (Storn and Price 1997).

Another interesting development is that no-free-lunch (NFL) theorems were proved in 1997 by D.H. Wolpert and W.G. Macready, which had much impact in the optimization and machine learning communities (Wolpert and Macready 1997). This basically dashed the dreams for finding the best algorithms for all problems because NFL theorems state that all algorithms are equally effective if measured in terms of averaged performance for *all* possible problems. Then, researchers realized that the performance and efficiency in practice are not measured by averaging over all possible problems. Instead, we are more concerned with a particular class of problems in a particular discipline, and there is no need to use an algorithm to solve all possible problems. Consequently, for a finite set of problems and for a given few algorithms, empirical observations and experience suggest that some algorithms can perform better than others. For example, algorithms that can use problem-specific knowledge such as convexity can be more efficient than random search. Therefore, further research should identify the types of problems that a given algorithm can solve, or the most suitable algorithms for a given type of problems. Thus, research resumes and continues, just with a different emphasis and from different perspectives.

At the turn of this century, the developments of social algorithms became more active. In 2004, a honeybee algorithm for optimizing Internet hosting centers was developed by Sunil Nakrani and Craig Tovey (2004). In

2005, Pham et al. (2005) developed the bees algorithm, and the virtual bee algorithm was developed by Xin-She Yang in 2005 (Yang 2005). About the same time, the artificial bee colony (ABC) algorithm was developed by Karaboga in 2005 (Karaboga 2005). All these algorithms are bee-based algorithms and they all use some (but different) aspects of the foraging behavior of social bees.

Then, in late 2007 and early 2008, the firefly algorithm (FA) was developed by Xin-She Yang, inspired by the flashing behavior of tropic firefly species (Yang 2008). The attraction mechanism, together with the variation of light intensity, was used to produce a nonlinear algorithm that can deal with multimodal optimization problems. In 2009, cuckoo search (CS) was developed by Xin-She Yang and Suash Deb, inspired by the brood parasitism of the reproduction strategies of some cuckoo species (Yang and Deb 2009). This algorithm simulated partly the complex social interactions of cuckoo-host species coevolution. Then, in 2010, the bat algorithm (BA) was developed by Xin-She Yang, inspired by the echolocation characteristics of microbats (Yang 2010b), and uses frequency-tuning in combination with the variations of loudness and pulse emission rates during foraging. All these algorithms are can be considered as social algorithms because they use the “social” interactions and their biologically inspired rules.

There are other algorithms developed in the last two decades, but they are not social algorithms. For example, harmony search is a music-inspired algorithm (Geem et al. 2001), while gravitational search algorithm (GSA) is a physics-inspired algorithm (Rashedi et al. 2009). In addition, flower pollination algorithm (FPA) is an algorithm inspired by the pollination features of flowering plants (Yang 2012) with promising applications (Yang et al. 2014; Rodrigues et al. 2016). All these algorithms are population-based algorithm, but they do not strictly belong in swarm intelligence-based or social algorithms. A wider range of applications of nature-inspired algorithms can be found in the recent literature (Yang et al. 2015; Yang and Papa 2016; Yang 2018).

As the focus of this work is on social algorithms, we will now explain some of the social algorithms in greater details.

## Algorithms and Optimization

In order to demonstrate the role of social algorithms in solving optimization problems, let us first briefly outline the essence of an algorithm and the general formulation of an optimization problem.

### Essence of an Algorithm

An algorithm is a computational, iterative procedure. For example, Newton’s method for finding the roots of a polynomial  $p(x) = 0$  can be written as

$$x_{t+1} = x_t - \frac{p(x_t)}{p'(x_t)}, \quad (1)$$

where  $x_t$  is the approximation at iteration  $t$ , and  $p'(x)$  is the first derivative of  $p(x)$ . This procedure typically starts with an initial guess  $x_0$  at  $t = 0$ .

In most cases, as long as  $p' \neq 0$  and  $x_0$  is not too far away from the target solution, this algorithm can work very well. As we do not know the target solution  $x_* = \lim_{t \rightarrow \infty} x_t$  in advance, the initial guess can be an educated guess or a purely random guess. However, if the initial guess is too far way, the algorithm may never reach the final solution or simply fail. For example, for  $p(x) = x^2 + 9x - 10 = (x - 1)(x + 10)$ , we know its roots are  $x_* = 1$  and  $x_* = -10$ . We also have  $p'(x) = 2x + 9$  and

$$x_{t+1} = x_t - \frac{(x_t^2 + 9x_t - 10)}{2x_t + 9}. \quad (2)$$

If we start from  $x_0 = 10$ , we can easily reach  $x_* = 1$  in less than five iterations. If we use  $x_0 = 100$ , it may take about eight iterations, depending on the accuracy we want. If we start any value  $x_0 > 0$ , we can only reach  $x_* = 1$  and we will never reach the other root  $x_* = -10$ . If we start with  $x_0 = -5$ , we can reach  $x_* = -10$  in about seven steps with an accuracy of  $10^{-9}$ .

However, if we start with  $x_0 = -4.5$ , the algorithm will simply fail because  $p'(x_0) = 2x_0 + 9 = 0$ .

This has clearly demonstrated that the final solution will usually depend on where the initial solution is.

This method can be modified to solve optimization problems. For example, for a single objective function  $f(x)$ , the minimal and maximal values should occur at stationary points  $f'(x) = 0$ , which becomes a root-finding problem for  $f'(x)$ . Thus, the maximum or minimum of  $f(x)$  can be found by modifying Newton's method as the following iterative formula:

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}. \quad (3)$$

For a  $D$ -dimensional problem with an objective  $f(\mathbf{x})$  with independent variables  $\mathbf{x} = (x_1, x_2, \dots, x_D)$ , the above iteration formula can be generalized to a vector form

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \frac{\nabla f(\mathbf{x}^t)}{\nabla^2 f(\mathbf{x}^t)}, \quad (4)$$

where we have used the notation convention  $\mathbf{x}^t$  to denote the current solution vector at iteration  $t$  (not to be confused with an exponent).

In general, an algorithm  $A$  can be written as

$$\mathbf{x}^{t+1} = A(\mathbf{x}^t, \mathbf{x}_*, p_1, \dots, p_K), \quad (5)$$

which represents that fact that the new solution vector is a function of the existing solution vector  $\mathbf{x}^t$ , some historical best solution  $\mathbf{x}_*$  during the iteration history, and a set of algorithm-dependent parameters  $p_1, p_2, \dots, p_K$ . The exact function forms will depend on the algorithm, and different algorithms are only different in terms of the function form, number of parameters, and the ways of using historical data.

## Optimization

In general, an optimization problem can be formulated in a  $D$ -dimensional design space as

$$\text{minimize } f(\mathbf{x}), \quad \mathbf{x} = (x_1, x_2, \dots, x_D) \in \mathbb{R}^D, \quad (6)$$

subject to

$$h_i(\mathbf{x}) = 0, (i = 1, 2, \dots, M), g_j(\mathbf{x}) \leq 0, \quad (j = 1, 2, \dots, N),$$

where  $h_i$  and  $g_j$  are the equality constraints and inequality constraints, respectively. In a special case when the problem functions  $f(\mathbf{x})$ ,  $h_i(\mathbf{x})$ , and  $g_j(\mathbf{x})$  are all linear, the problem becomes linear programming, which can be solved efficiently by using George Dantzig's Simplex Method. However, in most cases, the problem functions  $f(\mathbf{x})$ ,  $h_i(\mathbf{x})$ , and  $g_j(\mathbf{x})$  are all nonlinear, and such nonlinear optimization problems can be challenging to solve. There are a wide class of optimization techniques, including linear programming, quadratic programming, convex optimization, interior-point method, trust-region method, conjugate-gradient methods (Süli and Mayer 2003; Yang 2010c) as well as evolutionary algorithms (Goldberg 1989), heuristics (Judea 1984), and metaheuristics (Yang 2008, 2014b).

An interesting way of looking at algorithms and optimization is to consider an algorithm system as a complex, self-organized system (Ashby 1962; Keller 2009), but nowadays researchers tend to look at algorithms from the point of view of swarm intelligence (Kennedy et al. 2001; Engelbrecht 2005; Fisher 2009; Yang 2014b).

## Traditional Algorithms or Social Algorithms?

As there are many traditional optimization techniques, a natural question is why we need new algorithms such as social algorithms? One may wonder what is wrong with traditional algorithms? A short answer is that there is nothing wrong. Extensive literature and studies have demonstrated that traditional algorithms work quite well for many different types of problems, but they do have some serious drawbacks:

- Traditional algorithms are mostly local search, there is no guarantee for global optimality for most optimization problems, except for linear programming and convex optimization. Consequently, the final solution will often depend

on the initial starting points (except for linear programming and convex optimization).

- Traditional algorithms tend to be problem-specific because they usually use some information such as derivatives about the local objective landscape. They cannot solve highly nonlinear, multimodal problems effectively, and they struggle to cope with problems with discontinuity, especially when gradients are needed.
- These algorithms are largely deterministic, and thus the exploitation ability is high, but their exploration ability and diversity of solutions are low.

Social algorithms, in contrast, attempts to avoid these disadvantages by using a population-based approach with nondeterministic or stochastic components to enhance their exploration ability. Compared with traditional algorithms, metaheuristic social algorithm are mainly designed for global search and tend to have the following advantages and characteristics:

- Almost all social algorithms are global optimizers, it is more likely to find the true global optimality. They are usually gradient-free methods and they do not use any derivative information, and thus can deal with highly nonlinear problems and problems with discontinuities.
- They often treat problems as a black box without specific knowledge, thus they can solve a wider range of problems.
- Stochastic components in such algorithms can increase the exploration ability and also enable the algorithms to escape any local modes (thus avoiding being trapped locally). The final solutions tend to “forget” the starting points, and thus independent of any initial guess and incomplete knowledge of the problem under consideration.

Though with obvious advantages, social algorithms do have some disadvantages. For example, the computational efforts of these algorithms tend to be higher than those for traditional algorithms because more iterations are needed. Due to the stochastic nature, the final solutions obtained by such algorithms cannot be repeated exactly, and multiple runs should be carried out to ensure consistency and some meaningful statistical analysis.

## Social Algorithms

The literature of social algorithms and swarm intelligence is expanding rapidly, here we will introduce some of the most recent and widely used social algorithms.

### Ant Colony Optimization

Ants are social insects that live together in well-organized colonies with a population size ranging from about 2 million to 25 million. Ants communicate with each other and interact with their environment in a swarm using local rules and scent chemicals or pheromone. There is no centralized control. Such a complex system with local interactions can self-organize with emerging behavior, leading to some form of social intelligence.

Based on these characteristics, the ant colony optimization (ACO) was developed by Marco Dorigo in 1992 (Dorigo 1992), and ACO attempts to mimic the foraging behavior of social ants in a colony. Pheromone is deposited by each agent, and such chemical will also evaporate. The model for pheromone deposition and evaporation may vary slightly, depend on the variants of ACO. However, in most cases, incremental deposition and exponential decay are used in the literature.

From the implementation point of view, for example, a solution in a network optimization problem can be a path or route. Ants will explore the network paths and deposit pheromone when it moves. The quality of a solution is related to the pheromone concentration on the path. At the same time, pheromone will evaporate as (pseudo)time increases. At a junction with multiple routes, the probability of choosing a particular route is determined by a decision criterion, depending on the normalized concentration of the route, and relative fitness of this route, comparing with all others. For example, in most studies, the probability  $p_{ij}$  of choose a route from node  $i$  to node  $j$  can be calculated by

$$p_{ij} = \frac{\phi_{ij}^\alpha d_{ij}^\beta}{\sum_{i,j}^n \phi_{ij}^\alpha d_{ij}^\beta}, \quad (8)$$

where  $\alpha, \beta > 0$  are the so-called influence parameters, and  $\phi_{ij}$  is the pheromone concentration on

the route between  $i$  and  $j$ . In addition,  $d_{ij}$  is the desirability of the route (for example, the distance of the overall path). In the simplest case when  $\alpha = \beta = 1$ , the choice probability is simply proportional to the pheromone concentration.

It is worth pointing out that ACO is a mixed of procedure and some simple equations such as pheromone deposition and evaporation as well as the path selection probability. ACO has been applied to many applications from scheduling to routing problems (Dorigo 1992).

### Particle Swarm Optimization

Many swarms in nature such as fish and birds can have higher-level behavior, but they all obey simple rules. For example, a swarm of birds such as starlings simply follow three basic rules: each bird flies according to the flight velocities of their neighbor birds (usually about seven adjacent birds), and birds on the edge of the swarm tend to fly into the center of the swarm (so as to avoid being eaten by potential predators such as eagles). In addition, birds tend to fly to search for food or shelters, thus a short memory is used. Based on such swarming characteristics, particle swarm optimization (PSO) was developed by Kennedy and Eberhart in 1995, which uses equations to simulate the swarming characteristics of birds and fish (Kennedy and Eberhart 1995).

For the ease of discussions below, let us use  $\mathbf{x}_i$  and  $\mathbf{v}_i$  to denote the position (solution) and velocity, respectively, of a particle or agent  $i$ . In PSO, there are  $n$  particles as a population, thus  $i = 1, 2, \dots, n$ . There are two equations for updating positions and velocities of particles, and they can be written as follows:

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \alpha \epsilon_1 [\mathbf{g}^* - \mathbf{x}_i^t] + \beta \epsilon_2 [\mathbf{x}_i^* - \mathbf{x}_i^t], \quad (9)$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}, \quad (10)$$

where  $\epsilon_1$  and  $\epsilon_2$  are two uniformly distributed random numbers in  $[0, 1]$ . The learning parameters  $\alpha$  and  $\beta$  are usually in the range of  $[0, 2]$ . In the above equations,  $\mathbf{g}^*$  is the best solution found so far by all the particles in the population, and each

particle has an individual best solution  $\mathbf{x}_i^*$  by itself during the entire past iteration history.

It is clearly seen that the above algorithmic equations are linear in the sense that both equation only depends on  $\mathbf{x}_i$  and  $\mathbf{v}_i$  linearly. PSO has been applied in many applications, and it has been extended to solve multiobjective optimization problems (Kennedy et al. 2001; Engelbrecht 2005). However, there are some drawbacks because PSO can often have so-called premature convergence when the population loses diversity and thus gets stuck locally. Consequently, there are more than 20 different variants to try to remedy this with various degrees of improvements.

### Bees-Inspired Algorithms

Bees such as honeybees live a colony and there are many subspecies of bees. Honeybees have three castes, including worker bees, queens, and drones. The division of labor among bees is interesting, and worker bees forage, clean hive, and defend the colony, and they have to collect and store honey. Honeybees communicate by pheromone and “waggle dance” and other local interactions, depending on species. Based on the foraging and social interactions of honeybees, researchers have developed various forms and variants of bees-inspired algorithms.

The first use of bees-inspired algorithms was probably by Nakrani and Tovey in 2004 to study web-hosting servers (Nakrani and Tovey 2004), while slightly later in 2004 and early 2005, Yang used the virtual bee algorithm to solve optimization problems (Yang 2005). At around the same time, Karaboga used the artificial bee colony (ABC) algorithm to carry out numerical optimization (Karaboga 2005). In addition, Pham et al. (2005) used bees algorithm to solve continuous optimization and function optimization problems. In about 2007, Afshar et al. (2007) used a honeybee mating optimization approach for optimizing reservoir operations.

For example, in ABC, the bees are divided into three groups: forager bees, onlooker bees, and scouts. For each food source, there is one forager bee who shares information with onlooker bees after returning to the colony from foraging, and the number of forager bees is equal to the number of food sources. Scout bees do random flight to

explore, while a forager at a scarce food source may have to be forced to become a scout bee. The generation of a new solution  $v_{i,k}$  is done by

$$v_{i,k} = x_{i,k} + \phi(x_{i,k} - x_{j,k}), \quad (11)$$

which is updated for each dimension  $k = 1, 2, \dots, D$  for different solutions (e.g.,  $i$  and  $j$ ) in a population of  $n$  bees ( $i, j = 1, 2, \dots, n$ ). Here,  $\phi$  is a random number in  $[-1, 1]$ . A food source is chosen by a roulette-based probability criterion, while a scout bee uses a Monte Carlo style randomization between the lower bound (L) and the upper bound (U).

$$x_{i,k} = L_k + r(U_k - L_k), \quad (12)$$

where  $k = 1, 2, \dots, D$ , and  $r$  is a uniformly distributed random number in  $[0, 1]$ .

Bees-inspired algorithms have been applied in many applications with diverse characteristics and variants (Pham et al. 2005; Karaboga 2005).

### Bat Algorithm

Bats are the only mammals with wings, and it is estimated that there are about 1000 different bat species. Their sizes can range from tiny bumblebee bats to giant bats. Most bat species use echolocation to a certain degree, though microbats extensively use echolocation for foraging and navigation. Microbats emit a series of loud, ultrasonic sound pulses and listen their echoes to “see” their surrounding. The pulse properties vary and correlate with their hunting strategies. Depending on the species, pulse emission rates will increase when homing for prey with frequency-modulated short pulses (thus varying wavelengths to increase the detection resolution). Each pulse may last about 5–20 milliseconds with a frequency range of 25–150 kHz, and the spatial resolution can be as small as a few millimetres, comparable to the size of insects they hunt.

Bat algorithm (BA), developed by Xin-She Yang in 2010, uses some characteristics of frequency-tuning and echolocation of microbats (Yang 2010b, Yang 2011). It also uses the

variations of pulse emission rate  $r$  and loudness  $A$  to control exploration and exploitation. In the bat algorithm, main algorithmic equations for position  $\mathbf{x}_i$  and velocity  $\mathbf{v}_i$  for bat  $i$  are

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \quad (13)$$

$$\mathbf{v}_i^t = \mathbf{v}_i^{t-1} + (\mathbf{x}_i^{t-1} - \mathbf{x}_*)f_i, \quad (14)$$

$$\mathbf{x}_i^t = \mathbf{x}_i^{t-1} + \mathbf{v}_i^t, \quad (15)$$

where  $\beta \in [0, 1]$  is a random vector drawn from a uniform distribution so that the frequency can vary from  $f_{\min}$  to  $f_{\max}$ . Here,  $\mathbf{x}_*$  is the current best solution found so far by all the virtual bats.

From the above equations, we can see that both equations are linear in terms of  $\mathbf{x}_i$  and  $\mathbf{v}_i$ . But, the control of exploration and exploitation is carried out by the variations of loudness  $A(t)$  from a high value to a lower value and the emission rate  $r$  from a lower value to a higher value. That is

$$A_i^{t+1} = \alpha A_i^t, \quad r_i^{t+1} = r_i^0(1 - e^{-\gamma t}), \quad (16)$$

where  $0 < \alpha < 1$  and  $\gamma > 0$  are two parameters. As a result, the actual algorithm can have a weak nonlinearity. Consequently, BA can have a faster convergence rate in comparison with PSO. BA has been extended to multiobjective optimization and hybrid versions (Yang 2011, 2014b).

### Firefly Algorithm

There are about 2000 species of fireflies and most species produce short, rhythmic flashes by bioluminescence. Each species can have different flashing patterns and rhythms, and one of the main functions of such flashing light acts as a signaling system to communicate with other fireflies. As light intensity in the night sky decreases as the distance from the flashing source increases, the range of visibility can be typically a few hundred metres, depending on weather conditions. The attractiveness of a firefly is usually linked to the brightness of its flashes and the timing accuracy of its flashing patterns.

Based on the above characteristics, Xin-She Yang developed in 2008 the firefly algorithm

(FA) (Yang 2008, 2010a). FA uses a nonlinear system by combining the exponential decay of light absorption and inverse-square law of light variation with distance. In the FA, the main algorithmic equation for the position  $\mathbf{x}_i$  (as a solution vector to a problem) is

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2} (\mathbf{x}_j^t - \mathbf{x}_i^t) + \alpha \epsilon_i^t, \quad (17)$$

where  $\alpha$  is a scaling factor controlling the step sizes of the random walks, while  $\gamma$  is a scale-dependent parameter controlling the visibility of the fireflies (and thus search modes). In addition,  $\beta_0$  is the attractiveness constant when the distance between two fireflies is zero (i.e.,  $r_{ij} = 0$ ). This system is a nonlinear system, which may lead to rich characteristics in terms of algorithmic behaviour.

Since the brightness of a firefly is associated with the objective landscape with its position as the indicator, the attractiveness of a firefly seen by others, depending on their relative positions and relative brightness. Thus, the beauty is in the eye of the beholder. Consequently, a pair comparison is needed for comparing all fireflies. The main steps of FA can be summarized as the pseudocode in Algorithm 1.

It is worth pointing out that  $\alpha$  is a parameter controlling the strength of the randomness or perturbations in FA. The randomness should be gradually reduced to speed up the overall convergence. Therefore, we can use

$$\alpha = \alpha_0 \delta^t, \quad (18)$$

where  $\alpha_0$  is the initial value and  $0 < \delta < 1$  is a reduction factor. In most cases, we can use  $\delta = 0.9\text{--}0.99$ , depending on the type of problems and the desired quality of solutions.

If we look at Eq. 17 closely, we can see that  $\gamma$  is an important scaling parameter. At one extreme, we can set  $\gamma = 0$ , which means that there is no exponential decay and thus the visibility is very high (all fireflies can see each other). At the other extreme, when  $\gamma \gg 1$ , then the visibility range is very short. Fireflies are essentially flying in a dense fog and they cannot see each other. Thus, each firefly flies independently and randomly.

Therefore, a good value of  $\gamma$  should be linked to the scale or limits of the design variables so that the fireflies within a range are visible to each other. This range is determined by

$$L = \frac{1}{\sqrt{\gamma}}, \quad (19)$$

where  $L$  the typical size of the search domain or the radius of a typical mode shape in the objective landscape. If there is no prior knowledge about its possible scale, we can start with  $\gamma = 1$  for most problems.

### Algorithm 1 Firefly Algorithm

```

Initialize all the parameters  $\alpha, \beta, \gamma, n$ ;
Initialize a population of  $n$  fireflies;
Determine the light intensity/fitness at  $\mathbf{x}_i$  by  $f(\mathbf{x}_i)$ ;

while  $t < MaxGeneration$  do
    for All fireflies ( $i = 1 : n$ ) do
        for All other fireflies ( $j = 1 : n$ )
        with  $i \neq j$  (inner loop) do
            if Firefly  $j$  is better/brighter
            than  $i$  then
                Move firefly  $i$  toward  $j$  using
Eq. 17;
            end
        end
        Evaluate the new solution;
        Accept the new solution if better;
    end
    Rank and update the best solution
found;
    Update iteration counter  $t \leftarrow t + 1$ ;
    Reduce  $\alpha$  (randomness strength) by a
factor;
end
```

In fact, since FA is a nonlinear system, it has the ability to automatically subdivide the whole swarm into multiple subswarms. This is because shortdistance attraction is stronger than long-distance attraction, and the division of swarm is related to the mean range of attractiveness variations. After division into multiswarms, each subswarm can potentially swarm around a local

mode. Consequently, FA is naturally suitable for multimodal optimization problems. Furthermore, there is no explicit use of the best solution  $\mathbf{g}^*$ , thus selection is through the comparison of relative brightness according to the rule of ‘beauty is in the eye of the beholder’.

It is worth pointing out that FA has some significant differences from PSO. Firstly, FA is nonlinear, while PSO is linear. Secondly, FA has an ability of multiswarming, while PSO cannot. Thirdly, PSO uses velocities (and thus have some drawbacks), while FA does not use velocities. Finally, FA has some scaling control by using  $\gamma$ , while PSO has no scaling control. All these differences enable FA to search the design spaces more effectively for multimodal objective landscapes.

FA has been applied to a diverse range of applications and has been extended to multi-objective optimization and hybridization with other algorithms (Yang 2014a, b).

### Cuckoo Search

In the natural world, among 141 cuckoo species, 59 species engage the so-called obligate brood parasitism (Davies 2011). These cuckoo species do not build their own nests and they lay eggs in the nests of host birds such as warblers. Sometimes, host birds can spot the alien eggs laid by cuckoos and thus can get rid of the eggs or abandon the nest by flying away to build a new nest in a new location so as to reduce the possibility of raising an alien cuckoo chick. The eggs of cuckoos can be sufficiently similar to eggs of host birds in terms the size, color, and texture so as to increase the survival probability of cuckoo eggs. In reality, about 1/5 to 1/4 of eggs laid by cuckoos will be discovered and abandoned by hosts. In fact, there is an arms race between cuckoo species and host species, forming an interesting cuckoo-host species coevolution system.

Based the above characteristics, Yang and Deb developed in 2009 the cuckoo search (CS) algorithm (Yang and Deb 2009). CS uses a combination of both local and global search capabilities, controlled by a discovery probability  $p_a$ . There are two algorithmic equations in CS, and one equation is

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha s \otimes H(p_a - \epsilon) \otimes (\mathbf{x}_j^t - \mathbf{x}_k^t), \quad (20)$$

where  $\mathbf{x}_j^t$  and  $\mathbf{x}_k^t$  are two different solutions selected randomly by random permutation,  $H(u)$  is a Heaviside function,  $\epsilon$  is a random number drawn from a uniform distribution, and  $s$  is the step size. This step is primarily local, though it can become global search if  $s$  is large enough. However, the main global search mechanism is realized by the other equation with Lévy flights:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha L(s, \lambda), \quad (21)$$

where the Lévy flights are simulated (or drawn random numbers) by drawing random numbers from a Lévy distribution

$$L(s, \lambda) \sim \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, \quad (s \gg 0). \quad (22)$$

Here  $\alpha > 0$  is the step size scaling factor.

By looking at the equations in CS carefully, we can clearly see that CS is a nonlinear system due to the Heaviside function, discovery probability, and Lévy flights. There is no explicit use of global best  $\mathbf{g}^*$ , but selection of the best solutions is by ranking and elitism where the current best is passed onto the next generation. In addition, the use of Lévy flights can enhance the search capability because a fraction of steps generated by Lévy flights are larger than those used in Gaussian. Thus, the search steps in CS are heavytailed (Pavlyukevich 2007; Reynolds and Rhodes 2009). Consequently, CS can be very effective for nonlinear optimization problems and multi-objective optimization (Yang and Deb 2010; Gandomi et al. 2013; Yang and Deb 2013; Yang 2014a; Yildiz 2013). A relatively comprehensive literature review of cuckoo search has been carried out by Yang and Deb (2014).

### Other Algorithms

As we mentioned earlier, the literature is expanding and more social algorithms are being developed by researchers, but we will not introduce more algorithms here. Instead, we will focus on summarizing the key characteristics of social algorithms and

other population-based algorithms so as to gain a deeper understanding of these algorithms.

## Algorithm Analysis and Insight

### Characteristics of Social Algorithms

Though different social algorithms have different characteristics and inspiration from nature, they do share some common features. Now let us look at these algorithms in terms of their basic steps, search characteristics and algorithm dynamics.

- All social algorithms use a population of multiple agents (e.g., particles, ants, bats, cuckoos, fireflies, bees, etc.), each agent corresponds to a solution vector. Among the population, there is often the best solution  $\mathbf{g}^*$  in terms of objective fitness. Different solutions in a population represent both diversity and different fitness.
- The evolution of the population is often achieved by some operators (e.g., mutation by a vector or by randomization), often in terms of some algorithmic formulas or equations. Such evolution is typically iterative, leading to evolution of solutions with different properties. When all solutions become sufficiently similar, the system can be considered as converged.
- All algorithms try to carry out some sort of both local and global search. If the search is mainly local, it increases the probability of getting stuck locally. If the search focuses too much on global moves, it will slow down the convergence. Different algorithms may use different amount of randomization and different portion of moves for local or global search so as to balance exploitation and exploration (Blum and Roli 2001).
- Selection of the better or best solutions is carried out by the “survival of the fittest” or simply elitism so that the best solution  $\mathbf{g}^*$  is kept in the population in the next generation. Such selection essentially acts a driving force to drive the diverse population into a converged population with reduced diversity but with a more organized structure.

These basic components, characteristics, and their properties can be summarized in Table 1.

**Social Algorithms, Table 1** Characteristics and properties of social algorithms.

Components/characteristics	Role or properties
Population (multiagents)	Diversity and sampling
Randomization/perturbations	Exploration and escape local optima
Selection and elitism	Exploitation and driving force for convergence
Algorithmic equations	Iterative evolution of solutions

### No-Free-Lunch Theorems

Though there are many algorithms in the literature, different algorithms can have different advantages and disadvantages and thus some algorithms are more suitable to solve certain types of problems than others. However, it is worth pointing out that there is no single algorithm that can be most efficient to solve all types of problems as dictated by the no-free-lunch (NFL) theorems (Wolpert and Macready 1997). Even the no-free-lunch theorems hold under certain conditions, but these conditions may not be rigorously true for actual algorithms. For example, one condition for proving these theorems is the so-called no-revisiting condition. That is, the points during iterations form a path, and these points are distinct and will not be visited exactly again, though their nearby neighborhood can be revisited. This condition is not strictly valid because almost all algorithms for continuous optimization will revisit some of their points in history. Such minor violation of assumptions can potentially leave room for free lunches. It has also been shown that under the right conditions such as coevolution, certain algorithms can be more effective (Wolpert and Macready 2005).

In addition, a comprehensive review by T. Joyce and J.M. Herrmann on no-free-lunch (NFL) theorems (Joyce and Herrmann 2018), free lunches may exist for a finite set of problems, especially those algorithms that can exploit the objective landscape structure and knowledge of optimization problems to be solved. If the performance is not averaged over all possible problems, then free lunches can exist. In fact, for a given

finite set of problems and a finite set of algorithms, the comparison is essentially equivalent to a zero-sum ranking problem. In this case, some algorithms can perform better than others for solving a *certain type* of problems. In fact, almost all research papers published about comparison of algorithms use a few algorithms and a finite set (usually under 100 benchmarks), such comparisons are essentially ranking. However, it is worth pointing out that for a finite set of benchmarks, the conclusions (e.g., ranking) obtained can only apply for that set of benchmarks, they may not be valid for other sets of benchmarks and the conclusions can be significantly different. If interpreted in this sense, such comparison studies and their conclusions are consistent with NFL theorems.

## Future Directions

The research area of social algorithms is very active, and there are many hot topics for further research directions concerning these algorithms. Here we highlight a few:

- **Theoretical framework:** Though there are many studies concerning the implementations and applications of social algorithms, mathematical analysis of such algorithms lag behind. There is a strong need to build a theoretical framework to analyze these algorithms mathematically so as to gain indepth understanding (He et al. 2017). For example, it is not clear how local rules can lead to the rise of self-organized structure in algorithms. More generally, it still lacks key understanding about the rise of social intelligence and swarm intelligence in a multi-agent system and their exact conditions.
- **Parameter tuning:** Almost all algorithms have algorithm-dependent parameters, and the performance of an algorithm is largely influenced by its parameter setting. Some parameters may have stronger influence than others (Eiben and Smit 2011). Therefore, proper parameter tuning and sensitivity analysis are needed to tune algorithms to their best. However, such tuning is largely done by trial and

error in combination with the empirical observations. How to tune them quickly and automatically is still an open question.

- **Large-scale applications:** Despite the success of social algorithms and their diverse applications, most studies in the literature have concerned problems of moderate size with the number of variables up to a few hundred at most. In real-world applications, there may be thousands and even millions of design variables, it is not clear yet how these algorithms can be scaled up to solve such large-scale problems. In addition, though social algorithms have been applied to solve combinatorial problems such as scheduling and the travelling salesman problem with promising results, these problems are typically nondeterministic polynomial-time (NP) hard, and thus for larger problem sizes, they can be very challenging to solve. Researchers are not sure how to modify exist social algorithms to cope with such challenges.
- **Hybrid and coevolutionary algorithms:** The algorithms we have covered here are algorithms that are “pure” and “standard” in the sense that they have not been heavily modified by hybridizing with others. Both empirical observations and studies show that the combination of the advantages from two or more different algorithms can produce a better hybrid, which can use the distinct advantages of its component algorithms and potentially avoid their drawbacks. In addition, it is possible to build a proper algorithm system to allow a few algorithms to coevolve to obtain an overall better performance.

Though NFL theorems may hold for simple algorithms, it has been shown that there can be free lunches for coevolutionary algorithms (Wolpert and Macready 2005). Therefore, future research can focus on figuring out how to assemble different algorithms into an efficient coevolutionary system and then tune the system to its best.

- **Self-adaptive and self-evolving algorithms:** Sometimes, the parameters in an algorithm can vary to suit for different types of problems. In

addition to parameter tuning, such parameter adaptivity can be advantageous. There are some basic form of adaptive algorithms in the literature and they mainly use random variations of parameters in a fixed range. Ideally, an algorithm should be self-adaptive and be able to automatically tune itself to suit for a given type of problems without much supervision from the users (Yang et al. 2013). Such algorithms should also be able to evolve by learning from their past performance histories. The ultimate aim for researchers is to build a set of self-adaptive, self-tuning, self-learning, and self-evolving algorithms that can solve a diverse range of real-world applications efficiently and quickly in practice.

Social algorithms have become a powerful tool set for solving optimization problems and their studies form an active area of research. It is hoped that this work can inspire more research concerning the above important topics.

## Bibliography

### Primary Literature

- Afshar A, Haddad OB, Marino MA, Adams BJ (2007) Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation. *J Franklin Inst* 344(4):452–462
- Ashby WA (1962) Principles of the self-organizing system. In: Von Foerster H, Zopf GW Jr (eds) Principles of self-organization: transactions of the University of Illinois Symposium. Pergamon Press, London, pp 255–278
- Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Comput Surv* 35(2):268–308
- Chabert JL (1999) A history of algorithms: from the pebble to the microchip. Springer, Heidelberg
- Dorigo M (1992) Optimization, learning and natural algorithms. PhD thesis, Politecnico di Milano, Italy
- Eiben AE, Smit SK (2011) Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm Evol Comput* 1(1):19–31
- Engelbrecht AP (2005) Fundamentals of computational swarm intelligence. Wiley, Hoboken
- Fisher L (2009) The perfect swarm: the science of complexity in everyday life. Basic Books, New York
- Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley, New York
- Gandomi AH, Yang XS, Alavi AH (2013) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Eng Comput* 29(1):17–35
- Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: harmony search. *SIMULATION* 76(2):60–68
- Glover F (1986) Future paths for integer programming and links to artificial intelligence. *Comput Oper Res* 13(5): 533–549
- Goldberg DE (1989) Genetic algorithms in search, optimisation and machine learning. Addison Wesley, Reading
- He XS, Yang XS, Karamanoglu M, Zhao YX (2017) Global convergence analysis of the flower pollination algorithm: a discrete-time Markov chain approach. *Proc Comput Sci* 108(1):1354–1363
- Holland J (1975) Adaptation in natural and Artificial systems. University of Michigan Press, Ann Arbor
- Joyce T, Herrmann JM (2018) A review of no free lunch theorems, and their implications for metaheuristic optimisation. In: Yang XS (ed) Nature-inspired algorithms and applied optimization. Springer, Cham, Switzerland, pp 27–52
- Judea P (1984) Heuristics. Addison-Wesley, New York
- Karaboga D (2005) An idea based on honeybee swarm for numerical optimization, Technical Report. Erciyes University, Turkey
- Keller EF (2009) Organisms, machines, and thunderstorms: a history of self-organization, part two. Complexity, emergence, and stable attractors. *Hist Stud Nat Sci* 39(1):1–31
- Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, Piscataway, pp 1942–1948
- Kennedy J, Eberhart RC, Shi Y (2001) Swarm intelligence. Academic Press, London
- Kirkpatrick S, Gellat CD, Vecchi MP (1983) Optimization by simulated annealing. *Science* 220(4598):671–680
- Lazer D (2015) The rise of the social algorithm. *Science* 348(6239):1090–1091
- Nakrani S, Tovey C (2004) On honeybees and dynamic server allocation in internet hosting centers. *Adapt Behav* 12(3):223–240
- Pavlyukevich I (2007) Lévy flights, non-local search and simulated annealing. *J Comput Phys* 226(2): 1830–1844
- Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S, Zaidi M (2005) The bees algorithm, technical note. Manufacturing Engineering Centre, Cardiff University, Cardiff
- Rashedi E, Nezamabadi-pour H, Sazaydi S (2009) GSA: a gravitational search algorithm. *Inf Sci* 179(13): 2232–2248
- Reynolds AM, Rhodes CJ (2009) The Lévy flight paradigm: random search patterns and mechanisms. *Ecology* 90(4):877–887
- Rodrigues D, Silva GFA, Papaá JP, Marana AN, Yang XS (2016) EEG-based person identification through binary flower pollination algorithm. *Expert Syst Appl* 62(1): 81–90

- Storn R, Price K (1997) Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces. *J Glob Optim* 11(4):341–359
- Süli E, Mayer D (2003) An introduction to numerical analysis. Cambridge University Press, Cambridge
- Turing AM (1948) Intelligent machinery, National Physical Laboratory, Technical report
- Wolpert DH, Macready WG (1997) No free lunch theorem for optimization. *IEEE Trans Evol Comput* 1(1):67–82
- Wolpert DH, Macready WG (2005) Coevolutionary free lunches. *IEEE Trans Evol Comput* 9(6):721–735
- Yang XS (2005). Engineering optimizaton via nature-inspired virtual bee algorithms. In: Artificial intelligence and knowledge engineering application: a bioinspired approach, proceedings of IWINAC, pp 317–323
- Yang XS (2008) Nature-inspired metaheuristic algorithms. Luniver Press, Bristol
- Yang XS (2010a) Firefly algorithm, stochastic test functions and design optimisation. *Int J Bio-Inspired Comput* 2(2):78–84
- Yang XS (2010b) A new metaheuristic bat-inspired algorithm. In: Nature-inspired cooperative strategies for optimization (NICSO 2010). Springer, pp 65–74. SCI 284
- Yang XS (2010c) Engineering optimization: an introduction with metaheuristic applications. Wiley, Hoboken
- Yang XS (2011) Bat algorithm for multi-objective optimisation. *Int J Bio-Inspired Comput* 3(5):267–274
- Yang XS, (2012). Flower pollination algorithm for global optimization. In: Unconventional computation and natural computation. Lecture notes in computer science, vol 7445, pp 240–249
- Yang XS (2014a) Cuckoo search and firefly algorithm: theory and applications. Studies in computational intelligence, vol 516. Springer, Heidelberg
- Yang XS (2014b) Nature-inspired optimization algorithms. Elsevier Insight, London
- Yang XS (2018) Nature-inspired algorithms and applied Optimizaton. Springer, Cham, Switzerland. (in press)
- Yang XS, Deb S (2009) Cuckoo search via Lévy flights. In: Proceedings of world congress on nature & biologically inspired computing (NaBic 2009). IEEE Publications, Coimbatore, pp 210–214
- Yang XS, Deb S (2010) Engineering optimization by cuckoo search. *Int J Math Mod Num Optim* 1(4): 330–343
- Yang XS, Deb S (2013) Multiobjective cuckoo search for design optimization. *Comput Oper Res* 40(6): 1616–1624
- Yang XS, Deb S (2014) Cuckoo search: recent advances and applications. *Neural Comput & Applic* 24(1): 169–174
- Yang XS, Papa JP (2016) Bio-inspired computation and applications in image processing. Academic Press, London
- Yang XS, Deb S, Loomes M, Karamanoglu M (2013) A framework for self-tuning optimization algorithm. *Neural Comput & Applic* 23(7–8):2051–2057
- Yang XS, Karamanoglu M, He XS (2014) Flower pollination algorithm: a novel approach for multiobjective optimization. *Eng Optim* 46(9):1222–1237
- Yang XS, Chien SF, Ting TO (2015) Bio-inspired computation in telecommunications. Morgan Kaufmann, Waltham
- Yildiz AR (2013) Cuckoo search algorithm for the selection of optimal machine parameters in milling operations. *Int J Adv Manuf Technol* 64(1):55–61
- ### Books and Reviews
- Allan M (1977) Darwin and his flowers. Faber & Faber, London
- Altringham JD (1998) Bats: biology and behaviour. Oxford University Press, Oxford
- Beer D (2016) The social power of algorithms. *Inf Commun Soc* 20(1):1–13
- Bekdas G, Nigdeli SM, Yang XS (2015) Sizing optimization of truss structures using flower pollination algorithm. *Appl Soft Comput* 37(1):322–331
- Bell WJ (1991) Searching behaviour: the Behavioural ecology of finding resources. Chapman & Hall, London
- Berlinski D (2001) The advent of the algorithm: the 300-year journey from an idea to the computer. Harvest Book, New York
- Bolton B (1995) A new general catalogue of the ants of the world. Harvard University Press, Cambridge, MA
- Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press, Cambridge
- Brin S, Page L (1998) The anatomy of a large-scale hyper-textual web search engine. *Comput Netw ISDN Syst* 30(1–7):107–117
- Copeland BJ (2004) The essential turing. Oxford University Press, Oxford
- Dantzig GB, Thapa MN (1997) Linear programming 1: introduction. Springer, Heidelberg
- Davies NB (2011) Cuckoo adaptations: trickery and tuning. *J Zool* 284(1):1–14
- Fishman GS (1995) Monte carlo: concepts, Algorithms and Applications. Springer, New York
- Glover BJ (2007) Understanding flowers and flowering: an integrated approach. Oxford University Press, Oxford
- Hölldobler B, Wilson EO (2009) The superorganism: the beauty, Elegance and strangeness of insect Societies. Norton & Co, New York
- Jamil M, Yang XS (2013) A literature survey of benchmark functions for global optimisation problems. *Int J Math Mod Numer Optim* 4(2):150–194
- Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA
- Lewis SM, Cratsley CK (2008) Flash signal evolution, mate choice and predation in fireflies. *Annu Rev Entomol* 53(2):293–321
- Lindauer M (1971) Communication among social bees. Harvard University Press, Cambridge, MA
- Page L, Brin S, Motwani R, Winograd T (1998) The PageRank citation ranking: bringing order to the web. Technical Report. Stanford University, Stanford, USA

- Singh S (1999) The code book. Fouth Estate, London
- Struik DJ (1987) A concise history of mathematics, 4th edn. Dover Publications, New York
- Surowiecki J (2004) The wisdom of crowds. Doubleday, Anchor
- Vapnik V (1995) The nature of statistical learning theory. Springer, New York
- Waser NM (1986) Flower constancy: definition, cause and measurement. *Am Nat* 127(5):596–603
- Yang XS (2011) Metaheuristic optimization. Scholarpedia 6(8):11472
- Yang XS, Cui ZH, Xiao RB, Gandom AH, Karamanoglu M (2013) Swarm intelligence and bio-inspired computation: theory and applications. Elsevier, London



## Cellular Ants Computing

Konstantinos Ioannidis and  
Georgios Ch. Sirakoulis  
School of Engineering, Department of Electrical  
and Computer Engineering, Democritus  
University of Thrace (DUTH), Xanthi, Greece

### Article Outline

Glossary  
Definition of the Subject  
Introduction  
Cellular Automata Fundamentals  
Ant Colony Optimization Algorithms  
Principles and Applications of Cellular Ants  
Future Directions  
Bibliography

### Glossary

**Artificial Intelligence** The study of “intelligent devices” which perceive their environment and act to maximize the possibility of their success at some goal.

**Classification** A general process related to categorization where ideas and objects are recognized, differentiated, and understood.

**Clustering** The process of partitioning a dataset into specific meaningful subsets, by categorizing or grouping similar data items together.

**Dynamic System** A system in which a function describes the time dependence of a point in a geometrical space.

**Field-Programmable Gate Array (FPGA)** An integrated circuit designed to be configured by a customer or a designer after manufacturing.

**Swarm Intelligence** The collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence.

**Traveling Salesman Problem** An NP-problem where, providing a list of nodes and their correlation, the shortest possible route is defined.

### Definition of the Subject

As the community encounters novel computing environments that offer new opportunities while posing new challenges, it is reasonable to seek inspiration from natural analogues of these environments. Following this research trend, much attention was given to bio-inspired techniques, which have been proven capable of successfully handling complex algorithmic problems. As such, Ant Colony Optimization (ACO) algorithms have been introduced as a meta-heuristic optimization technique originating from the swarm intelligence research field. In addition, many decades ago, Cellular Automata (CA) have been proposed as a powerful parallel computational tool aiming at modeling biological systems. Exploiting a CA as a basis and introducing the fundamental principles of ACO, an unconventional computational model results taking advantage of their common prominent features, such as simplicity, locality, and self-organization. Cellular ants have been proposed to overcome many computational and algorithmic issues providing efficient solutions in the fields of clustering and swarm robotics among others.

### Introduction

CA is a discrete, spatially explicit extended dynamic system characterized by a simple structure. It is composed of individual elements, called cells, which evolve in discrete time steps according to a common local transition rules. CA were originally introduced by John von Neumann (1966) and his colleague Stanislaw Ulam (1952).

CA have sufficient expressive dynamics to represent phenomena of arbitrary complexity,

while they can be simulated exactly by digital computers, due to their intrinsic discreteness, i.e., the topology of the simulated object is reproduced in the simulating device (Vichniac 1984). The CA approach is consistent with the modern notion of unified space-time. In computer science, space corresponds to memory and time to the processing unit. In CA, memory (CA cell state) and processing unit (CA local rule) are strictly related to a CA cell (Sirakoulis et al. 2003). Moreover, CA are an alternative to partial differential equations (Omohundro 1984; Toffoli 1984), and they can efficiently process complex boundary and initial conditions, inhomogeneities, and anisotropies (Sirakoulis et al. 2000). Algorithms based on CA exploit their inherit parallelism to execute their processes faster on digital computers and could be appropriate for implementation on dedicated massively parallel computers, such as the Cellular Automaton Machine (CAM) (Toffoli and Margolus 1987).

On the other hand, artificial ACO was based on the behavior of ants to solve initially the shortest path problem when seeking food without the use of visual information (Dorigo et al. 1996). To exchange information about which path should be followed, ants communicate with each other using pheromones, which are unique to ants. Pheromones are detected by moving ants to drive their motion; nonetheless, they display a specific evaporation rate. When the population of ants following a specific path is increased, the amount of the deposited pheromones is also increased, resulting in a higher probability for other ants to follow this route.

Essentially, the ACO algorithms are a colony of artificial ants or cooperative agents, designed to solve combinatorial optimization problems. These algorithms are probabilistic in nature since local minima entrapment could be avoided, and, thus, they can provide efficient solutions similar to the corresponding natural solutions. ACO algorithms are extensively utilized in a variety of applications such as the travel salesman problem (Dorigo et al. 1996), image retrieval (Konstantinidis et al. 2009),

classification (Martens et al. 2007), communication networks (Di Caro and Dorigo 1998), etc.

The combination of these two nature inspired computation techniques advantages of their common prominent features, like simplicity, locality, and self-organization. The cellular model for parallel ACO comprises a generic proposal following the cellular model for parallel evolutionary algorithms. In addition, it displays major alterations compared to similar cellular-like models such as hardware parallel ACOs using processing arrays (Merkle and Middendorf 2002) or FPGAs (Scheuermann et al. 2004). This differentiation exists mainly due to the application of a cellular automata model (Albuquerque and Dupuis 2002) but without introducing a parallel implementation. In addition, various alterations of such combinations have also been reported like defining CA transition rules using ACO algorithms (Yang et al. 2012), adapting CA locality for stigmergy (Li et al. 2011), etc. Most of these approaches combine both AI methods in an opposite approach (adopt CA features in ACO approaches) leading to sequential processes forfeiting the inherit parallelism.

Cellular ants aim to provide a framework of solutions, which incorporate the benefits of both artificial intelligence methods, on one hand, the inherit parallelism and simplicity of the CA and, on the other hand, the effectiveness of ACO in computing optimized solutions. Such models where a single colony is structured in small neighborhoods with limited interactions are in line with the most widely accepted categories of parallel evolutionary algorithms (Cantu-Paz 2000; Alba and Tomassini 2002). Current applications that benefit from cellular ants display a wide diversity including clustering (Bitsakidis et al. 2015), robotics (Rosenberg 2008), network design (Liu et al. 2008), etc.

The rest of the chapter includes the CA and ACO fundamentals as well as details about their adaptation in specific applications. Finally, future directions will address the potential impacts of the cellular ants on the development of specific research fields.

## Cellular Automata Fundamentals

CA can be considered as dynamical systems in which space and time are discrete and interactions are local. In general, a CA is divided into a number of identical entities, which display local connectivity arranged on a regular array. A finite CA is usually defined by the quadruple  $A = (S_d, Q, N, \delta)$  where:

- $A$  represents the CA.
- $S_d$  denotes the space where cells are distributed.
- $Q$  represents the set of states based on which every cell is marked.
- $N$  defines the utilized type of neighborhood  $\forall z \in S_d, N(z) \subseteq S_d$ .
- $\delta$  indicates a rule of states transition which conducts the cells to transit from one state to another,  $\forall z \in S_d, \delta(z) : Q^{|N(z)|} \mapsto Q$ .

Cellular space can be either a 1-D or a 2-D grid  $G$ . Let  $G = [0, \dots, w(n)-1] \times [0, \dots, h(n)-1]$  represent the cellular space which is a two-dimensional array of all positions  $(x, y)$ , where  $w(n)$  and  $h(n)$  are functions related to the agent number  $n$ . For CAs of a 2-D dimensionality, two types of neighborhoods are usually considered, von Neumann and Moore neighborhood, and can be expressed, respectively, as follows:

$$N_{(xi, yi)}^{VN} = \{(x, y) : |x - xi| + |y - yi| \leq r\} \quad (1)$$

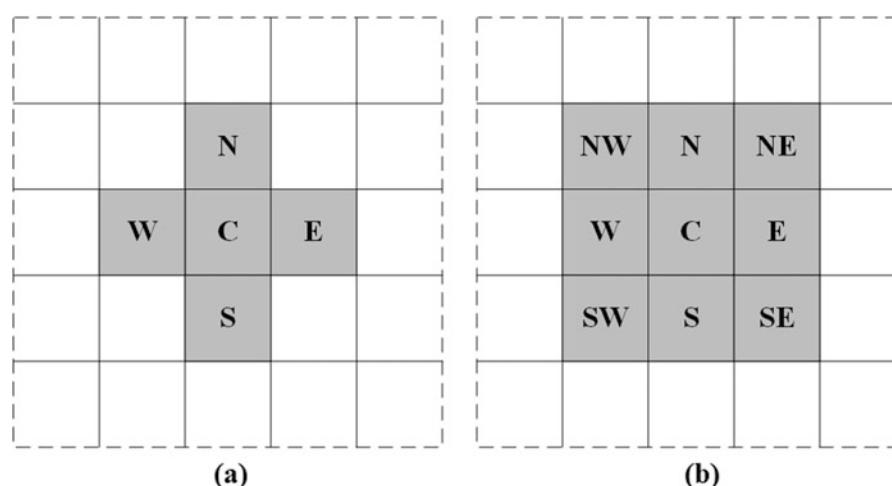
$$N_{(xi, yi)}^M = \{(x, y) : |x - xi| \leq r, |y - yi| \leq r\} \quad (2)$$

where  $x_i$  and  $y_i$  correspond to the coordinates in the lattice of a given cell and  $r$  the range of the applied neighborhood. A pictorial representation of both types of range equal to one is provided in Fig. 1.

The transition set of rules  $\delta$  determines how every cell updates its state at each evolution step. The state of every cell is affected only by its current state and the states of its adjacent cells, based on the applied transition rules. It should be highlighted that every cell of the lattice updates its state simultaneously, providing an inherent parallel system. As, for example, considering that the transition rule corresponds to a logical AND (+) and von Neumann neighborhood of range 1 is used, every cell will update its state based on the following equation:

$$\begin{aligned} C_{i,j}^{t+1} = & C_{i,j}^t + C_{i-1,j}^t + C_{i,j-1}^t + C_{i+1,j}^t \\ & + C_{i,j+1}^t \end{aligned} \quad (3)$$

where  $C$  represents a cell state,  $i$  and  $j$  the coordinates of a cell, and  $t$  the evolution step.



**Cellular Ants Computing, Fig. 1** 2-D CA neighborhoods (a) von Neumann and (b) Moore

Finally, as a finite system, boundary conditions define the way that boundary cells are evolved due to the different spatial neighborhood they display. Most commonly, it is assumed that the lattice is augmented by a set of virtual cells beyond its limits and according to the applied range of neighborhood. Periodic and null boundary conditions are the most widely used techniques to resolve the issue, meaning the lattice displays a torus-like topology or virtual cells are denoted with a null state, respectively.

## Ant Colony Optimization Algorithms

The basic element of ACO algorithms is agents with simple capabilities, which mimic the behavior of real insects, namely ants (Dorigo 1992). Real ants are considered unsophisticated insects in some ways due to their limited memory and since they exhibit individual behavior that appears to have a large random component. Nonetheless, acting as a collective, ants collaborate to achieve a variety of complicated tasks with increased reliability and consistency, such as solving the shortest path problem. This type of social behavior relies on the self-organization feature, a set of dynamical mechanisms ensuring that the global aim of the system could be achieved through low-level interactions between its elements. The interaction between these elements has as prerequisite only the availability of local information. The transfer of such information between ants can be achieved with two discrete methods, a direct communication and an indirect communication, usually known as stigmergy. The latter is biologically realized through pheromones, a special secretory chemical described by an evaporation ratio and deposited as a trail by individual ants during their motion. Due to the ability of ants to deposit and detect pheromones, they tend to follow routes displaying higher pheromone concentrations. In ACO algorithms, an ant moves from point  $i$  to point  $j$  in the configuration area with a probability equal to:

$$p_{i,j} = \frac{(\tau_{i,j}^\alpha)(\eta_{i,j}^\beta)}{\sum (\tau_{i,j}^\alpha)(\eta_{i,j}^\beta)} \quad (4)$$

where  $\tau_{i,j}^\alpha$  and  $\eta_{i,j}^\beta$  correspond to the pheromone concentration and the heuristic value associated with an available solution route, respectively. Variables  $\alpha$  and  $\beta$  are positive real parameters whose values determine the relative importance of pheromone versus heuristic information.

During their search for food, all ants deposit a small quantity of a specific pheromone type. As soon as an ant discovers a food source, it evaluates its quantity and quality and transfers a proportional amount of food to the nest. During their return, every such ant deposits on the ground a different type of pheromone of specific quantity, based on the detected food. Pheromone quantities are updated as follows:

$$\tau_{i,j}^\alpha = (1 - \rho)\tau_{i,j}^\alpha + \Delta\tau_{i,j}^\alpha \quad (5)$$

where  $\rho$  represents the evaporation rate and  $\Delta\tau_{i,j}$  is the amount of the deposited pheromone, typically calculated by:

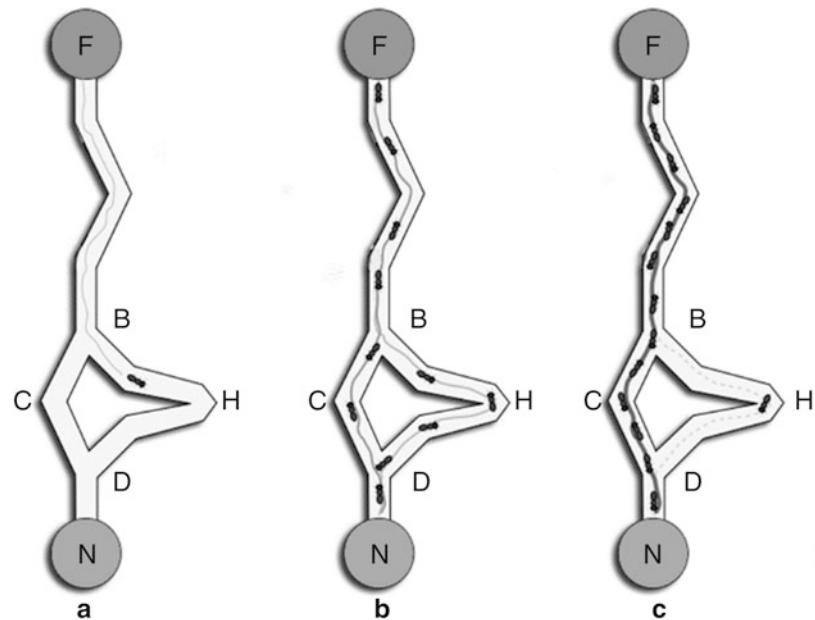
$$\Delta\tau_{i,j}^k = \begin{cases} \frac{1}{L_k} & \text{if ant } k \text{ travels on edge } i,j \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where  $L_k$  is the cost of the  $k^{\text{th}}$  tour of an ant (typically measured as length). Finally, the deposited trails will guide other ants to the food source by detecting the corresponding concentration.

For example, in Fig. 2, the ants wander from food source F to the nest N. At point B, every ant should decide which path must be followed (from either point C or point H). A higher quantity of pheromone through point C provides an ant a stronger motivation and, thus, a higher probability to follow this route. As no pheromone was deposited previously at point B, the first ant reaching point B displays the same probability to follow the path through the point C or point H. The first ant following the path BCD will reach point D earlier than the first ant that followed BHD path. Therefore, an ant returning from N to D will detect a stronger trail on path DCB, caused by the half of

**Cellular Ants Computing,**

**Fig. 2** An example of a real ant colony, (a) an ant follows BHD path, (b) ants follow both paths, and (c) the majority of the ants follow BCD path, which is the shortest route (Ioannidis et al. 2011)



all the ants that by chance followed DCBF path. Consequently, the number of ants following this path will increase during time, which results to higher pheromone concentrations on the shortest route. Thus, the probability based on which an ant selects its path is quickly biased toward the optimal route, meaning the shortest path.

### Principles and Applications of Cellular Ants

Cellular ants' novelty and their capability in providing efficient solutions in many diverse problems rely on the common features that both AI methods display. The presented intelligent algorithm can be characterized by simplicity also since both algorithms are based on simple structures, cells for the CA, and ants for the ACO algorithms. In addition, both of these structures are required to interact with their surroundings and react upon alterations. Local interactions should be valid to enable global behaviors rising from the collective effect of the simple components. As a result, cellular ants present the vital feature of self-organization since initially disordered systems can diverge to more stable states based on the local interactions. In addition, both CA and

ACO algorithms fully cover the basic pillars of self-organization. They display dynamical non-linearity behavior, balance of exploitation and exploration, and multiple interactions. Therefore, common features comprise the basis of a unified system, which can exploit the advantages of both evolutionary intelligence approaches.

Contrary to attempts of combining ACO with CA principles (Yang et al. 2012), cellular ants theory foresees the adaptation of ACO features into a CA system. This is significantly beneficial for the overall objective and the complexity since ACO features will provide efficient solutions, while the inherit parallelism of the CA will decrease the required execution time. Following the CA principles, the combined algorithm assumes that the operational area or the processed data are divided into a lattice of cells. The applied neighborhoods and the boundary conditions are similar with the corresponding CA features. In most applications, Moore neighborhood is adopted so that every cell could react more efficiently with its adjacent cells.

Eventually, cellular ants' applicability relies on the development of proper sets for cells' states and the corresponding transition rules in order to evolve during time. The novelty of every cellular ant algorithm is based on the proposed alterations

regarding these two basic features. Mostly, the developed transition set of rules designates the operations and the final objectives. Therefore, in most cases, researchers concentrate their efforts in developing an appropriate evolution rule set based on the application of the system. For reasons of comprehension, cellular ants' basic principles are provided through the presentation of applications since the transition rules strictly depend on the application itself.

### Cellular Ants in Data Clustering

The process of a clustering algorithm sets as an objective to partition datasets into specific meaningful subsets. Several data clustering approaches exist in the literature that are distinguished by algorithmic principles, e.g., data types, cluster shapes, etc. Among others, traditional ACO methods have also been reported as clustering methods (Ji et al. 2013). The cellular ant method conceptually differs from these by considering the ants as unique data objects that can wander around the search space looking for similar ants. Moreover, parallel implementations of ACO algorithms significantly differ from similar implementations of cellular ants since parallel processes are executed nonetheless, without self-organization.

One of the first attempts of applying cellular ants in data clustering was made by Chen et al. (2004). The method, entitled by the authors as “Ants Sleeping Model” (ASM), extends the classical CA model by incorporating biological swarm intelligence for solving data clustering problems in data mining. The authors selected to embody ACO main features to their model and consider that the ants tend to group with ants having similar physiques. The method inserts a “sleeping” mode to every ant as a feature and modifies the final objective of the total system. The ants repeatedly search for comfortable and secure positions in their surrounding environment. When an ant discovers such positions, it becomes inactive and terminates its exploration. In addition, when an ant at sleeping state is not satisfied with its current position, it becomes active and starts its exploration to find an additional “comfort” position. The method denotes the scattered data as cells, while the measurement of fitness of a cell ant is calculated by:

$$f(ag_i) = \max \left\{ 0, \frac{1}{(2s_x + 1) \times (2s_y + 1)} \sum_{ag_j \in N(ag_i)} \left( 1 - \frac{d(ag_i, ag_j)}{\alpha} \right) \right\} \quad (7)$$

where  $\alpha$  is a constant representing the substitution of the average distance between two agents,  $d$  denotes the measurement of distance between two agents, and  $s_x$  and  $s_y$  is the radius of the neighborhood toward both directions. Its value measures the similarity of an agent based on its environment. The probability of activating a cell ant from its sleeping mode is given by:

$$p_a(f) = \frac{\beta^\lambda}{\beta^\lambda + f^\lambda} \quad (8)$$

where  $\beta$  and  $\lambda$  denote a threshold of the ants' active fitness and their active pressure (usually a constant), respectively.

The method successfully divides the dataset into the desired clusters nonetheless; stigmergy of the classical ACO algorithms is not applied.

The aforementioned method exploits a fraction of the ACO advantages disregarding basic principles that could be proven significantly beneficial in data clustering. A more relevant method with the concept of cellular ants includes multiple additions regarding the cells' states and the corresponding transition rules in order to exploit fully their advantages (Moere and Clayden 2005). The method abolishes the term of sleeping state for every cellular ant and considers that the actions of every agent are determined by the discrete number of similar ants in their neighborhood. This assumption has impact on the final set of cell states and the transition rules. Data similarity between pairs of ants is calculated as follows:

$$\text{data}_i = (z_{i1}, z_{i2}, \dots, z_{ip}) \in R^p, p \in Z^+ \quad (9)$$

$$d_{ij} = d(\text{data}_i - \text{data}_j) = \|\text{data}_i - \text{data}_j\|_p \quad (10)$$

$$d_{ij} < t \Rightarrow \text{similar}(\text{ant}_i, \text{ant}_j) = \text{true} \quad (11)$$

where  $p$  is the dimensionality of the dataset and  $t$  is considered similar to the object distance measure variable  $a$  in normal ant-based clustering

approaches. In addition, transition rules were developed in order to achieve positional swapping so that adjacent cell ants could swap positions in the grid based on their data similarity and so clusters could be ordered internally. These swapping rules ensure that for any three ants that are linearly neighboring, the pair of cellular ants with the highest distance in parameter space will be positioned at the outer grid positions. Finally, the cell states are updated with additional states that correspond to pheromone concentrations, and the required search time for defining similar ants could be decreased through stigmergy.

The method was evaluated using the IRIS dataset and produced adequate results in terms of clustering accuracy. Nonetheless, its accuracy is decreased due to some factors relating to the approach itself and the developed transition rules. Due to the positional swapping and the similarity measures, which affect directly the transition rules, cellular ants display excessive mobility in cases where no similar adjacent ants or no pheromone are present. In addition, the concept of the discrete data tolerance faces two major issues: the diverse weight of differences between the values of two data dimensions when a wider allocation of values is displayed in one of the dimensions and defining a proper tolerance threshold. Finally, the method suffers from the so-called “trapped” cellular ants, which emerges due to faulty positioning on initial random allocation or their boundary location where their evolution is problematic (due to boundary conditions).

These flaws were thoroughly studied, and an enhanced version of the cellular ant method for data clustering was introduced (Bitsakidis et al. 2015) by mostly developing a more detailed transition rule set which considers these drawbacks. The excessive mobility that boundary cellular ants display, which also leads to non-comprehensive visualization, was addressed by applying a rectangular grid instead of a toroidal. In addition, the issues inserted by the Euclidean distance as a similarity measure were addressed by applying the proportional distance measurement where the proportion between the values of two data objects is examined. The similarity threshold should distinguish two agents according to their percentage proportional distance (Eq. 12). Considering also

the dimensionality issue, the enhanced cellular ant method imposes different predefined weights for each data dimension, and so, the proportional distance measurement embodied as a step of the transition rules is computed as:

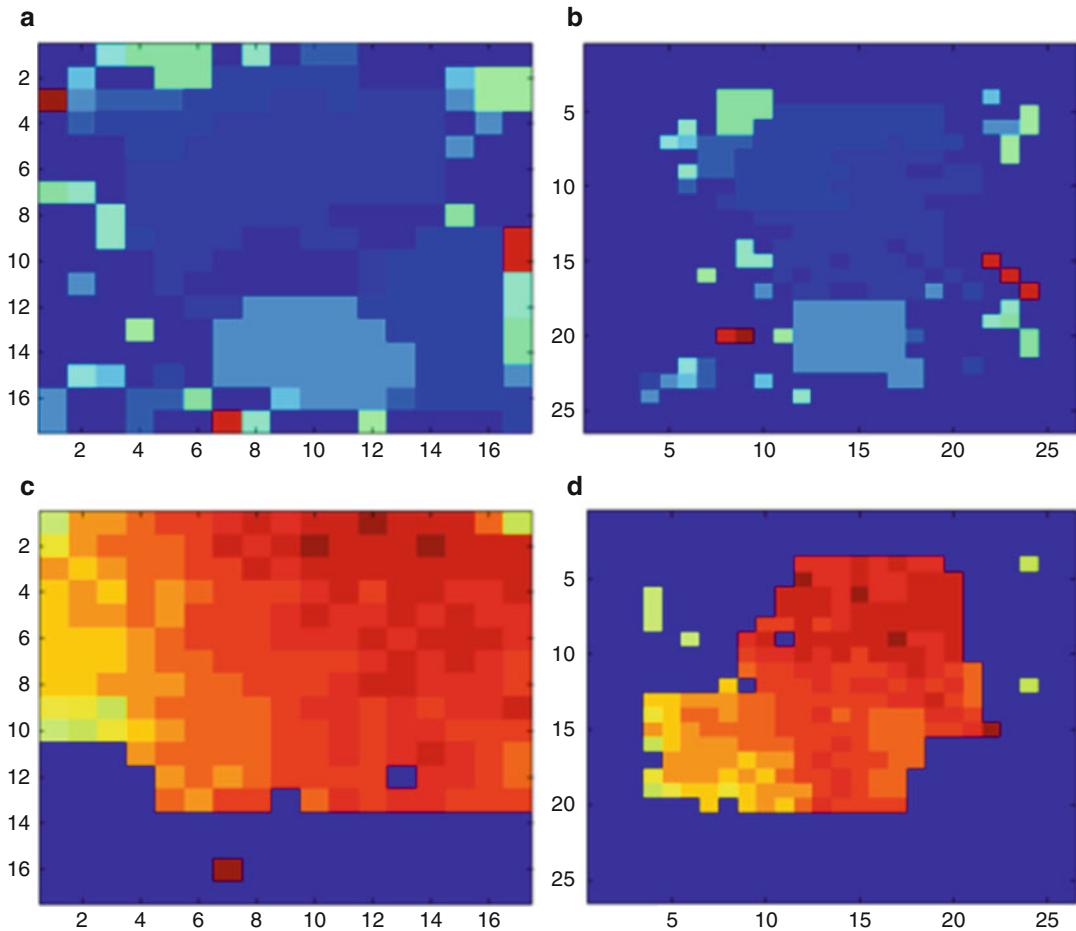
$$d_{ij} = 2 \left| \frac{100 \times \sum_{i=1}^{i=j} \frac{b_i}{a_i + b_i}}{\sum_{i=1}^{i=j} \text{weight}_i} - 50 \right| \quad (12)$$

Finally, the method inserts a new concept to solve the “trapped” cellular ants’ problem, named as Hyper-Cellular Ants. Essentially, the simplicity of the model only requires the insertion of an additional transition rule, which will handle such occasions. More specific, Hyper-Cellular Ants present two additional capabilities in order to avoid local minima entrapment: **(a)** extended visibility and interaction capability beyond their direct Moore neighborhood of radius 1 and **(b)** capability of a “flying mode” to transit its state beyond its direct neighborhood and in analogy of certain ant types.

The cellular ant data clustering method was evaluated using a number of different datasets. The clustering results using the computer hardware dataset (Ein-Dor and Feldmesser 1987) are provided in Fig. 3. Based on the results, the enhanced version exhibits better results in terms of clustering accuracy and overall performance. Due to the method’s simplicity, various modifications could improve the results and display similar or even better outcomes with state-of-the-art clustering techniques.

### Cellular Ants in Swarm Robotics

Due to the discretization of the operational area and the inherit locality feature, cellular ants comprise a very promising approach in the research field of robotics also. Both CAs and ACO algorithms have been utilized in robotic systems for providing a framework to estimate the motion of a robot inside its configuration area. For either obstacle avoidance, path planning, or any task execution process, the corresponding algorithms have produced very promising results. Nonetheless, due to complex environments and the need of exploiting less



**Cellular Ants Computing, Fig. 3** Data clustering results with cellular ants by applying: (a) long interaction rule, (b) progressive expansive grid, (c) long interaction and proportional distance measurement, and (d) all the above

computational resources, the use of simple, yet efficient, methods for such objectives is imperative.

A cellular ant-based model, named as cellular automata, was introduced to define the pathway of a robot in an operating area depending on its final objective (Rosenberg 2008). The approach considers that the robots operate in a type of intelligent floor in order to comply with the stigmergy feature of ACO algorithms. The floor is divided into multiple Finite-State Machines (FSM) in order to develop a two-dimensional mesh whose nodes form tiles. Every FSM can be denoted with finite values similar to a classical CA. The model considers also values for the pheromone concentrations except the states that correspond to robots.

Moreover, every FSM comprises both a transmitter and a receiver to simulate the communication that the robots must exhibit. The exchanged packets include information about (i) the presence of adjacent cellular ants, food, etc. and (ii) commands to the resident ant. The developed transition rules based on which every cellular ant will develop its state relied on the system's final objective. Three scenarios as final tasks were tested: (i) parking, (ii) food-finding, and (iii) maze-threading. Parking scenario aims at positioning cellular ants to the boundary of the grid (more precise to corners), while food-finding case targets to simulate the classic ACO problem (Fig. 2). Finally, the third scenario's objective was to determine the path of a robot, operating in a maze. The

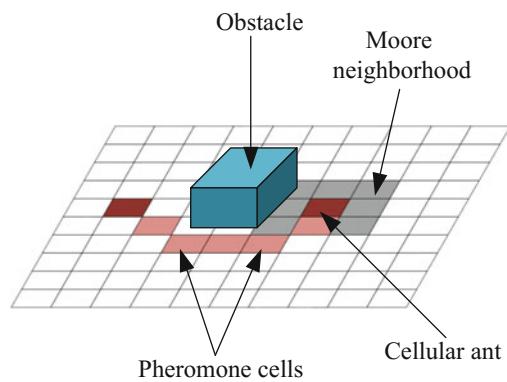
model had successfully defined the solution for the robots in order to accomplish their tasks.

Similar approach was also followed in Ioannidis et al. (2011) for a system of swarm robots. The developed model incorporates features of ACO algorithms into a CA in order to provide the system a framework to accomplish multiple objectives. The system is composed of multiple robots which must exhibit collective behavior in order to achieve one main objective as a team, e.g., to cover a specified distance in space retaining their initial formation. Every cell robot as individual must also be able to detect and bypass every obstacle found during its route to its final destination point. The model should provide solutions in both problems: obstacle avoidance and formation control.

Following the CA principles, the configuration area is considered as two-dimensional lattice of cells, while Moore neighborhood was used in order for a robot cell to have a complete overview of its surroundings. Moreover, the set of states includes one state for every member of the robot team as well as discrete values that correspond to pheromone concentrations. The entire set was divided into four subsets each for every type of cell, meaning free, obstacle, robot, and pheromone cell. Transition rules were also divided into subsets based on the desired operation: obstacle avoidance, formation control, and pheromone evaporation.

More specific, the method considers that potential obstacles should be avoided before proceeding to the next processing step. Thus, transition rules for obstacle avoidance are applied first. Every cellular ant searches its Moore neighborhood for the presence of an object. If an obstacle was found, the robot cellular ant will eventually bypass the obstacle with the application of the appropriate transition rules (Fig. 4).

In addition, appropriate transition rules were developed to simulate the evaporation process of the pheromones. Based on the defined evaporation rate, a pheromone cell decreases its value after some evolution time steps. Finally, the method simulates the collective behavior that the swarm must display by applying the proper transition rules. The team must proceed to its final

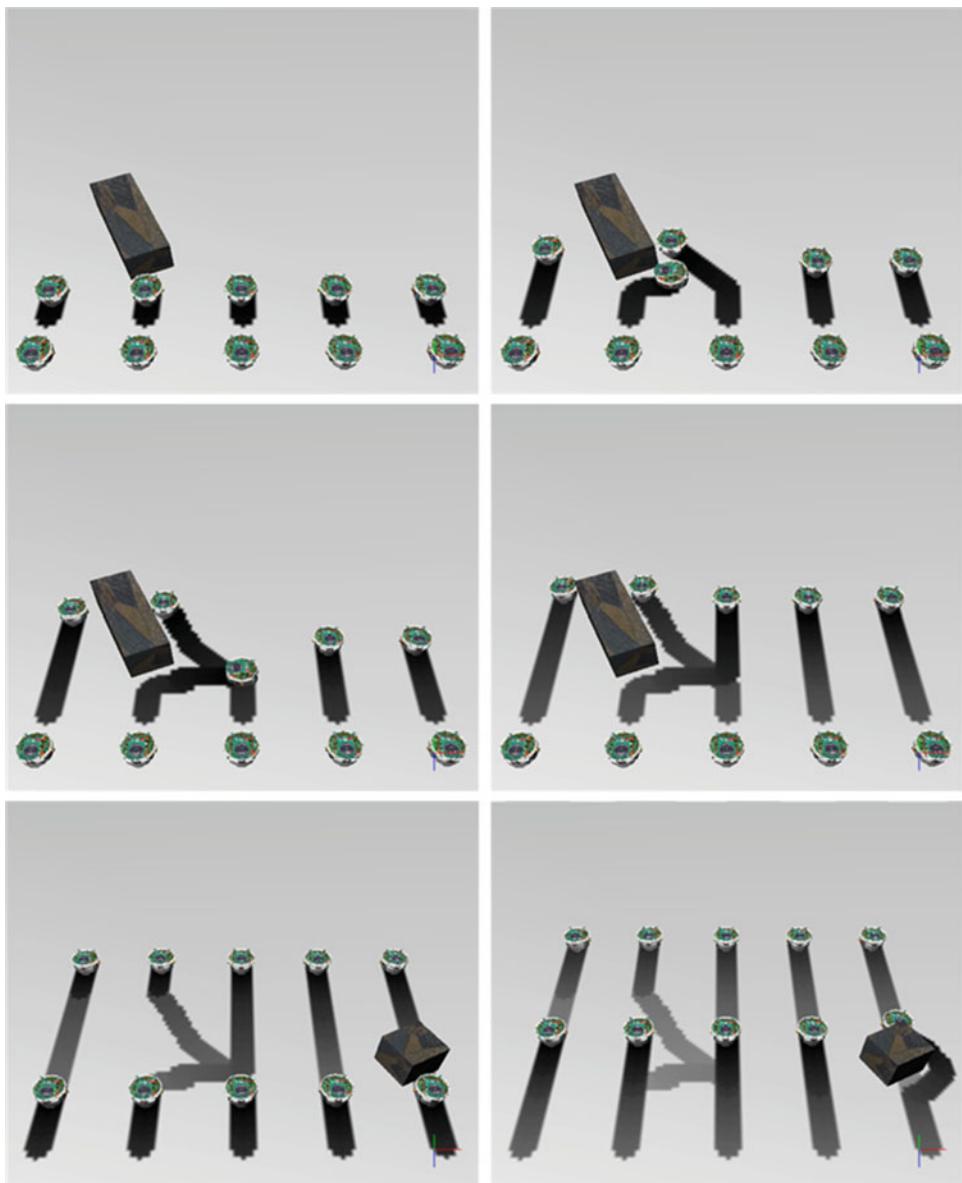


**Cellular Ants Computing, Fig. 4** Obstacle avoidance for a cellular ant

positions retaining its formation during the entire process. In cases of scattered formations, the cellular ants collaborate to exchange their position in the formation so that the initial structure could be regained as soon as possible. The method divides the entire team into subgroups, so after the first subgroup, the second sub-team starts covering the desired distance. During their transition to their final destination, each cellular ant deposits a quantity of pheromone, which is detected by the forthcoming cellular ants of the next sub-team. The pheromone trails assist the cellular ants to avoid the communication between their collaborative members leading to a more simple and efficient solution for the formation control process. The method was successfully implemented in “real robots” using a simulation environment and miniature robots with low capabilities (Fig. 5). The simplicity of the cellular ant approach and its low requirements for computational resources render its implementation possible in real swarm robotic teams.

## Future Directions

As we encounter novel computing environments that offer new opportunities while posing new challenges, many researchers focus on proposing and evaluating methods inspired from natural analogues of these environments. Novel algorithms are required in many cases to solve complex problems with the minimum possible resources.



**Cellular Ants Computing, Fig. 5** Cellular ant implementation approach in simulated swarm robotic team (from left to right and from upper to lower images)

Current literature includes algorithms that solve sufficiently various issues, but high amounts of computational and memory resources are required in real implementations.

Cellular ant algorithms and its numerous alterations can provide efficient solutions in many problems. The combination of two widely

known artificial intelligence methods, CA and ACO, can be severely beneficial in many applications exploiting the advantages of both methods. The cellular ant advantages include its algorithmic simplicity based on a few predefined variables and a small set of ant behavior rules that are completely determined by fixed, discrete

values. Therefore, the method displays low requirements for computational resources due to its simplicity. This advantage renders the method appropriate for implementations using low-cost systems. Nonetheless, the most significant contribution probably consists of the decentralized behavior of the method. Self-organization feature provides the ability to a system of evolving during time without the need of external interference. The system will modulate its actions based on internal processes in order to converge to its final objective. ACO principles contribute to accelerate this process, and in combination with a CA, the algorithm provides the advantage of inherit parallelism to the system.

Despite the adequate results of its applications, cellular ants are still in a very preliminary stage of exploitation. The method is limited to discrete and finite systems due to the CA nature. Thus, its adaptation in systems where real data must be processed will lead to a possible accuracy decrease. The use of possible relaxation of CA definition with the introduction of continuous space CA instead of the already defined discrete CA could solve this issue; nonetheless, multiplied computational resources will be required. In addition, inappropriate boundary conditions could lead to unexpected results especially when pheromone trails are detected on the grid's boundaries and a torus-like grid is used. Fortunately, due to the discretization of space, flexible options could be studied aiming at more natural solutions and avoiding potential deviations regarding the evolution of the system. Finally, enhanced and novel transition rules will increase the applicability of the method in more systems and not restrict its usage in specific applications. Modeling a problem with the use of proper evolution rules could allow the use of cellular ants and, thus, exploit fully their advantages.

## Bibliography

### Primary Literature

Alba E, Tomassini M (2002) Parallelism and evolutionary algorithms. *IEEE Trans Evol Comput* 6:443–462

- Albuquerque P, Dupuis A (2002) A parallel cellular ant colony algorithm for clustering and sorting. In: Bandini S, Chopard B, Tomassini M (eds) *Cellular Automata*. ACRI 2002. Lecture Notes in Computer Science, vol 2493. Springer, Berlin, Heidelberg, pp 220–230
- Bitsakidis NP, Chatzichristofis SA, Sirakoulis GC (2015) Hybrid cellular ants for clustering problems. *Int J Unconv Comput* 11(2):103–130
- Cantu-Paz E (2000) Efficient and accurate parallel genetic algorithms, 2000. Kluwer, New York
- Chen L, Xu X, Chen Y, He P (2004) A novel ant clustering algorithm based on cellular automata. In: Proceedings. IEEE/WIC/ACM international conference on intelligent agent technology, 2004. (IAT 2004), pp 148–154. <http://ieeexplore.ieee.org/document/1342937/>
- Di Caro G, Dorigo M (1998) AntNet: distributed stigmergetic control for communications networks. *J Artif Intell Res* 9:317–365
- Dorigo M (1992) Optimization, learning and natural algorithms. PhD thesis, Politecnico di Milano, Italy
- Dorigo M, Maniezzo V, Colorni A (1996) The ant system: optimization by a colony of cooperation agents. *IEEE Trans Syst Man Cybern* 26:29–41
- Ein-Dor P, Feldmesser J (1987) Attributes of the performance of central processing units: a relative performance prediction model. *Commun ACM* 30: 308–317
- Ioannidis K, Sirakoulis GC, Andreadis I (2011) Cellular ants: a method to create collision free trajectories for a cooperative robot team. *Robot Auton Syst* 59:113–127
- Ji J, Song X, Liu C, Zhang X (2013) Ant colony clustering with fitness perception and pheromone diffusion for community detection in complex networks. *Phys A* 392:3260–3272
- Konstantinidis K, Sirakoulis GC, Andreadis I (2009) Design and implementation of a fuzzy-modified ant colony hardware structure for image retrieval. *IEEE Trans Syst Man Cybern Part C Appl Rev* 39:520–533
- Li X, Lao C, Liu X, Chen Y (2011) Coupling urban cellular automata with ant colony optimization for zoning protected natural areas under a changing landscape. *Int J Geogr Inf Sci* 25:575–593
- Liu C, Li L, Xiang Y (2008) Research of multi-path routing protocol based on parallel ant colony algorithm optimization in mobile ad hoc networks. In: Information technology: new generations, 2008. Fifth international conference on ITNG 2008, pp 1006–1010. <http://ieeexplore.ieee.org/document/4492616/>
- Martens D, De Backer M, Haesen R, Vanthienen J, Snoeck M, Baesens B (2007) Classification with ant colony optimization. *IEEE Trans Evol Comput* 11:651–665
- Merkle D, Middendorf M (2002) Fast ant colony optimization on runtime reconfigurable processor arrays. *Genet Program Evolvable Mach* 3:345–361
- Moere AV, Clayden JJ (2005) Cellular ants: combining ant-based clustering with cellular automata. In: Tools with Artificial Intelligence, 2005. 17th IEEE international

- conference on ICTAI 05, p 8. <http://ieeexplore.ieee.org/document/1562933/>
- Omohundro S (1984) Modelling cellular automata with partial differential equations. *Phys D* 10:128–134
- Rosenberg AL (2008) Cellular automata: food-finding and maze-threading. In: Parallel processing, 2008, 37th international conference on ICPP'08, pp 528–535. <http://ieeexplore.ieee.org/document/4625890/>
- Scheuermann B, So K, Guntsch M, Middendorf M, Diessel O, ElGindy H, Schmeck H (2004) FPGA implementation of population-based ant colony optimization. *Appl Soft Comput* 4:303–322
- Sirakoulis GC, Karafyllidis I, Mardiris V, Thanailakis A (2000) Study of the effects of photoresist surface roughness and defects on developed profiles. *Semicond Sci Technol* 15:98
- Sirakoulis GC, Karafyllidis I, Thanailakis A (2003) A CAD system for the construction and VLSI implementation of cellular automata algorithms using VHDL. *Microprocess Microsyst* 27:381–396
- Toffoli T (1984) Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Phys D* 10:117–127
- Toffoli T, Margolus N (1987) Cellular automata machines: a new environment for modeling. MIT Press, Cambridge
- Ulam S (1952) Random processes and transformations. In: Proceedings of the international congress on mathematics, American Mathematical Society. pp 264–275. <https://archive.org/details/proceedingsofint00inte>
- Vichniac GY (1984) Simulating physics with cellular automata. *Phys D* 10:96–116
- Von Neumann J, Burks AW et al (1966) Theory of self-reproducing automata. IEEE Trans Neural Netw 5:3–14
- Yang X, Zheng X-Q, Lv L-N (2012) A spatiotemporal model of land use change based on ant colony optimization, Markov chain and cellular automata. *Ecol Model* 233:11–19

## Books and Reviews

- Bastien C, Michel D (1998) Cellular automata modeling of physical systems. Cellular automata modeling of physical systems. Cambridge University Press, New York
- Feynman RP (1982) Simulating physics with computers. *Int J Theor Phys* 21:467–488
- Petley C (1997) Diffusion (cellular) models. In: Back, Thomas, Fogel, David B, Halewicz, Zbigniew (eds) Handbook of Evolutionary Computation (IOP Publishing Ltd and Oxford University Press), pages C6.4:1–6



## Artificial Chemistry

Peter Dittrich

Department of Mathematics and Computer  
Science, Friedrich Schiller University Jena, Jena,  
Germany

### Article Outline

Glossary

Definition of the Subject

Introduction

Basic Building Blocks of an Artificial Chemistry

Structure-to Function Mapping

Space

Theory

Evolution

Information Processing

Future Directions

Bibliography

### Glossary

**(Chemical) Organization** A closed and self maintaining set of molecules.

**Autocatalytic set** A (self maintaining) set where each molecule is produced catalytically by molecules from that set. Note that an autocatalytic may produce molecules not present in that set.

**Closure** A set of molecules  $A$  is closed, if no combination of molecules from  $A$  can react to form a molecule outside  $A$ . Note that the term “closure” has been also used to denote the catalytical closure of an autocatalytic set.

**Molecular species** A molecular species is an abstract class denoting an ensemble of identical molecules. Equivalently the terms “species”, “compound”, or just “molecule” are used; in some specific context also the terms “substrate” or “metabolite”.

**Molecule** A molecule is a concrete instance of a molecular species. Molecules are those entities of an artificial chemistry that react. Note that sometimes the term “molecule” is used equivalently to molecular species.

**Multiset** A multiset is like a set but where elements can appear more than once; that is, each element has a multiplicity, e.g.,  $\{a, a, b, c, c, c\}$ .

**Order of a reaction** The order of a reaction is the sum of the exponents of concentrations in the kinetic rate law (if given). Note that an order can be fractional. If only the stoichiometric coefficients of the reaction rule are given Eq. (1), the order is the sum of the coefficients of the left-hand side species. When assuming mass action kinetics, both definitions are equivalent.

**Reaction network** A set of molecular species together with a set of reaction rules. Formally, a reaction network is equivalent to a Petri network. A reaction network describes the stoichiometric structure of a reaction system.

**Self maintaining** A set of molecules is called self maintaining, if it is able to maintain all its constituents. In a purely autocatalytic system under flow condition, this means that every molecule can be catalytically produced by at least one reaction among molecule from the set.

### Definition of the Subject

Artificial chemistries are chemical like systems or abstract models of chemical processes. They are studied in order to illuminate and understand fundamental principles of chemical systems as well as to exploit the chemical metaphor as a design principle for information processing systems in fields like chemical computing or nonlinear optimization.

An artificial chemistry (AC) is usually a formal (and, more seldom, a physical) system that consists of objects called molecules, which interact according to rules called reactions. Compared to

conventional chemical models, artificial chemistries are more abstract in the sense that there is usually not a one-to-one mapping between the molecules (and reactions) of the artificial chemistry to real molecules (and reactions). An artificial chemistry aims at capturing the logic of chemistry rather than trying to explain a particular chemical system. More formally, an artificial chemistry can be defined by a set of molecular species  $\mathcal{M}$ , a set of reaction rules  $\mathcal{R}$ , and specifications of the dynamics, e.g., kinetic laws, update algorithm, and geometry of the reaction vessel.

The scientific motivation for AC research is to build abstract models in order to understand chemical like systems in all kind of domains ranging from physics, chemistry, biology, computer science, and sociology. Abstract chemical models to study fundamental principles, such as spatial pattern formation and self replication, can be traced back to the beginning of modern computer science in the 40s (Turing 1952; von Neumann and Burks 1966). Since then a growing number of approaches have tackled questions concerning the origin of complex forms, the origin of life itself (Eigen 1971; Segre et al. 2000), or cellular diversity (Kauffman 1993). In the same way a variety of approaches for constructing ACs have appeared, ranging from simple ordinary differential equations (Eigen and Schuster 1977) to complex individual based algebraic transformation systems (Fontana 1992; Hofstadter 1979; Laing 1972).

The engineering motivation for Acre search aims at employing the chemical metaphor as a programming and computing paradigm. Approaches can be distinguished according to whether chemistry serves as a paradigm to program or construct conventional *in silico* information processing systems (Banâtre and Métayer 1986), Or whether real molecules are used for information processing like in molecular computing (Conrad 1992) or DNA computing (Adleman 1994) (► [Cellular Computing](#), ► [DNA Computing](#)).

## Introduction

The term artificial chemistry appeared around 1990 in the field of artificial life (Rasmussen

et al. 1990). However, models that fell under the definition of artificial chemistry appeared also decades before, such as von Neumann's universal self replicating automata (von Neumann and Burks 1966). In the 50s, a famous abstract chemical system was introduced by Turing (1952) in order to show how spatial diffusion can destabilize a chemical system leading to spatial pattern formation. Turing's artificial chemistries consist of only a handful of chemical species interacting according to a couple of reaction rules, each carefully designed. The dynamics is simulated by a partial differential equation.

Turing's model possesses the structure of a conventional reaction kinetics chemical model (subsection “[The Chemical Differential Equation](#)”). However it does not aim at modeling a particular reaction process, but aims at exploring how, in principle, spatial patterns can be generated by simple chemical processes. Turing's artificial chemistries allow one to study whether a particular mechanism (e.g., destabilization through diffusion) can explain a particular phenomena (e.g., symmetry breaking and pattern formation).

The example illustrates that studying artificial chemistries is more a synthetic approach. That is, understanding should be gained through the synthesis of an artificial system and its observation under various conditions (Kaneko 2007; Langton 1989). The underlying assumption is that understanding a phenomena and how this phenomena can be (re-)created (without copying it) is closely related (Vico 1710).

Today's artificial chemistries are much more complex than Turing's model. They consist of a huge, sometimes infinite, amount of different molecular species and their reaction rules are defined not manually one by one, but implicitly (section “[Structure-to Function Mapping](#)”) and can evolve (section “[Evolution](#)”). Questions that are tackled are the structure and organization of large chemical networks, the origin of life, prebiotic chemical evolution, and the emergence of chemical information and its processing.

## Introductory Example

It is relatively easy to create a complex, infinitely sized reaction network. Assume that the set of

possible molecules  $\mathcal{M}$  are strings over an alphabet. Next, we have to define a reaction function describing what happens when two molecules  $a_1, a_2 \in \mathcal{M}$  collide. A general approach is to map  $a_1$  to a machine, operator, or function  $f = \text{fold}(a_1)$  with  $f: \mathcal{M} \rightarrow \mathcal{M} \cup \{\text{elastic}\}$ . The mapping fold can be defined in various ways, such as by interpreting  $a_1$  as a computer program, a Turing machine, or a lambda expression (section “[Structure-to Function Mapping](#)”). Next, we apply the machine  $f$  derived from  $a_1$  to the second molecule. If  $f(a_2) = \text{elastic}$  the molecules collided elastically and nothing happens. This could be the case if the machine  $f$  does not halt within a pre-defined amount of time. Otherwise the molecules react and a new molecule  $a_3 = f(a_2)$  is produced. How this new molecule changes the reaction vessel depends on the algorithm simulating the dynamics of the vessel. Algorithm 1 is a simple, widely applied algorithm that simulates second order catalytic reactions under flow conditions: Because the population size is constant and thus limited, there is competition and only those molecules that are somehow created will sustain.

### Algorithm 1

```

INPUT: P : population (array of k
molecules)
    randomPosition() := randomInt
(0, k-1);
    reaction(r1, r2) := (fold(r1))
(r2);
    while not terminate do
        reactant1 := P[randomPosition
()];
        reactant2 := P[randomPosition
()];
        product := reaction(reactant1,
reactant2);
        if product == not elastic
            P[randomPosition()] := product;
        fi
        t := t + 1/k ; increment simulated
time
    od

```

First, the algorithm chooses two molecules randomly, which simulates a **collision**. Note that

a realistic measure of time takes the amount of collision and not the amount of reaction events into account. Then, we determine stochastically whether the selected molecules react. If the molecules react, a randomly selected molecule from the population is replaced by the new product, which assures that the population size stays constant. The replaced molecules constitute the outflow. The inflow is implicitly assumed and is generated by the catalytic production of molecules. From a chemical point of view, the algorithm assumes that a product is built from an energy rich substrate, which does not appear in the algorithm and which is assumed to be available at constant concentration.

The following section (“[Basic Building Blocks of an Artificial Chemistry](#)”) describes the basic building blocks of an artificial chemistry in more detail. In section “[Structure-to Function Mapping](#)” we will explore various techniques to define reaction rules. The role of space is briefly discussed in section “[Space](#)”. Then the important theoretical concepts of an autocatalytic set and a chemical organization are explained (section “[Theory](#)”). Section “[Evolution](#)” shows how artificial chemistries can be used to study (chemical) evolution. This article does not discuss information processing in detail, because there are a series of other specialized articles on that topic, which are summarized in section “[Information Processing](#)”.

## Basic Building Blocks of an Artificial Chemistry

### Molecules

The first step in defining an AC requires that we define the set of all molecular species that can appear in the AC. The easiest way to specify this set is to enumerate explicitly all molecules as symbols. For example:  $\mathcal{M} = \{H_2, O_2, H_2O, H_2O_2\}$  or, equivalently,  $\mathcal{M} = \{a, b, c, d\}$ . A symbol is just a name without any structure. Most conventional (bio-)chemical reaction system models are defined this way. Also many artificial chemistries where networks are designed “by hand” (Eigen and Schuster 1977), are created randomly

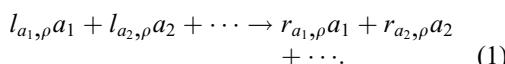
(Stadler et al. 1993), or are evolved by changing the reaction rules (Jain and Krishna 1998, 1999, 2001) use only symbolic molecules without any structure.

Alternatively, molecules can possess a structure. In that case, the set of molecules is implicitly defined. For example:  $\mathcal{M} = \{\text{all polymers that can be made from the two monomers } a \text{ and } b\}$ . In this case, the set of all possible molecules can even become infinite, or at least quite large. A vast variety of such rule-based molecule definitions can be found in different approaches. For example, structured molecules may be abstract character sequences (Bagley and Farmer 1992; Farmer et al. 1986; Kauffman 1993; McCaskill et al. 1994), sequences of instructions (Adami and Brown 1994; Hutton 2002), lambda expressions (Fontana 1992), combinatorics (Speroni di Fenizio 2000), binary strings (Banzhaf 1993; Dittrich and Banzhaf 1998; Thürk 1993), numbers (Banzhaf et al. 1996), machine code (Rasmussen et al. 1990), graphs (Salzberg 2007), swarms (Sayama 2009), or proofs (Fontana and Buss 1996).

We can call a molecule's representation its **structure**, in contrast to its **function**, which is given by the reaction rules  $\mathcal{R}$ . The description of the valid molecules and their structure is usually the first step when an AC is defined. This is analogous to a part of chemistry that describes what kind of atom configurations form stable molecules and how these molecules appear.

## Reaction Rules

The set of reaction rules  $\mathcal{R}$  describes how molecules from  $\mathcal{M} = \{a_1, a_2, \dots\}$  interact. A rule  $\rho \in \mathcal{R}$  can be written according to the chemical notation of reaction rules in the form



The stoichiometric coefficients  $l_{a, \rho}$  and  $r_{a, \rho}$  describe the amount of molecular species  $a \in \mathcal{M}$  in reaction  $\rho \in \mathcal{R}$  on the left-hand and right-hand side, respectively. Together, the stoichiometric coefficients define the **stoichiometric matrix**

$$\mathbf{S} = (s_{a, \rho}) = (r_{a, \rho} - l_{a, \rho}). \quad (2)$$

An entry  $s_{a, \rho}$  of the stoichiometric matrix denotes the net amount of molecules of type  $a$  produced in reaction  $\rho$ .

A reaction rule determines a multiset of molecules (the left-hand side) that can react and subsequently be replaced by the molecules on the right-hand side. Note that the sign “+” is not an operator here, but should only separate the components on either side. The set of all molecules  $\mathcal{M}$  and a set of reaction rules  $\mathcal{R}$  define the **reaction network**  $\langle \mathcal{M}, \mathcal{R} \rangle$  of the AC. The reaction network is equivalent to a Petri net (Petri 1962). It can be represented by two matrices,  $(l_{a, \rho})$  and  $(r_{a, \rho})$ , made up of the stoichiometric coefficients. Equivalently we can represent the reaction network as a hyper-graph or a directed bipartite graph with two node types denoting reaction rules and molecular species, respectively, and edge labels for the stoichiometry.

A rule is applicable only if certain conditions are fulfilled. The major condition is that all of the left-hand side components must be available. This condition can be broadened easily to include other parameters such as neighborhood, rate constants, probability for a reaction, influence of modifier species, or energy consumption. In such a case, a reaction rule would also contain additional information or further parameters. Whether or not these additional predicates are taken into consideration depends on the objectives of the artificial chemistry. If it is meant to simulate real chemistry as accurate as possible, it is necessary to integrate these parameters into the simulation system. If the goal is to build an abstract model, many of these parameters can be omitted.

Like for the set of molecules we can define the set of reaction rules explicitly by enumerating all rules symbolically (Eigen and Schuster 1977), or we can define it implicitly by referring to the structure of the molecules. Implicit definitions of reaction rules use, for example, string matching/string concatenation (Bagley and Farmer 1992; Lugowski 1989; McCaskill et al. 1994), lambda calculus (Fontana 1992; Fontana and Buss 1994), Turing machines (Ikegami and Hashimoto 1995; Thürk 1993), finite state machines (Zauner and

Conrad 1996) or machine code language (Adami and Brown 1994; Dittrich and Banzhaf 1998; Salzberg 2007; Suzuki and Ikegami 2006), proof theory (Fontana and Buss 1994), matrix multiplication (Banzhaf 1993), swarm dynamics (Sayama 2009), or simple arithmetic operations (Banzhaf et al. 1996). Note that in some cases the reactions can emerge as the result of the interaction of many atomic particles, whose dynamics is specified by force fields or rules (Sayama 2009). Section “Structure-to Function Mapping” will discuss implicitly defined reactions in more details.

### Dynamics

The third element of an artificial chemistry is its specification of how the reaction rules give rise to a dynamic process of molecules reacting in some kind of reaction vessel. It is assumed that the reaction vessel contains multiple copies of molecules, which can be seen as instances of the molecular species  $\mathcal{M}$ .

This section summarizes how the dynamics of a reaction vessel (which usually contains a huge number of molecules) can be modeled and simulated. The approaches can be characterized roughly by whether each molecule is treated explicitly or whether all molecules of one type are represented by a number denoting their frequency or concentration.

### The Chemical Differential Equation

A reaction network  $\mathcal{M}, \mathcal{R}$  specifies the structure of a reaction system, but does not contain any notion of time. A common way to specify the dynamics of the reaction system is by using a system of ordinary differential equations of the following form:

$$\dot{\mathbf{x}}(t) = \mathbf{Sv}(\mathbf{x}(t)), \quad (3)$$

where  $\mathbf{x} = (x_1, \dots, x_m)^T \in \mathbb{R}^m$  is a concentration vector depending on time  $t$ ,  $\mathbf{S}$  the stoichiometric matrix derived from  $\mathcal{R}$  Eq. (2), and  $\mathbf{v} = (v_1, \dots, v_r)^T \in \mathbb{R}^r$  a flux vector depending on the current concentration vector. A flux  $v_\rho \geq 0$  describes the velocity or turnover rate of reaction  $\rho \in \mathcal{R}$ . The actual value of  $v_\rho$  depends usually on the concentration of the species participating in

the reaction  $\rho$  (i.e.,  $\text{LHS}(\rho) \equiv \{a \in \mathcal{M} : l_{a,\rho} > 0\}$ ) and sometimes on additional modifier species, whose concentration is not influenced by that reaction. There are two important assumptions that are due to the nature of reaction systems. These assumptions relate the kinetic function  $\mathbf{v}$  to the reaction rules  $\mathcal{R}$ :

### Assumption 1

A reaction  $\rho$  can only take place if all species of its left-hand side  $\text{LHS}(\rho)$  are present. This implies that for all molecules  $a \in \mathcal{M}$  and reactions  $\rho \in \mathcal{R}$  with  $a \in \text{LHS}(\rho)$ , if  $x_a = 0$  then  $v_\rho = 0$ . The flux  $v_\rho$  must be zero, if the concentration  $x_a$  of a molecule appearing on the left-hand side of this reaction is zero. This assumption meets the obvious fact that a molecule has to be present to react.

### Assumption 2

If all species  $\text{LHS}(\rho)$  of a reaction  $\rho \in \mathcal{R}$  are present in the reactor (e.g. for all  $a \in \text{LHS}(\rho)$ ,  $x_a > 0$ ) the flux of that reaction is positive, (i.e.,  $v_\rho > 0$ ). In other words, the flux  $v_\rho$  must be positive, if all molecules required for that reaction are present, even in small quantities. Note that this assumption implies that a modifier species necessary for that reaction must appear as a species on the left- (and right-) hand side of  $\rho$ .

In short, taking Assumption 1 and 2 together, we demand for a chemical differential equation:

$$v_\rho > 0 \Leftrightarrow \forall a \in \text{LHS}(\rho), x_a > 0. \quad (4)$$

Note that Assumption 1 is absolutely necessary. Although Assumption 2 is very reasonable, there might be situations where it is not true. Assume, for example, a reaction of the form  $a + a \rightarrow b$ . If there is only one single molecule  $a$  left in the reaction vessel, the reaction cannot take place. Note however, if we want to model this effect, an ODE is the wrong approach and we should choose a method like those described in the following sections.

There are a large number of kinetic laws fulfilling these assumptions Eq. (4), including all laws that are usually applied in practice. The most fundamental of such kinetic laws is **mass**

**action kinetics**, which is just the product of the concentrations of the interacting molecules (cf. Gillespie 1976):

$$v_\rho(\mathbf{x}) = \prod_{a \in \mathcal{M}} x_a^{l_{a,\rho}}. \quad (5)$$

The sum of the exponents in the kinetic law  $\sum_{a \in \mathcal{M}} l_{a,\rho}$  is called the **order** of the reaction. Most more complicated laws like Michaelis–Menten kinetics are derived from mass action kinetics. In this sense, mass action kinetics is the most general approach, allowing one to emulate all more complicated laws derived from it, but at the expense of a larger amount of molecular species.

### Explicitly Simulated Collisions

The chemical differential equation is a time and state continuous approximation of a discrete system where molecules collide stochastically and react. In the introduction we saw how this can be simulated by a rather simple algorithm, which is widely used in AC research (Banzhaf 1993; Dittrich and Banzhaf 1998; Fontana 1992). The algorithm presented in the introduction simulates a second order catalytic flow system with constant population size. It is relatively easy to extend this algorithm to include reaction rates and arbitrary orders.

### Algorithm 2

```
while not terminate() do
    reactands := chooseASubsetFrom
    (P);
    if randomReal
    (0, 1) < reactionProbability
    (reactands);
    products = reaction(reactands);
    P := remove(P, reactands);
    P := insert(P, products);
    fi
    t := t + 1 / sizeOf(P); ; increment
    simulated time
od
```

First, the Algorithm 2 chooses a subset from  $P$ , which simulates a collision among molecules. Note that the resulting subset could be empty, which can be used to simulate an inflow (i.e.,

reactions of the form  $\rightarrow A$ , where the left-hand side is empty). By defining the probability of obtaining a subset of size  $n$ , we can control the rate of reactions of order  $n$ . If the molecules react, the reactants are removed from the population and the products are added. Note that this algorithm can be interpreted as a stochastic rewriting system operating on a multiset of molecules (Păun 2000; Suzuki and Tanaka 1997).

For large population size  $k$ , the dynamics created by the algorithm tends to the continuous dynamics of the chemical differential equation assuming mass action kinetics. For the special case of second order catalytic flow, as described by the algorithm in the introduction, the dynamics can be described by the catalytic network equation (Stadler et al. 1993)

$$\begin{aligned} \dot{x}_k &= \sum_{i,j \in \mathcal{M}} \alpha_{i,j}^k x_i x_j - x_k \Phi(\mathbf{x}), \text{with} \\ \Phi(\mathbf{x}) &= \sum_{i,j,k \in \mathcal{M}} \alpha_{i,j}^k x_i x_j, \end{aligned} \quad (6)$$

which is a generalization of the replicator equation (Hofbauer and Sigmund 1988). If the reaction function is deterministic, we can set the kinetic constants to

$$\alpha_{i,j}^k = \begin{cases} 1 & \text{reaction}(i,j) = k, \\ 0 & \text{otherwise}. \end{cases} \quad (7)$$

This dynamic is of particular interest, because competition is generated due to a limited population size. Note that the limited population size is equivalent to an unlimited population size with a limited substrate, e.g., (Bagley and Farmer 1992).

### Discrete Event Simulation

If the copy number of a particular molecular species becomes high or if the reaction rates differ strongly, it is more efficient to use discrete event simulation techniques. The most famous technique is Gillespie's algorithm (Gillespie 1976). Roughly, in each step, the algorithm generates two random numbers depending on the current population state to determine the next reaction to occur as well as the time interval  $\Delta t$  when it will occur. Then the simulated time is advanced

$t := t + \Delta t$ , and the molecule count of the population updated according to the reaction that occurred.

The Gillespie algorithm generates a statistically correct trajectory of the chemical master equation. The chemical master equation is the most general way to formulate the stochastic dynamics of a chemical system. The chemical master equation is a first-order differential equation describing the time evolution of the probability of a reaction system to occupy each one of its possible discrete set of states. In a well stirred (non spatial) AC a state is equivalent to a multiset of molecules.

## Structure-to Function Mapping

A fundamental “logic” of real chemical systems is that molecules posses a structure and that this structure determines how the molecule interacts with other molecules. Thus, there is a mapping from the structure of a molecule to its function, which determines the reaction rules.

In real chemistry the mapping from structure to function is given by natural or physical laws. In artificial chemistries the structure-to function mapping is usually given by some kind of algorithm. In most cases, an artificial chemistry using a structure-to function mapping does not aim at modeling real molecular dynamics in detail. Rather the aim is to capture the fact that there is a structure, a function, and a relation between them.

Having an AC with a structure-to function mapping at hand, we can study strongly constructive dynamical systems (Fontana and Buss 1994, 1996). These are systems where, through the interaction of its components, novel molecular species appear.

### Example: Lambda Calculus (*AlChem*)

This section demonstrates a structure-to function that employs a concept from computer science: the  $\lambda$ -calculus. The  $\lambda$ -calculus has been used by Fontana (1992) and Fontana and Buss (1994) to define a constructive artificial chemistry.

In the so called *AlChem*, a molecule is a normalized  $\lambda$ -expression. A  $\lambda$ -expression is a word

over an alphabet  $A = \{\lambda, ., ,\} \cup V$  where  $V = \{x_1, x_2, \dots\}$  is an infinite set of available variable names. The set of  $\lambda$  expressions  $\Gamma$  is defined for  $x \in V, s_1 \in \Gamma, s_2 \in \Gamma$  by

$$\begin{array}{ll} x \in \Gamma & \text{variable name,} \\ \lambda x. s_2 \in \Gamma & \text{abstraction,} \\ (s_2)s_1 \in \Gamma & \text{application.} \end{array}$$

An abstraction  $\lambda x. s_2$  can be interpreted as a function definition, where  $x$  is the parameter in the “body”  $s_2$ . The expression  $(s_2)s_1$  can be interpreted as the application of  $s_2$  on  $s_1$ , which is formalized by the rewriting rule

$$(\lambda x. s_2)s_1 = s_2[x \leftarrow s_1], \quad (8)$$

where  $s_2[x \leftarrow s_1]$  denotes the term which is generated by replacing every unbounded occurrence of  $x$  in  $s_2$  by  $s_1$ . A variable  $x$  is bounded if it appears in in a form like  $\dots(\lambda x. \dots x. \dots)\dots$ . It is also not allowed to apply the rewriting rule if a variable becomes bounded. Example: Let  $s_1 = \lambda x_1.(x_1)\lambda x_2.x_2$  and  $s_2 = \lambda x_3.x_3$  then we can derive:

$$\begin{aligned} (s_1)s_2 &\Rightarrow (\lambda x_1.(x_1)\lambda x_2.x_2)\lambda x_3.x_3 \\ &\Rightarrow (\lambda x_3.x_3)\lambda x_2.x_2 \Rightarrow \lambda x_2.x_2. \end{aligned} \quad (9)$$

The simplest way to define a set of second order catalytic reactions  $\mathcal{R}$  by applying a molecule  $s_1$  to another molecule  $s_2$ :

$$s_1 + s_2 \rightarrow s_1 + s_2 + \text{normalForm}((s_1)s_2). \quad (10)$$

The procedure *normalForm* reduces its argument term to normal form, which is in practice bounded by a maximum of available time and memory; if these resources are exceeded before termination, the collision is considered to be elastic. It should be noted that the  $\lambda$ -calculus allows an elegant generalization of the collision rule by defining it by  $\lambda$ -expression  $\Phi \in \Gamma : s_1 + s_2 \rightarrow s_1 + s_2 + \text{normalForm}(((\Phi)(s_1)s_2))$ .

In order to simulate the artificial chemistry, Fontana and Buss (1994) used an algorithm like the algorithm described in subsection “Explicitly

[Simulated Collisions](#)”, which simulates second order catalytic mass action kinetics in a well-stirred population under flow condition.

*AlChem*y possesses a couple of important general properties: Molecules come in two forms: as passive data possessing a structure and as active machines operating on those structures. This generates a “strange loop” like in typogenetics (Hofstadter 1979; Vareto 1993), which allows molecules to refer to themselves and to operate on themselves. Structures operate on structures, and by doing so change the way structures operate on structures, and so on; creating a “strange”, self referential dynamic loop. Obviously, there are always some laws that cannot be changed, which are here the rules of the lambda calculus, defining the structure-to function mapping. Thus, we can interpret these fixed and predefined laws of an artificial chemistry as the natural or physical laws.

### Arithmetic and Logic Operations

One of the most easy ways to define reaction rules implicitly is to apply arithmetic operations taken from mathematics. Even the simplest rules can generate interesting behaviors. Assume, for example, that the molecules are natural numbers:  $\mathcal{M} = \{0, 1, 2, \dots, n - 1\}$  and the reaction rules are defined by division: reaction( $a_1, a_2$ ) =  $a_1/a_2$  if  $a_1$  is a multiple of  $a_2$ , otherwise the molecules do not react. For the dynamics we assume a finite, discrete population and an algorithm like the one described in subsection “[Explicitly Simulated Collisions](#)”. With increasing population size the resulting reaction system displays a phase transition, at which the population is able to produce prime numbers with high probability (see Banzhaf et al. 1996 for details). In a typical simulation, the initially high diversity of a random population is reduced leaving a set of non reacting prime numbers.

A quite different behavior can be obtained by simply replacing the division by addition: reaction( $a_1, a_2$ ) =  $a_1 + a_2 \bmod n$  with  $n = |\mathcal{M}|$  the number of molecular species. Initializing the well-stirred tank reactor only with the molecular species 1, the diversity rapidly increases towards its maximum and the reactor behaves apparently totally chaotic after a very short transient phase

(for a sufficiently large population size). However, there are still regularities, which depend on the prime factors of  $n$  (cf. Dittrich 2001).

### Matrix Multiplication Chemistry

More complex reaction operators can be defined that operate on vectors or strings instead of scalar numbers as molecules. The matrix multiplication chemistry introduce by Banzhaf (1993, 1994, 1995) uses binary strings as molecules. The reaction between two binary strings is performed by folding one of them into a matrix which then operates on the other string by multiplication.

### Example of a reaction for 4-bit molecules

Assume a reaction  $s_1 + s_2 \Rightarrow s_3$ . The general approach is:

1. Fold  $s_1$  to matrix  $M$ . Example:  $s_1 = (s_1^1, s_1^2, s_1^3, s_1^4)$

$$M = \begin{pmatrix} s_1^1 & s_1^2 \\ s_1^3 & s_1^4 \end{pmatrix}. \quad (11)$$

2. Multiply  $M$  with subsequences of  $s_2$ . Example: Let  $s_2 = (s_2^1, s_2^2, s_2^3, s_2^4)$  be divided into two subsequences  $s_2^{12} = (s_2^1, s_2^2)$  and  $s_2^{34} = (s_2^3, s_2^4)$ . Then we can multiply  $M$  with the subsequences:

$$s_3^{12} = M \odot s_2^{12}, s_3^{34} = M \odot s_2^{34}. \quad (12)$$

3. Compose  $s_3$  by concatenating the products. Example:  $s_3 = s_3^{12} \oplus s_3^{34}$ .

There are various ways of defining the vector matrix product  $\odot$ . It was mainly used with the following **threshold multiplication**. Given a bit vector  $x = (x_1, \dots, x_n)$  and a bit matrix  $M = (M_{ij})$  then the term  $y = M \odot x$  is defined by:

$$y_j = \begin{cases} 0 & \text{if } \sum_{i=1}^n x_i M_{i,j} \leq \Phi. \\ 1 & \text{otherwise.} \end{cases} \quad (13)$$

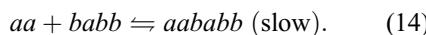
The threshold multiplication is similar to the common matrix vector product, except that the resulting vector is mapped to a binary vector by using the threshold  $\Phi$ .

Simulating the dynamics by an ODE or explicit molecular collisions (subsection “[Explicitly Simulated Collisions](#)”), we can observe that such a system would develop into a steady state where some string species support each other in production and thus become a stable autocatalytic cycle, whereas others would disappear due to the competition in the reactor.

The system has a couple of interesting properties: Despite the relatively simple definition of the basic reaction mechanism, the resulting complexity of the reaction system and its dynamics is surprisingly high. Like in typogenetics and Fontana’s lambda chemistry, molecules appear in two forms; as passive data (binary strings) and as active operators (binary matrix). The folding of a binary string to a matrix is the central operation of the structure-to function mapping. The matrix is multiplied with substrings of the operand and thus some kind of locality is preserved, which mimics local operation of macromolecules (e.g. ribosomes or restriction enzymes) on polymers (e.g. RNA or DNA). Locality is also conserved in the folding, because bits that are close in the string are also close in the matrix.

### Autocatalytic Polymer Chemistries

In order to study the emergence and evolution of autocatalytic sets (Eigen [1971](#); Kauffman [1971](#); Rössler [1971](#)) Bagley, Farmer, Fontana, Kauffman and others (Bagley and Farmer [1992](#); Farmer et al. [1986](#); Kauffman [1986, 1993](#); Lohn et al. [1998](#)) have used artificial chemistries where the molecules are character sequences (e.g.,  $\mathcal{M} = \{a, b, aa, ab, ba, bb, aaa, aab \dots\}$ ) and the reactions are concatenation and cleavage, for example:



Additionally, each molecule can act as a catalyst enhancing the rate of a concatenation reaction.

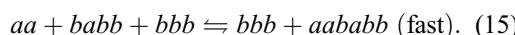


Figure [1](#) shows an example of an autocatalytic network that appeared. There are two time scales.

Reactions that are not catalyzed are assumed to be very slow and not depicted, whereas catalyzed reactions, inflow, and decay are fast.

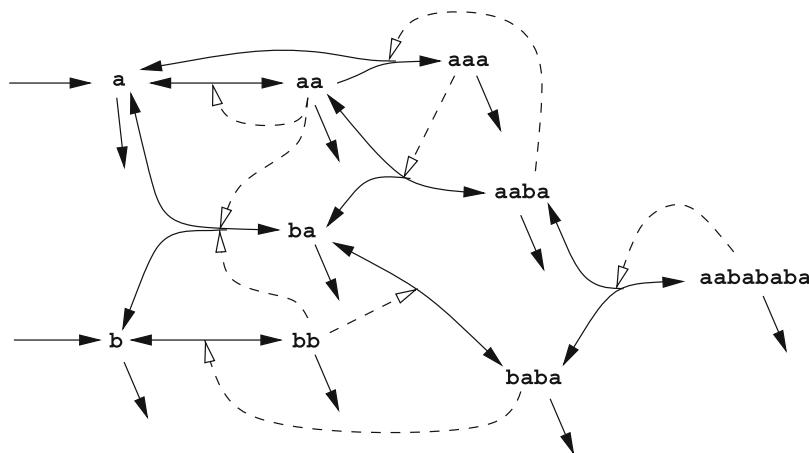
Typical experiments simulate a well-stirred flow reactor using meta dynamical ODE frame work (Bagley and Farmer [1992](#)); that is, the ODE model is dynamically changed when new molecular species appear or present species vanish. Note that the structure of the molecules does not fully determine the reaction rules. Which molecule catalyzes which reaction (the dotted arrows in Fig. [1](#)) is assigned explicitly randomly. An interesting aspect is that the catalytic or autocatalytic polymer sets (or reaction networks) evolve without having a genome (Kauffman [1986, 1993](#)). The simulation studies show how small, spontaneous fluctuations can be amplified by an autocatalytic network, possibly leading to a modification of the entire network (Bagley and Farmer [1992](#)). Recent studies by Fernando and Rowe ([2007](#)) include further aspects like energy conservation and compartmentalization.

Bagley and Farmer ([1992](#)) showed that autocatalytic sets can be silhouetted against a noisy background of spontaneously reacting molecules under moderate (that is, neither too low nor too high) flow.

### Artificial Chemistries Inspired by Turing Machines

The concept of an abstract automaton or Turing machine provides a base for a variety of structure function mappings. In these approaches molecules are usually represented as a sequence of bits, characters, or instructions (Hofstadter [1979](#); Laing [1972](#); McCaskill [1988](#); Suzuki [2007](#)). A sequence of bits specifies the behavior of a machine by coding for its state transition function. Thus, like in the matrix multiplication chemistry and the lambda chemistry, a molecule appears in two forms, namely as passive data (e.g., a binary string) and as an active machine. Also here we can call the mapping from a binary string into its machine **folding**, which might be indeterministic and may depend on other molecules (e.g., Ikegami and Hashimoto [1995](#)).

In the early 1970s Laing ([1972](#)) argued for abstract, non analogous models in order to



**Artificial Chemistry, Fig. 1** Example of an autocatalytic polymer network. The dotted lines represent catalytic links, e.g.,  $aa + ba + aaa \rightarrow aaba + aaa$ . All molecules are subject to a dilution flow. In this example, all polymerization reactions are reversible and there is a continuous

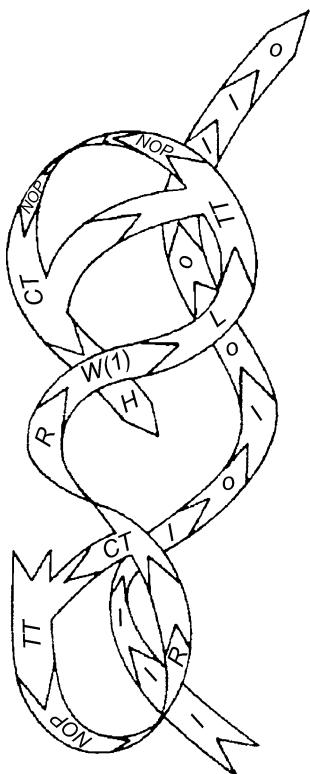
inflow of the monomers *a* and *b*. Note that the network contains further auto catalytic networks, for example,  $\{a, b, aa, ba\}$  and  $\{a, aa\}$ , of which some are closed and thus organizations, for example,  $\{a, b, aa, ba\}$ . The set  $\{a, aa\}$  is not closed, because *b* and *ba* can be produced

develop a general theory for living systems. For developing such general theory, so called **artificial organisms** would be required. Laing suggested a series of artificial organisms (Laing 1972, 1975, 1977) that should allow one to study general properties of life and thus would allow one to derive a theory which is not restricted to the instance of life we observe on earth. The artificial organisms consist of different compartments, e.g., a “brain” plus “body” parts. These compartments contain binary strings as molecules. Strings are translated to a sequence of instructions forming a three dimensional shape (cf. Salzberg 2007; Suzuki 2007; Vareto 1993). In order to perform a reaction, two molecules are attached such that they touch at one position (Fig. 2). One of the molecules is executed like a Turing machine, manipulating the passive data molecule as its tape. Laing proved that his artificial organisms are able to perform universal computation. He also demonstrated different forms of self reproduction, self description and self inspection using his molecular machines (Laing 1977).

*Typogenetics* is a similar approach, which was introduced in 1979 by Hofstadter in order to illustrate the “formal logic of life” (Hofstadter 1979). Later, *typogenetics* was simulated and investigated in more detail (Morris 1989; Vareto 1993,

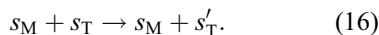
1998). The molecules of *typogenetics* are character sequences (called strands) over the alphabet A, C, G, T. The reaction rules are “typographic” manipulations based on a set of predefined basic operations like cutting, insertion, or deletion of characters. A sequence of such operations forms a unit (called an enzyme) which may operate on a character sequence like a Turing machine on its tape. A character string can be “translated” to an enzyme (i.e., a sequence of operations) by mapping two characters to an operation according to a predefined “genetic code”.

Another artificial chemistry whose reaction mechanism is inspired by the Turing machine was suggested by McCaskill (1988) in order to study the self organization of complex chemical systems consisting of catalytic polymers. Variants of this AC were realized later in a specially designed parallel reconfigurable computer based on FPGAs – Field Programmable Gate Arrays (Breyer et al. 1999; Ehricht et al. 1997; Tangen et al. 1997). In this approach, molecules are binary strings of fixed length (e.g., 20 (McCaskill 1988)) or of variable length (Breyer et al. 1999). As in previous approaches, a string codes for an automaton able to manipulate other strings. And again, pattern matching is used to check whether two molecules can react and to obtain the “binding



**Artificial Chemistry, Fig. 2** Illustration of Laing'smolecular machines. A program molecule is associated with a data molecule. (Figure from Laing (1975))

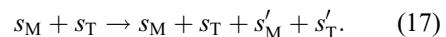
site”; i.e., the location where the active molecule (machine) manipulates the passive molecule (tape). The general reaction scheme can be written as:



In experimental studies, self replicating strings appeared frequently. In coevolution with parasites, an evolutionary arms race started among these species and the self replicating string diversified to an extent that the parasites could not coadapt and went extinct. In a spatial environment (e.g., a 2-D lattice), sets of cooperating polymers evolved, interacting in a hypercyclic fashion. The authors also observed a *chemoton*-like (Gánti 1975) cooperative behavior, with spatially isolated, membrane bounded evolutionary stable molecular organizations.

### Machines with Fixed Tape Size

In order to perform large-scale systematic simulation studies like the investigation of noise (Ikegami and Hashimoto 1995) or intrinsic evolution (Dittrich and Banzhaf 1998), it makes sense to limit the study to molecules of fixed tractable length. Ikegami and Hashimoto (1995) developed an abstract artificial chemistry with two types of molecular species: 7 bit long tapes, which are mapped to 16 bit long machines. Tapes and machines form two separated populations, simulated in parallel. Reactions take place between a tape and a machine according to the following reaction scheme:



A machine  $s_M$  can react with a tape  $s_T$ , if its head matches a substring of the tape  $s_T$  and its tail matches a different substring of the tape  $s_T$ . The machine operates only between these two substrings (called reading frame) which results in a tape  $s'_T$ . The tape  $s'_T$  is then “translated” (folded) into a machine  $s'_M$ .

Ikegami and Hashimoto (1995) showed that under the influence of low noise, simple autocatalytic loops are formed. When the noise level is increased, the reaction network is destabilized by parasites, but after a relatively long transient phase (about 1000 generations) a very stable, dense reaction network appears, called **core network** (Ikegami and Hashimoto 1995). A core network maintains its relatively high diversity even if the noise is deactivated. The active mutation rate is high. When the noise level is very high, only small, degenerated core networks emerge with a low diversity and very low (even no) active mutation. The core networks which emerged under the influence of external noise are very stable so that their is no further development after they have appeared.

### Assembler Automata

An **Assembler automaton** is like a parallel von Neumann machine. It consists of a core memory and a set of processing units running in parallel. Inspired by *Core Wars* (Dewdney 1984), assembler automata have been used to create certain

artificial life systems, such as *Coreworld* (Rasmussen et al. 1990, 1992), *Tierra* (Ray 1992), *Avida* (Adami and Brown 1994), and *Amoeba* (Pargellis 1996). Although these machines have been classified as artificial chemistries (Rasmussen et al. 1990), it is in general difficult to identify molecules or reactions. Furthermore, the assembler automaton *Tierra* has explicitly been designed to imitate the Cambrian explosion and *not* a chemical process. Nevertheless, in some cases we can interpret the execution of an assembler automaton as a chemical process, which is especially possible in a clear way in experiments with *Avida*. Here, a molecule is a single assembler program, which is protected by a memory management system. The system is initialized with manually written programs that are able to self replicate. Therefore, in a basic version of *Avida* only unimolecular first order reactions occur, which are of replicator type. The reaction scheme can be written as  $a \rightarrow a + \text{mutation}(a)$ . The function *mutation* represents the possible errors that can occur while the program is self replicating.

### Lattice Molecular Systems

In this section systems are discussed which consist of a regular lattice, where each lattice site can hold a part (e.g. atom) of a molecule. Between parts, bonds can be formed so that a molecule covers many lattice sites. This is different to systems where a lattice site holds a complete molecule, like in *Avida*. The important difference is that in lattice molecular systems, the space of the molecular structure is identical to the space in which the molecules are floating around. In systems where a molecule covers just one lattice site, the molecular structure is described in a different space independently from the space in which the molecule is located.

Lattice molecular systems have been intensively used to model polymers (Vanderzande 1998), protein folding (Socci and Onuchic 1995), and RNA structures. Besides approaches which intend to model real molecular dynamics as accurately as possible, there are also approaches which try to build abstract models. These models should give insight into statistical properties of

polymers like their energy landscape and folding processes (Sali et al. 1994a, b), but should not give insight into questions concerning origin and evolution self maintaining organizations or molecular information processing. For these questions more abstract systems are studied as described in the following.

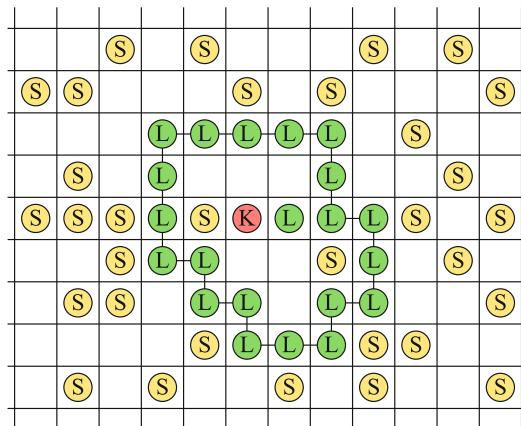
Varela, Maturana, and Uribe introduced in (Varela et al. 1974) a lattice molecular system to illustrate their concept of autopoiesis (cf. McMullin and Varela 1997; Zeleny 1977). The system consists of a 2-D square lattice. Each lattice site can be occupied by one of the following atoms: Substrate *S*, catalyst *K*, and monomer *L*. Atoms may form bonds and thus form molecular structures on the lattice. If molecules come close to each other they may react according to the following reaction rules:

$K + 2S \rightarrow K + L$	(1) Composition: Formation of a monomer.
$\dots - L -$ $L + L \rightarrow \dots - L -$ $L - L$	(2) Concatenation:
$L + L \rightarrow L - L$	Formation of a bond between a monomer and another monomer with no more than one bond. This reaction is inhibited by a double-bounded monomer (McMullin and Varela 1997).
$L \rightarrow 2S$	(3) Disintegration: Decomposition of a monomer

Figure 3 illustrates an autopoietic entity that may arise. Note that it is quite difficult to find the right dynamical conditions under which such autopoietic structures are stable (McMullin and Varela 1997; Ono and Ikegami 2000).

### Cellular Automata

Cellular automata (Mathematical Basis of Cellular Automata, Introduction to) can be used as a medium to simulate chemical like processes in various ways. An obvious approach is to use the cellular automata to model space where each cell can hold a molecule (like in *Avida*) or atom (like in lattice molecular automata). However there are approaches where it is not clear at the onset what a molecule or reaction is. The model specification



**Artificial Chemistry, Fig. 3** Illustration of a lattice molecular automaton that contains an autopoietic entity. Its membrane is formed by a chain of L monomers and encloses a catalyst K. Only substrate S may diffuse through the membrane. Substrate inside is catalyzed by K to form free monomers. If the membrane is damaged by disintegration of a monomer L it can be quickly repaired by a free monomer. See (McMullin and Varela 1997; Varela et al. 1974) for details

does not contain any notion of a molecule or reaction, so that an observer has to identify them. Molecules can be equated with self propagating patterns like gliders, self reproducing loops (Langton 1984; Sayama 1998), or with the moving boundary between two homogeneous domains (Hordijk et al. 1996). Note that in the latter case, particles become visible as space-time structures, which are defined by boundaries and not by connected set of cells with specific states. An interaction between these boundaries is then interpreted as a reaction.

### Mechanical Artificial Chemistries

There are also physical systems that can be regarded as artificial chemistries. Hosokawa et al. (1994) presented a mechanical self assembly system consisting of triangular shapes that form bonds by permanent magnets. Interpreting attachment and detachment as chemical reactions, Hosokawa et al. (1994) derived a chemical differential equation (subsection “[The Chemical Differential Equation](#)”) modeling the kinetics of the structures appearing in the system. Another approach uses millimeter sized magnetic disks at

a liquid air interface, subject to a magnetic field produced by a rotating permanent magnet (Grzybowski et al. 2000). These magnetic discs exhibit various types of structures, which might even interact resembling chemical reactions. The important difference to the first approach is that the rotating magnetic discs form dissipative structures, which require continuous energy supply (Fig. 4).

### Semi Realistic Molecular Structures and Graph Rewriting

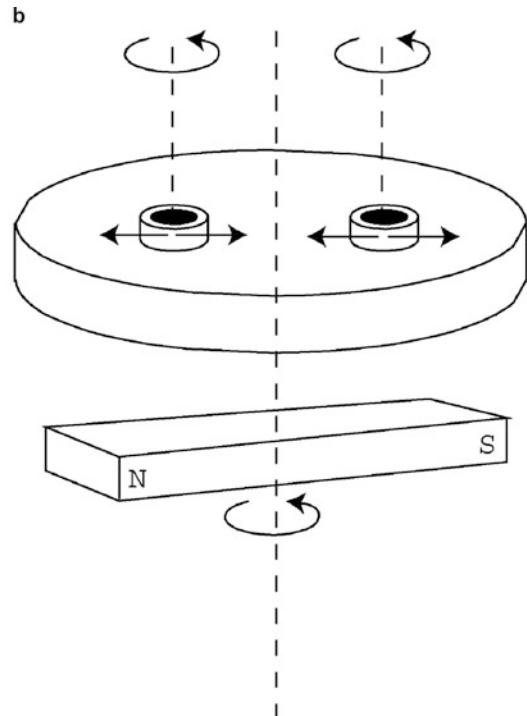
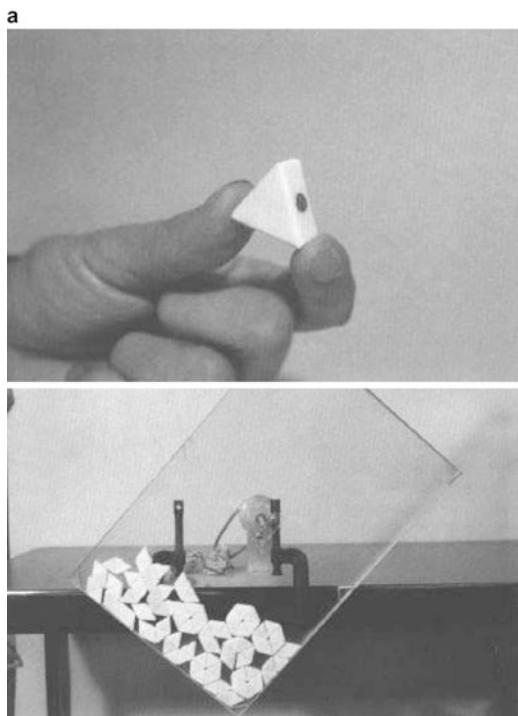
Recent development in artificial chemistry aims at more realistic molecular structures and reactions, like *oo chemistry* by Bersini (2000) (see also Lenaerts and Bersini 2009) or *toy chemistry* (Benkő et al. 2003). These approaches apply graph rewriting, which is a powerful and quite universal approach for defining transformations of graph-like structures (Kniemeyer et al. 2004). In toy chemistry, molecules are represented as labeled graphs; i.e., by their structural formulas; their basic properties are derived by a simplified version of the extended Hückel MO theory that operates directly on the graphs; chemical reaction mechanisms are implemented as graph rewriting rules acting on the structural formulas; reactivities and selectivities are modeled by a variant of the frontier molecular orbital theory based on the extended Hückel scheme. Figure 5 shows an example of a graph rewriting rule for unimolecular reactions.

### Space

Many approaches discussed above assume that the reaction vessel is well stirred. However, especially in living systems, space plays an important role. Cells, for example, are not a bag of enzymes, but spatially highly structured. Moreover, it is assumed that space has played an important role in the origin of life.

### Techniques for Modeling Space

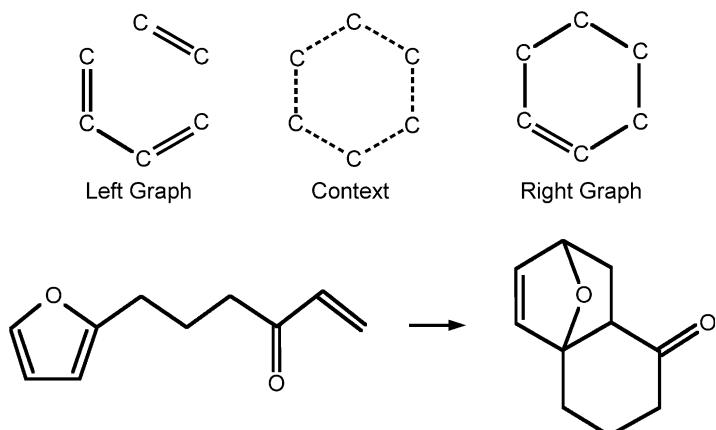
In chemistry, space is usually modeled by assuming an Euclidean space (partial differential equation or particle based) or by compartments, which



**Artificial Chemistry, Fig. 4** Example of mechanical artificial chemistries. (a) Self assembling magnetic tiles by Hosokawa et al. (1994). (b) Rotating magnetic discs by

Grzybowski et al. (2000). (Figures taken (and modified) from Grzybowski et al. (2000) and Hosokawa et al. (1994))

**Artificial Chemistry, Fig. 5** An example of a graph rewriting rule (top) and its application to the synthesis of a bridge ring system (bottom). (Figure from Benkő et al. (2003))



we can find also in many artificial chemistries (Furusawa and Kaneko 1998; Păun 2000). However, some artificial chemistries use more abstract, “computer friendly” spaces, such as a core memory like in *Tierra* (Ray 1992) or a planar triangular graph (Speroni di Fenizio 2000). There are

systems like MGS (Giavitto and Michel 2001) that allow to specify various topological structures easily. Approaches like P-systems and membrane computing (Păun 2000) allow one even to change the spatial structure dynamically (► **Membrane Computing: Power and Complexity**).

## Phenomena in Space

In general, space delays the extinction of species by fitter species, which is for example, used in Avida to obtain more complex evolutionary patterns (Adami and Brown 1994). Space leads usually to higher diversity. Moreover, systems that are unstable in a well stirred reaction vessel can become stable in a spatial situation. An example is the hypercycle (Eigen and Schuster 1977), which stabilizes against parasites when space is introduced (Boerlijst and Hogeweg 1991). Conversely, space can destabilize an equilibrium, leading to symmetry breaking, which has been suggested as an important mechanism underlying morphogenesis (Turing 1952). Space can support the co existence of chemical species, which would not co exist in a well stirred system (Kirner et al. 1999). Space is also necessary for the formation of auto-poietic structures (Varela et al. 1974) and the formation of units that undergo Darwinian evolution (Fernando and Rowe 2007).

## Theory

There is a large body of theory from domains like chemical reaction system theory (Érdi and Tóth 1989), Petri net theory (Petri 1962), and rewriting system theory (e.g., P-systems (Păun 2000)), which applies also to artificial chemistries. In this section, however, we shall investigate in more detail those theoretical concepts which have emerged in artificial chemistry research.

When working with complex artificial chemistries, we are usually more interested in the qualitative than quantitative nature of the dynamics. That is, we study how the set of molecular species present in the reaction vessel changes over time, rather than studying a more detailed trajectory in concentration space.

The most prominent qualitative concept is the autocatalytic set, which has been proposed as an important element in the origin of life (Eigen 1971; Kauffman 1971; Rössler 1971). An **autocatalytic set** is a set of molecular species where each species is produced by at least one catalytic reaction within the set (Jain and Krishna 1998; Kauffman 1986) (Fig. 1). This property has also

been called self maintaining (Fontana and Buss 1994) or (catalytic) closure (Kauffman 1993). Formally: A set of species  $A \subseteq \mathcal{M}$  is called an autocatalytic set (or sometimes, catalytically closed set), if for all species  $a \in A$  there is a catalyst  $a' \in A$ , and a reaction  $\rho \in \mathcal{R}$  such that  $a'$  is catalyst in  $\rho$  (i.e.,  $a' \in \text{LHS}(\rho)$  and  $a' \in \text{RHS}(\rho)$ ) and  $\rho$  can take place in  $A$  (i.e.,  $\text{LHS}(\rho) \subseteq A$ ) (For general reaction systems, this definition has to be refined. When  $A$  contains species that are part of the inflow, like  $a$  and  $b$  in Fig. 1, but which are not produced in a catalytic way, we might want them to be part of an autocatalytic set. Assume, for example, the set  $A = \{a, b, aa, ba\}$  from Fig. 1, where  $aa$  catalyzes the production of  $aa$  and  $ba$ , while using up “substrate”  $a$  and  $b$ , which are not catalytically produced.).

### Example 1 (autocatalytic sets)

$\mathcal{R} = \{a \rightarrow 2a, a \rightarrow a + b, a \rightarrow , b \rightarrow\}$ . Molecule  $a$  catalyzes its own production and the production of  $b$ . Both molecules are subject to a dilution flow (or, equivalently, spontaneous decay), which is usually assumed in ACs studying autocatalytic sets. In this example there are three autocatalytic sets: two non trivial autocatalytic sets  $\{a\}$  and  $\{a, b\}$ , and the empty set  $\{\}$ , which is from a mathematical point of view also an autocatalytic set.

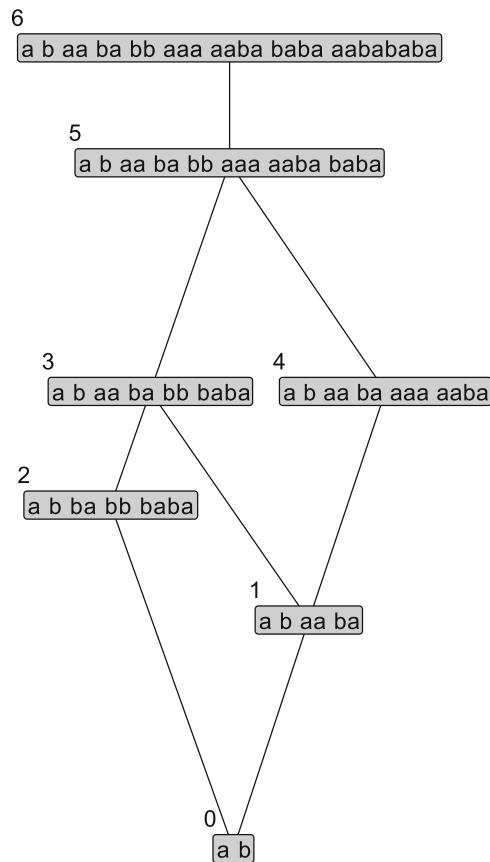
The term “autocatalytic set” makes sense only in ACs where catalysis is possible, it is not useful when applied to arbitrary reaction networks. For this reason Dittrich and Speroni di Fenizio (Dittrich and Speroni di Fenizio 2007) introduced a general notion of self maintenance, which includes the autocatalytic sets as a special case: Formally, given a reaction network  $\langle \mathcal{M}, \mathcal{R} \rangle$  with  $m = |\mathcal{M}|$  molecules and  $r = |\mathcal{R}|$  reactions, and let  $\mathbf{S} = (s_{a,\rho})$  be the  $(m \times r)$  stoichiometric matrix implied by the reaction rules  $\mathcal{R}$ , where  $s_{a,\rho}$  denotes the number of molecules of type  $a$  produced in reaction  $\rho$ . A set of molecules  $C \subseteq \mathcal{M}$  is called **self maintaining**, if there exists a flux vector  $\mathbf{v} \in \mathbb{R}^r$  such that the following three conditions apply: (1) for all reactions  $\rho$  that can take place in  $C$  (i.e.,  $\text{LHS}(\rho) \subseteq C$ ) the flux  $v_\rho > 0$ ; (2) for all remaining reactions  $\rho$  (i.e.,  $\text{LHS}(\rho) \subseteq C$ ),

the flux  $v_\rho = 0$ ; and (3) for all molecules  $a \in C$ , the production rate  $(\mathbf{Sv})_a \geq 0$ .  $v_\rho$  denotes the element of  $\mathbf{v}$  describing the flux (i.e. rate) of reaction  $\rho$ .  $(\mathbf{Sv})_a$  is the production rate of molecule  $a$  given flux vector  $\mathbf{v}$ . In Example 1 there are three self maintaining (autocatalytic) sets.

Interestingly, there are self maintaining sets that cannot make up a stationary state. In our Example 1 only two of the three self maintaining sets are species combinations that can make up a stationary state (i.e., a state  $\mathbf{x}^0$  for which  $0 = \mathbf{Sv}(\mathbf{x}^0)$  holds). The self maintaining set  $\{a\}$  cannot make up a stationary state because  $a$  generates  $b$  through the reaction  $a \rightarrow a + b$ . Thus, there is no stationary state (of Eq. (3) with Assumptions 1 and 2) in which the concentration of  $a$  is positive while the concentration of  $b$  is zero. In order to filter out those less interesting self maintaining sets, Fontana and Buss (1994) introduced a concept taken from mathematics called closure: Formally, a set of species  $A \subseteq \mathcal{M}$  is **closed**, if for all reactions  $\rho$  with  $\text{LHS}(\rho) \subseteq A$  (the reactions that can take place in  $A$ ), the products are also contained in  $A$ , i.e.,  $\text{RHS}(\rho) \subseteq A$ .

Closure and self maintenance lead to the important concept of a chemical organization (Dittrich and Speroni di Fenizio 2007; Fontana and Buss 1994): Given an arbitrary reaction network  $\langle \mathcal{M}, \mathcal{R} \rangle$ , a set of molecular species that is closed and self maintaining is called an **organization**. The importance of an organization is illustrated by a theorem roughly saying that given a fixed point of the chemical ODE Eq. (3), then the species with positive concentrations form an organization (Dittrich and Speroni di Fenizio 2007). In other words, we have only to check those species combinations for stationary states that are organizations. The set of all organizations can be visualized nicely by a Hasse diagram, which sketches the hierarchical (organizational) structure of the reaction network (Fig. 6). The dynamics of the artificial chemistry can then be explained within this Hasse diagram as a movement from organization to organization (Matsumaru et al. 2006; Speroni Di Fenizio and Dittrich 2002).

Note that in systems under flow condition, the (finite) set of all organizations forms an algebraic



**Artificial Chemistry, Fig. 6** Lattice of organizations of the autocatalytic network shown in Fig. 1. An organization is a closed and self maintaining set of species. Two organizations are connected by a line, if one is contained in the other and there is no organization in between. The vertical position of an organization is determined by the number of species it contains

lattice (Dittrich and Speroni di Fenizio 2007). That is, given two organizations, there is always a unique organization union and organization intersection.

Although the concept of **autopoiesis** (Varela et al. 1974) has been described informally in quite some detail, a stringent formal definition is lacking. In a way, we can interpret the formal concepts above as an attempt to approach necessary properties of an autopoietic system step by step by precise formal means. Obviously, being (contained in) at least one organization is a necessary condition for a chemical autopoietic system. But it is not sufficient. Missing are the notion of

robustness and a spatial concept that formalizes a system's ability to maintain (and self create) its own identity, e.g., through maintaining a membrane (Fig. 3).

## Evolution

With artificial chemistries we can study chemical evolution. Chemical evolution (also called “prebiotic evolution”) describes the first step in the development of life, such as the formation of complex organic molecules from simpler (in-) organic compounds (Maynard Smith and Szathmáry 1995). Because there are no prebiotic fossils, the study of chemical evolution has to rely on experiments (Lazcano and Bada 2003; Miller 1953) and theoretic (simulation) models (Eigen 1971). Artificial chemistries aim at capturing the constructive nature of these chemical systems and try to reproduce their evolution in computer simulations. The approaches can be distinguished by whether the evolution is driven by external operators like mutation, or whether variation and selection appears intrinsically through the chemical dynamics itself.

### Extrinsic Evolution

In extrinsic approaches, an external variation operator changes the reaction network by adding, removing, or manipulating a reaction, which may also lead to the addition or removal of chemical species. In this approach, a molecule does not need to possess a structure. The following example by Jain and Krishna (1998) shows that this approach allows one to create an evolving system quite elegantly:

Let us assume that the reaction network consists of  $m$  species  $\mathcal{M} = \{1, \dots, m\}$ . There are first-order catalytic reaction rules of the form  $(i \rightarrow i + j) \in \mathcal{R}$  and a general dilution flow  $(a \rightarrow) \in \mathcal{R}$  for all species  $a \in \mathcal{M}$ . The reaction network is completely described by a directed graph represented by the adjacency matrix  $C = (c_{i,j})$ ,  $i, j \in \mathcal{M}$ ,  $c_{i,j} \in \{0, 1\}$ , with  $c_{i,j} = 1$  if molecule  $j$  catalyzes the production of  $i$ , and  $c_{i,j} = 0$  otherwise. In order to avoid that self replicators dominate the

system, Jain and Krishna assume  $c_{i,i} = 0$  for all molecules  $i \in \mathcal{M}$ .

At the beginning, the reaction network is randomly initialized, that is, (for  $i \neq j$ )  $c_{i,j} = 1$  with probability  $p$  and  $c_{i,j} = 0$ , otherwise. In order to simulate the dynamics, we assume a population of molecules represented by the concentration vector  $\mathbf{x} = (x_1, \dots, x_m)$ , where  $x_i$  represents the current concentration of species  $i$ . The whole system is simulated in the following way:

Step 1: Simulate the chemical differential equation

$$\dot{x}_i = \sum_{j \in \mathcal{M}} c_{j,i} - x_i \sum_{k,j \in \mathcal{M}} c_{k,j} x_j, \quad (18)$$

until a steady state is reached. Note that this steady state is usually independent of the initial concentrations, cf. (Stadler et al. 1993).

Step 2: Select the “mutating” species  $i$ , which is the species with smallest concentration in the steady state.

Step 3: Update the reaction network by replacing the mutating species by a new species, which is created randomly in the same way as the initial species. That is, the entries of the  $i$ th row and  $i$ th column of the adjacency matrix  $C$  are replaced by randomly chosen entries with the same probability  $p$  as during initialization.

Step 4: Go to Step 1.

Note that there are two explicitly simulated time scales: a slow, evolutionary time scale at which the reaction network evolves (Step 2 and 3), and a fast time scale at which the molecules catalytically react (Step 1).

In this model we can observe how an autocatalytic set inevitably appears after a period of disorder. After its arrival the largest autocatalytic set increases rapidly its connectivity until it spans the whole network. Subsequently, the connectivity converges to a steady state determined by the rate of external perturbations. The resulting highly non random network is not fully stable, so that we can also study the causes for crashes and recoveries. For example, Jain and Krishna (2002) identified that

in the absence of large external perturbation, the appearance of a new *viable* species is a major cause of large extinction and recoveries. Furthermore, crashes can be caused by extinction of a “keystone species”. Note that for these observations, the new species created in Step 3 do not need to inherit any information from other species.

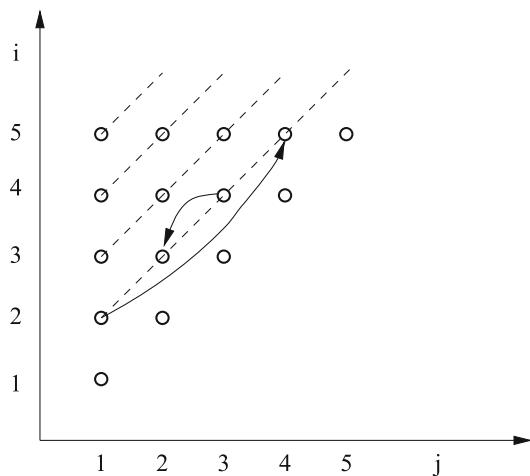
### Intrinsic Evolution

Evolutionary phenomena can also be caused by the intrinsic dynamics of the (artificial) chemistry. In this case, external operators like those mutating the reaction network are not required. As opposed to the previous approach, we can define the whole reaction network at the onset and let only the composition of molecules present in the reaction vessel evolve. The reaction rules are (usually) defined by a structure-to-function mapping similar to those described in section “[Structure-to Function Mapping](#)”. The advantage of this approach is that we need not define an external operator changing the reaction network’s topology. Furthermore, the evolution can be more realistic because, for example, molecular species that have vanished can reenter at a later time, which does not happen in an approach like the one described previously. Also, when we have a structure-to-function mapping, we can study how the structure of the molecules is related to the emergence of chemical organizations.

There are various approaches using, for example, Turing machines (McCaskill 1988), lambda calculus (Fontana 1992; Fontana and Buss 1994), abstract automata (Dittrich and Banzhaf 1998), or combinatorics (Speroni di Fenizio 2000) for the structure-to function mapping.

In all of those approaches we can observe an important phenomenon: While the AC evolves autocatalytic sets or more general, chemical organizations become visible. Like in the system by Jain and Krishna this effect is indicated also by an increase of the connectivity of the species within the population.

When an organization becomes visible, the system has focused on a sub-space of the whole set of possible molecules. Those molecules, belonging to the emerged organization, posses usually relatively high concentrations, because

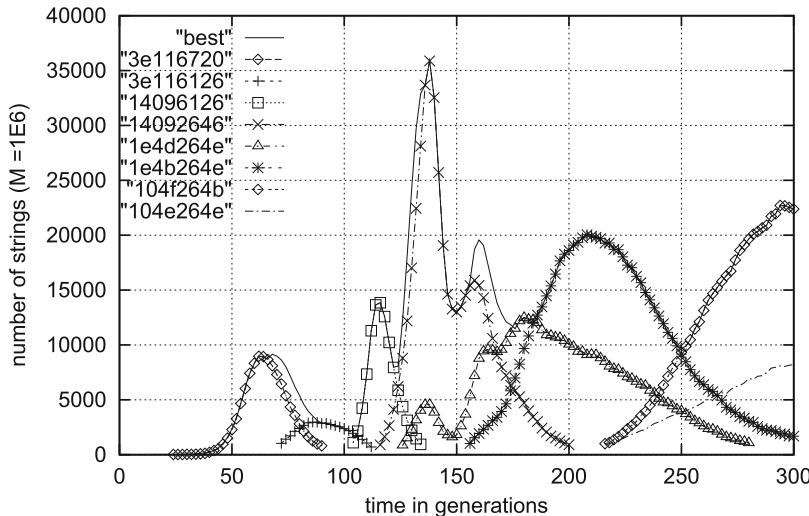


**Artificial Chemistry, Fig. 7** Illustration of syntactically and semantically closed organizations. Each organization consists of an infinite number of molecular species (connected by a dotted line). A circle represents a molecule having the structure:  $\lambda x_1. \lambda x_2. \dots 27$

they are generated by many reactions. The closure of this set is usually smaller than the universe  $\mathcal{M}$ . Depending on the setting the emerged organization can consist of a single self replicator, a small set of mutually producing molecules, or a large set of different molecules. Note that in the latter case the organization can be so large (and even infinite) that not all its molecules are present in the population (see Fig. 7 for an example). Nevertheless the population can carry the organization, if the system can keep a generating set of molecules within the population.

An emerged organization can be quite stable but can also change either by external perturbations like randomly inserted molecules or by internal fluctuations caused by reactions among molecules that are not part of the emerged organization, but which have remained in small quantities in the population, cf. for an example (Matsumaru et al. 2006).

Dittrich and Banzhaf (1998) have shown that it is even possible to obtain evolutionary behavior without any externally generated variation, that is, even without any inflow of random molecules. And without any explicitly defined fitness function or selection process. Selection emerges as a result of the dilution flow and the limited population size (Fig. 8).



**Artificial Chemistry, Fig. 8** Example of a self evolving artificial chemistry. The figure shows the concentration of some selected species of a reaction vessel containing approximately  $10^4$  different species. In this example, molecules are binary strings of length 32 bit. Two binary strings react by mapping one of them to a finite state machine operating on the second binary string. There is

only a general, non selective dilution flow. No other operators like mutation, variation, or selection are applied. Note that the structure of a new species tends to be similar to the structure of the species it has been created from. The dynamics simulated by the algorithm of the introduction. Population size  $k = 10^6$  molecules. (Figure from Dittrich and Banzhaf (1998))

### Syntactic and Semantic Closure

In many “constructive” implicitly defined artificial chemistries we can observe species appearing that share syntactical and functional similarities that are invariant under the reaction operation. Fontana and Buss (1994) called this phenomenon syntactic and semantic closure.

Syntactic closure refers to the observation that the molecules within an emerged organization  $O \subset \mathcal{M}$  are structurally similar. That is, they share certain structural features. This allows one to describe the set of molecules  $O$  by a formal language or grammar in a compact way. If we would instead pick a subset  $A$  from  $\mathcal{M}$  randomly, the expected length of  $A$ 's description is on the order of its size. Assume, for example, if we pick one million strings of length 100 randomly from the set of all strings of length 100, then we would need about 100 million characters to describe the resulting set. Interestingly, the organizations that appear in implicitly defined ACs can be described much more compactly by a grammar.

Syntactic and semantic closure should be illustrated with an example taken from (Fontana and Buss 1994), where  $O \subset \mathcal{M}$  is even infinite in size.

In particular experiments with the lambda chemistry, molecules appeared that posses the following structure:

$$A_{i,j} \equiv \lambda x_1.\lambda x_2.\dots\lambda x_i.x_j \text{ with } j \leq i, \quad (19)$$

for example  $\lambda x_1.\lambda x_2.\lambda x_3.x_2$ . Syntactical closure means that we can specify such structural rules specifying the molecules of  $O$  and that these structural rules are invariant under the reaction mechanism. If molecules with the structure given by Eq. (19) react, their product can also be described by Eq. (19).

Semantic closure means that we can describe the reactions taking place within  $O$  by referring to the molecule's grammatical structure (e.g. Eq. 19). In our example, all reactions within  $O$  can be described by the following laws (illustrated by Fig. 7):

$$\begin{aligned} \forall i, j > 1, k, l : A_{i,j} + A_{k,l} &\Rightarrow A_{i-1,j-1}, \\ \forall i > 1, j = 1, k, l : A_{i,j} + A_{k,l} &\Rightarrow A_{k+i-1,l+i-1}. \end{aligned} \quad (20)$$

For example  $\lambda x_1.x_2.\lambda x_3.x_4.x_3 + \lambda x_1.x_2.x_2 \Rightarrow \lambda x_1.x_2.\lambda x_3.x_2$  and  $\lambda x_1.x_2.x_1 + \lambda x_1.x_2.x_2$ .

$\lambda x_3.\lambda x_4.x_3 \Rightarrow \lambda x_1.\lambda x_2.\lambda x_3.\lambda x_4.\lambda x_5.x_4$ , respectively (Fig. 7). As a result of the semantic closure we need not to refer to the underlying reaction mechanism (e.g., the lambda calculus). Instead, we can explain the reactions within  $O$  on a more abstract level by referring to the grammatical structure (e.g., Eq. 19).

Note that syntactical and semantical closure appears also in real chemical system and is exploited by chemistry to organize chemical explanations. It might even be stated that without this phenomenon, chemistry as we know it would not be possible.

## Information Processing

The chemical metaphor gives rise to a variety of computing concepts, which are explained in detail in further articles of this encyclopedia. In approaches like amorphous computing ([► Amorphous Computing](#)) chemical mechanisms are used as a subsystem for communication between a huge number of simple, spatially distributed processing units. In membrane computing ([► Membrane Computing: Power and Complexity](#)), rewriting operations are not only used to react molecules, but also to model active transport between compartments and to change the spatial compartment structure itself.

Beside these theoretical and *in silico* artificial chemistries there are approaches that aim at using real molecules: Spatial patterns like waves and solitons in excitable media can be exploited to compute, see “[► Reaction-Diffusion Computing](#)” and “[► Computing in Geometrical Constrained Excitable Chemical Systems](#)”. Whereas other approaches like conformational computing and DNA computing ([► DNA Computing](#)) do not rely on spatially structured reaction vessels. Other closely related topics are molecular automata ([► Molecular Automata](#)) and bacterial computing ([► Bacterial Computing](#)).

## Future Directions

Currently, we can observe a convergence of artificial chemistries and chemical models towards

more realistic artificial chemistries (Benkő et al. 2003). On the one hand, models of systems biology are inspired by techniques from artificial chemistries, e.g., in the domain of rule-based modeling (Faeder et al. 2005; Hlavacek et al. 2006). On the other hand, artificial chemistries become more realistic, for example, by adopting more realistic molecular structures or using techniques from computational chemistry to calculate energies (Benkő et al. 2003; Lenaerts and Bersini 2009). Furthermore, the relation between artificial chemistries and real biological systems is made more and more explicit (Fernando et al. 2007; Kaneko 2007; Lenski et al. 1999).

In the future, novel theories and techniques have to be developed to handle complex, implicitly defined reaction systems. This development will especially be driven by the needs of system biology, when implicitly defined models (i.e., rule-based models) become more frequent.

Another open challenge lies in the creation of realistic (chemical) open-ended evolutionary systems. For example, an artificial chemistry with “true” open-ended evolution or the ability to show a satisfying long transient phase, which would be comparable to the natural process where novelties continuously appear, has not been presented yet. The reason for this might be lacking computing power, insufficient manpower to implement what we know, or missing knowledge concerning the fundamental mechanism of (chemical) evolution. Artificial chemistries could provide a powerful platform to study whether the mechanisms that we believe explain (chemical) evolution are indeed candidates.

As sketched above, there is a broad range of currently explored practical application domains of artificial chemistries in technical artifacts, such as ambient computing, amorphous computing, organic computing, or smart materials. And eventually, artificial chemistry might come back to the realm of real chemistry and inspire the design of novel computing reaction systems like in the field of molecular computing, molecular communication, bacterial computing, or synthetic biology.

## Bibliography

### Primary Literature

- Adami C, Brown CT (1994) Evolutionary learning in the 2D artificial life system *avida*. In: Brooks RA, Maes P (eds) *Proc artificial life IV*. MIT Press, Cambridge, pp 377–381 ISBN 0-262-52190-3
- Adleman LM (1994) Molecular computation of solutions to combinatorial problems. *Science* 266:1021
- Bagley RJ, Farmer JD (1992) Spontaneous emergence of a metabolism. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) *Artificial life II*. Addison Wesley, Redwood City, pp 93–140 ISBN 0-201-52570-4
- Banâtre J-P, Métayer DL (1986) A new computational model and its discipline of programming. Technical Report RR-0566. INRIA, Rennes
- Banzhaf W (1993) Self replicating sequences of binary numbers – foundations I and II: general and strings of length  $n = 4$ . *Biol Cybern* 69:269–281
- Banzhaf W (1994) Self replicating sequences of binary numbers: the build-up of complexity. *Complex Syst* 8:215–225
- Banzhaf W (1995) Self organizing algorithms derived from RNA interactions. In: Banzhaf W, Eeckman FH (eds) *Evolution and biocomputing, LNCS*, vol 899. Springer, Berlin, pp 69–103
- Banzhaf W, Dittrich P, Rauhe H (1996) Emergent computation by catalytic reactions. *Nanotechnology* 7(1): 307–314
- Benkő G, Flamm C, Stadler PF (2003) A graph-based toy model of chemistry. *J Chem Inf Comput Sci* 43(4):1085–1093. <https://doi.org/10.1021/ci0200570>
- Bersini H (2000) Reaction mechanisms in the oo chemistry. In: Bedau MA, JS MC, Packard NH, Rasmussen S (eds) *Artificial life VII*. MIT Press, Cambridge, pp 39–48
- Boerlijst MC, Hogeweg P (1991) Spiral wave structure in prebiotic evolution: hypercycles stable against parasites. *Physica D* 48(1):17–28
- Breyer J, Ackermann J, McCaskill J (1999) Evolving reaction diffusion ecosystems with self assembling structure in thin films. *Artif Life* 4(1):25–40
- Conrad M (1992) Molecular computing: The key-lock paradigm. *Computer* 25:11–22
- Dewdney AK (1984) In the game called core war hostile programs engage in a battle of bits. *Sci Am* 250:14–22
- Dittrich P (2001) On artificial chemistries. PhD thesis, University of Dortmund
- Dittrich P, Banzhaf W (1998) Self evolution in a constructive binary string system. *Artif Life* 4(2):203–220
- Dittrich P, Speroni di Fenizio P (2007) Chemical organization theory. *Bull Math Biol* 69(4):1199–1231. <https://doi.org/10.1007/s11538-006-9130-8>
- Ehricht R, Ellinger T, McCascill JS (1997) Cooperative amplification of templates by cross hybridization (CATCH). *Eur J Biochem* 243(1/2):358–364
- Eigen M (1971) Selforganization of matter and the evolution of biological macromolecules. *Naturwissenschaften* 58(10):465–523
- Eigen M, Schuster P (1977) The hypercycle: a principle of natural self organisation, part A. *Naturwissenschaften* 64(11):541–565
- Érdi P, Tóth J (1989) Mathematical models of chemical reactions: theory and applications of deterministic and stochastic models. Princeton University Press, Princeton
- Faeder JR, Blinov ML, Goldstein B, Hlavacek WS (2005) Rule-based modeling of biochemical networks. *Complexity*. <https://doi.org/10.1002/cplx.20074>
- Farmer JD, Kauffman SA, Packard NH (1986) Autocatalytic replication of polymers. *Physica D* 22:50–67
- Fernando C, Rowe J (2007) Natural selection in chemical evolution. *J Theor Biol* 247(1):152–167. <https://doi.org/10.1016/j.jtbi.2007.01.028>
- Fernando C, von Kiedrowski G, Szathmáry E (2007) A stochastic model of nonenzymatic nucleic acid replication: elongators sequester replicators. *J Mol Evol* 64(5):572–585. <https://doi.org/10.1007/s00239-006-0218-4>
- Fontana W (1992) Algorithmic chemistry. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) *Artificial life II*. Addison Wesley, Redwood City, pp 159–210
- Fontana W, Buss LW (1994) The arrival of the fittest: toward a theory of biological organization. *Bull Math Biol* 56:1–64
- Fontana W, Buss LW (1996) The barrier of objects: from dynamical systems to bounded organization. In: Casti J, Karlqvist A (eds) *Boundaries and barriers*. Addison Wesley, Redwood City, pp 56–116
- Furusawa C, Kaneko K (1998) Emergence of multicellular organisms with dynamic differentiation and spatial pattern. *Artif Life* 4:79–93
- Gánti T (1975) Organization of chemical reactions into dividing and metabolizing units: the chemotons. *Biosystems* 7(1):15–21
- Giavitto J-L, Michel O (2001) MGS: a rule-based programming language for complex objects and collections. *Electron Note Theor Comput Sci* 59(4):286–304
- Gillespie DT (1976) General method for numerically simulating stochastic time evolution of coupled chemical reaction. *J Comput Phys* 22(4):403–434
- Grzybowski BA, Stone HA, Whitesides GM (2000) Dynamic self assembly of magnetized, millimetre sized objects rotating at a liquid air interface. *Nature* 405(6790):1033–1036
- Hlavacek W, Faeder J, Blinov M, Posner R, Hucka M, Fontana W (2006) Rules for modeling signal transduction systems. *Sci STKE* 2006:re6
- Hofbauer J, Sigmund K (1988) *Dynamical systems and the theory of evolution*. University Press, Cambridge
- Hofstadter DR (1979) Gödel, Escher, Bach: An eternal golden braid. Basic Books, New York ISBN 0-465-02685-0
- Hordijk W, Crutchfield JP, Mitchell M (1996) Embedded particle computation in evolved cellular automata. In: Toffoli T, Biafore M, Leão J (eds) *PhysComp96*. New England Complex Systems Institute, Cambridge, pp 153–158

- Hosokawa K, Shimoyama I, Miura H (1994) Dynamics of self assembling systems: Analogy with chemical kinetics. *Artif Life* 1(4):413–427
- Hutton TJ (2002) Evolvable self replicating molecules in an artificial chemistry. *Artif Life* 8(4):341–356
- Ikegami T, Hashimoto T (1995) Active mutation in self reproducing networks of machines and tapes. *Artif Life* 2(3):305–318
- Jain S, Krishna S (1998) Autocatalytic sets and the growth of complexity in an evolutionary model. *Phys Rev Lett* 81(25):5684–5687
- Jain S, Krishna S (1999) Emergence and growth of complex networks in adaptive systems. *Comput Phys Commun* 122:116–121
- Jain S, Krishna S (2001) A model for the emergence of cooperation, interdependence, and structure in evolving networks. *Proc Natl Acad Sci U S A* 98(2):543–547
- Jain S, Krishna S (2002) Large extinctions in an evolutionary model: The role of innovation and keystone species. *Proc Natl Acad Sci U S A* 99(4):2055–2060. <https://doi.org/10.1073/pnas.032618499>
- Kaneko K (2007) Life: An introduction to complex systems biology. Springer, Berlin
- Kauffman SA (1971) Cellular homeostasis, epigenesis and replication in randomly aggregated macromolecular systems. *J Cybern* 1:71–96
- Kauffman SA (1986) Autocatalytic sets of proteins. *J Theor Biol* 119:1–24
- Kauffman SA (1993) The origins of order: self organization and selection in evolution. Oxford University Press, New York
- Kirner T, Ackermann J, Ehricht R, McCaskill JS (1999) Complex patterns predicted in an *in vitro* experimental model system for the evolution of molecular cooperation. *Biophys Chem* 79(3):163–186
- Kniemeyer O, Buck Sorlin GH, Kurth W (2004) A graph grammar approach to artificial life. *Artif Life* 10(4): 413–431. <https://doi.org/10.1162/1064546041766451>
- Laing R (1972) Artificial organisms and autonomous cell rules. *J Cybern* 2(1):38–49
- Laing R (1975) Some alternative reproductive strategies in artificial molecular machines. *J Theor Biol* 54:63–84
- Laing R (1977) Automaton models of reproduction by self inspection. *J Theor Biol* 66:437–456
- Langton CG (1984) Self reproduction in cellular automata. *Physica D* 10D(1–2):135–144
- Langton CG (1989) Artificial life. In: Langton CG (ed) Proceedings of artificial life. Addison Wesley, Redwood City, pp 1–48
- Lazcano A, Bada JL (2003) The 1953 Stanley L. Miller experiment: Fifty years of prebiotic organic chemistry. *Orig Life Evol Biosph* 33(3):235–242
- Lenaerts T, Bersini H (2009) A synthon approach to artificial chemistry. *Artif Life* 9 (in press)
- Lenski RE, Ofria C, Collier TC, Adami C (1999) Genome complexity, robustness and genetic interactions in digital organisms. *Nature* 400(6745):661–664
- Lohn JD, Colombano S, Scargle J, Stassinopoulos D, Haith GL (1998) Evolution of catalytic reaction sets using genetic algorithms. In: Proceedings of the IEEE international conference on evolutionary computation. IEEE, New York, pp 487–492
- Lugowski MW (1989) Computational metabolism: Towards biological geometries for computing. In: Langton CG (ed) Artificial life. Addison Wesley, Redwood City, pp 341–368 ISBN 0-201-09346-4
- Matsumaru N, Speroni di Fenizio P, Centler F, Dittrich P (2006) On the evolution of chemical organizations. In: Artmann S, Dittrich P (eds) Proceedings of the 7th german workshop of artificial life. IOS Press, Amsterdam, pp 135–146
- Maynard Smith J, Szathmáry E (1995) The major transitions in evolution. Oxford University Press, New York
- McCaskill JS (1988) Polymer chemistry on tape: A computational model for emergent genetics. Internal report. MPI for Biophysical Chemistry, Göttingen
- McCaskill JS, Chorongiewski H, Mekelburg D, Tangen U, Gemm U (1994) Configurable computer hardware to simulate long-time self organization of biopolymers. *Ber Bunsenges Phys Chem* 98(9):1114–1114
- McMullin B, Varela FJ (1997) Rediscovering computational autopoiesis. In: Husbands P, Harvey I (eds) Fourth european conference on artificial life. MIT Press, Cambridge, pp 38–47
- Miller SL (1953) A production of amino acids under possible primitive earth conditions. *Science* 117(3046): 528–529
- Morris HC (1989) Typogenetics: A logic for artificial life. In: Langton CG (ed) Artif life. Addison Wesley, Redwood City, pp 341–368
- Ono N, Ikegami T (2000) Self maintenance and self reproduction in an abstract cell model. *J Theor Biol* 206(2):243–253
- Pargellis AN (1996) The spontaneous generation of digital “life”. *Physica D* 91(1–2):86–96
- Păun G (2000) Computing with membranes. *J Comput Syst Sci* 61(1):108–143
- Petri CA (1962) Kommunikation mit Automaten. PhD thesis, University of Bonn
- Rasmussen S, Knudsen C, Feldberg R, Hindsholm M (1990) The coreworld: Emergence and evolution of cooperative structures in a computational chemistry. *Physica D* 42:111–134
- Rasmussen S, Knudsen C, Feldberg R (1992) Dynamics of programmable matter. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) Artificial life II. Addison Wesley, Redwood City, pp 211–291 ISBN 0-201-52570-4
- Ray TS (1992) An approach to the synthesis of life. In: Langton CG, Taylor C, Farmer JD, Rasmussen S (eds) Artificial life II. Addison Wesley, Redwood City, pp 371–408
- Rössler OE (1971) A system theoretic model for biogenesis. *Z Naturforsch B* 26(8):741–746
- Sali A, Shakhnovich E, Karplus M (1994a) How does a protein fold? *Nature* 369(6477):248–251
- Sali A, Shakhnovich E, Karplus M (1994b) Kinetics of protein folding: A lattice model study of the

- requirements for folding to the native state. *J Mol Biol* 235(5):1614–1636
- Salzberg C (2007) A graph-based reflexive artificial chemistry. *Biosystems* 87(1):1–12
- Sayama H (2009) Swarm chemistry. *Artif Life* (in press)
- Sayama H (1998) Introduction of structural dissolution into Langton's self reproducing loop. In: Adami C, Belew R, Kitano H, Taylor C (eds) *Artificial life VI*. MIT Press, Cambridge, pp 114–122
- Segre D, Ben-Eli D, Lancet D (2000) Compositional genomes: prebiotic information transfer in mutually catalytic noncovalent assemblies. *Proc Natl Acad Sci U S A* 97(8):4112–4117
- Socci ND, Onuchic JN (1995) Folding kinetics of proteinlike heteropolymers. *J Chem Phys* 101(2): 1519–1528
- Speroni di Fenizio P (2000) A less abstract artificial chemistry. In: Bedau MA, McCaskill JS, Packard NH, Rasmussen S (eds) *Artificial life VII*. MIT Press, Cambridge, pp 49–53
- Speroni Di Fenizio P, Dittrich P (2002) Artificial chemistry's global dynamics. Movement in the lattice of organisation. *J Three Dimens Images* 16(4):160–163 ISSN 1342-2189
- Stadler PF, Fontana W, Miller JH (1993) Random catalytic reaction networks. *Physica D* 63:378–392
- Suzuki H (2007) Mathematical folding of node chains in a molecular network. *Biosystems* 87(2–3):125–135. <https://doi.org/10.1016/j.biosystems.2006.09.005>
- Suzuki K, Ikegami T (2006) Spatial pattern induced evolution of a self replicating loop network. *Artif Life* 12(4):461–485. <https://doi.org/10.1162/artl.2006.12.4.461>
- Suzuki Y, Tanaka H (1997) Symbolic chemical system based on abstract rewriting and its behavior pattern. *Artif Life Robotics* 1:211–219
- Tangen U, Schulte L, McCaskill JS (1997) A parallel hardware evolvable computer polyp. In: Pocek KL, Arnold J (eds) *IEEE symposium on FPGAs for custom computing machines*. IEEE Computer Society, Los Alamitos
- Thürk M (1993) Ein Modell zur Selbstorganisation von Automatenalgorithmen zum Studium molekularer Evolution. PhD thesis, Universität Jena
- Turing AM (1952) The chemical basis of morphogenesis. *Philos Trans R Soc Lond B* 237:37–72
- Vanderzande C (1998) Lattice models of polymers. Cambridge University Press, Cambridge
- Varela FJ, Maturana HR, Uribe R (1974) Autopoiesis: the organization of living systems. *Biosystems* 5(4): 187–196
- Vareto L (1993) Typogenetics: An artificial genetic system. *J Theor Biol* 160(2):185–205
- Vareto L (1998) Studying artificial life with a molecular automaton. *J Theor Biol* 193(2):257–285
- Vico G (1710) *De antiquissima Italorum sapientia ex linguae originibus eruenda libri tres*. Neapel
- von Neumann J, Burks A (eds) (1966) *The theory of self reproducing automata*. University of Illinois Press, Urbana
- Zauner K-P, Conrad M (1996) Simulating the interplay of structure, kinetics, and dynamics in complex biochemical networks. In: Hofestädt R, Lengauer T, Löffler M, Schomburg D (eds) *Computer science and biology GCB'96*. University of Leipzig, Leipzig, pp 336–338
- Zeleny M (1977) Self organization of living systems: A formal model of autopoiesis. *Int J General Sci* 4:13–28

## Books and Reviews

- Adami C (1998) *Introduction to artificial life*. Springer, New York
- Dittrich P, Ziegler J, Banzhaf W (2001) Artificial chemistries – a review. *Artif Life* 7(3):225–275
- Hofbauer J, Sigmund K (1998) *Evolutionary games and population dynamics*. Cambridge University Press, Cambridge



## Amorphous Computing

Hal Abelson, Jacob Beal and Gerald Jay Sussman  
Computer Science and Artificial Intelligence  
Laboratory, Massachusetts Institute of  
Technology, Cambridge, MA, USA

### Article Outline

Glossary  
Definition of the Subject  
Introduction  
The Amorphous Computing Model  
Programming Amorphous Systems  
Amorphous Computing Paradigms  
Primitives for Amorphous Computing  
Means of Combination and Abstraction  
Supporting Infrastructure and Services  
Lessons for Engineering  
Future Directions  
Bibliography

### Glossary

**Amorphous computer** A collection of computational particles dispersed irregularly on a surface or throughout a volume, where individual particles have no *a priori* knowledge of their positions or orientations.

**Computational particle** A (possibly faulty) individual device for an amorphous computer. Each particle has modest computing power and a modest amount of memory. The particles are not synchronized, although they are all capable of operating at similar speeds, since they are fabricated by the same process. All particles are programmed identically, although each particle has means for storing local state and for generating random numbers.

**Field** A function assigning a value to every particle in an amorphous computer.

**Gradient** A basic amorphous computing primitive that estimates the distance from each particle to the nearest particle designated as a source of the gradient.

### Definition of the Subject

The goal of amorphous computing is to identify organizational principles and create programming technologies for obtaining intentional, pre-specified behavior from the cooperation of myriad unreliable parts that are arranged in unknown, irregular, and time-varying ways. The heightened relevance of amorphous computing today stems from the emergence of new technologies that could serve as substrates for information processing systems of immense power at unprecedentedly low cost, if only we could master the challenge of programming them.

### Introduction

Even as the foundations of computer science were being laid, researchers could hardly help noticing the contrast between the robustness of natural organisms and the fragility of the new computing devices. As John von Neumann remarked in 1948 (von Neumann 1951):

With our artificial automata we are moving much more in the dark than nature appears to be with its organisms. We are, and apparently, at least at present, have to be much more ‘scared’ by the occurrence of an isolated error and by the malfunction which must be behind it. Our behavior is clearly that of overcaution, generated by ignorance.

Amorphous computing emerged as a field in the mid-1990s, from the convergence of three factors:

- Inspiration from the cellular automata models for fundamental physics (Codd 1968; Margolus 1988).
- Hope that understanding the robustness of biological development could both help overcome

the brittleness typical of computer systems and also illuminate the mechanisms of developmental biology.

- The prospect of nearly free computers in vast quantities.

### Micro Fabrication

One technology that has come to fruition over the past decade is micro-mechanical electronic component manufacture, which integrates logic circuits, micro-sensors, actuators, and communication on a single chip. Aggregates of these can be manufactured extremely inexpensively, provided that not all the chips need work correctly, and that there is no need to arrange the chips into precise geometrical configurations or to establish precise interconnections among them. A decade ago, researchers envisioned *smart dust* elements small enough to be borne on air currents to form clouds of communicating sensor particles (Kahn et al. 1999).

Airborne sensor clouds are still a dream, but networks of millimeter-scale particles are now commercially available for environmental monitoring applications (Dust Networks 2007). With low enough manufacturing costs, we could mix such particles into bulk materials to form coatings like “smart paint” that can sense data and communicate its actions to the outside world. A smart paint coating on a wall could sense vibrations, monitor the premises for intruders, or cancel noise. Bridges or buildings coated with smart paint could report on traffic and wind loads and monitor structural integrity. If the particles have actuators, then the paint could even heal small cracks by shifting the material around. Making the particles mobile opens up entire new classes of applications that are beginning to be explored by research in *swarm robotics* (McLurkin 2004) and *modular robotics* (De Rosa et al. 2006).

### Cellular Engineering

The second disruptive technology that motivates the study of amorphous computing is microbiology. Biological organisms have served as motivating metaphors for computing since the days of calculating engines, but it is only over the past decade that we have begun to see how biology could literally be a substrate for computing,

through the possibility of constructing digital-logic circuits within individual living cells. In one technology logic signals are represented not by electrical voltages and currents, but by concentrations of DNA binding proteins, and logic elements are realized as binding sites where proteins interact through promotion and repression. As a simple example, if A and B are proteins whose concentrations represent logic levels, then an “inverter” can be implemented in DNA as a genetic unit through which A serves as a repressor that blocks the production of B (Knight and Sussman 1998; Weiss and Knight 2000; Weiss 2001; Weiss et al. 2003). Since cells can reproduce themselves and obtain energy from their environment, the resulting information processing units could be manufactured in bulk at a very low cost.

There is beginning to take shape a technology of cellular engineering that can tailor-make programmable cells to function as sensors or delivery vehicles for pharmaceuticals, or even as chemical factories for the assembly of nanoscale structures. Researchers in this emerging field of *synthetic biology* are starting to assemble registries of standard logical components implemented in DNA that can be inserted into *E. coli* (Registry of standard 2007). The components have been engineered to permit standard means of combining them, so that biological logic designers can assemble circuits in a mix-and-match way, similar to how electrical logic designers create circuits from standard TTL parts. There’s even an *International Genetically Engineered Machine Competition* where student teams from universities around the world compete to create novel biological devices from parts in the registry (igem 2006).

Either of these technologies—microfabricated particles or engineered cells—provides a path to cheaply fabricate aggregates of massive numbers of computing elements. But harnessing these for computing is quite a different matter, because the aggregates are unstructured. Digital computers have always been constructed to behave as precise arrangements of reliable parts, and almost all techniques for organizing computations depend upon this precision and reliability.

Amorphous computing seeks to discover new programming methods that do not require precise control over the interaction or arrangement of the individual computing elements and to instantiate these techniques in new programming languages.

## The Amorphous Computing Model

Amorphous computing models the salient features of an unstructured aggregate through the notion of an *amorphous computer*, a collection of *computational particles* dispersed irregularly on a surface or throughout a volume, where individual particles have no *a priori* knowledge of their positions or orientations. The particles are possibly faulty, may contain sensors and effect actions, and in some applications might be mobile. Each particle has modest computing power and a modest amount of memory. The particles are not synchronized, although they are all capable of operating at similar speeds, since they are fabricated by the same process. All particles are programmed identically, although each particle has means for storing local state and for generating random numbers. There may also be several distinguished particles that have been initialized to particular states.

Each particle can communicate with a few nearby neighbors. In an electronic amorphous computer the particles might communicate via short-distance radio, whereas bioengineered cells might communicate by chemical signals. Although the details of the communication model can vary, the maximum distance over which two particles can communicate effectively is assumed to be small compared with the size of the entire amorphous computer. Communication is assumed to be unreliable and a sender has no assurance that a message has been received (Higher-level protocols with message acknowledgement can be built on such unreliable channels).

We assume that the number of particles may be very large (on the order of  $10^6$  to  $10^{12}$ ). Algorithms appropriate to run on an amorphous computer should be relatively independent of the

number of particles: the performance should degrade gracefully as the number of particles decreases. Thus, the entire amorphous computer can be regarded as a massively parallel computing system, and previous investigations into massively parallel computing, such as research in cellular automata, are one source of ideas for dealing with amorphous computers. However, amorphous computing differs from investigations into cellular automata, because amorphous mechanisms must be independent of the detailed configuration, reliability, and synchronization of the particles.

## Programming Amorphous Systems

A central theme in amorphous computing is the search for programming paradigms that work within the amorphous model. Here, biology has been a rich source of metaphors for inspiring new programming techniques. In embryonic development, even though the precise arrangements and numbers of the individual cells are highly variable, the genetic “programs” coded in DNA nevertheless produce well-defined intricate shapes and precise forms. Amorphous computers should be able to achieve similar results.

One technique for programming an amorphous computer uses diffusion. One particle (chosen by some symmetry-breaking process) broadcasts a message. This message is received by each of its neighbors, which propagate it to their neighbors, and so on, to create a wave that spreads throughout the system. The message contains a count, and each particle stores the received count and increments it before re-broadcasting. Once a particle has stored its count, it stops re-broadcasting and ignores future count messages. This *count-up wave* gives each particle a rough measure of its distance from the original source. One can also produce regions of controlled size, by having the count message relayed only if the count is below a designated bound.

Two such count-up waves can be combined to identify a chain of particles between two given particles *A* and *B*. Particle *A* begins by generating a count-up wave as above. This time, however,

each intermediate particle, when it receives its count, performs a handshake to identify its “predecessor”—the particle from which it received the count (and whose own count will therefore be one less). When the wave of count messages reaches  $B$ ,  $B$  sends a “successor” message, informing its predecessor that it should become part of the chain and should send a message to *its* predecessor, and so on, all the way back to  $A$ . Note that this method works, even though the particles are irregularly distributed, provided there is a path from  $A$  to  $B$ .

The motivating metaphor for these two programs is chemical gradient diffusion, which is a foundational mechanism in biological development (Ashe and Briscoe 2006). In nature, biological mechanisms not only generate elaborate forms, they can also maintain forms and repair them. We can modify the above amorphous line-drawing program so that it produces self-repairing line: first, particles keep re-broadcasting their count and successor messages. Second, the status of a particle as having a count or being in the chain decays over time unless it is refreshed by new messages. That is, a particle that stops hearing successor messages intended for it will eventually revert to not being in the chain and will stop broadcasting its own successor messages. A particle that stops hearing its count being broadcast will start acting as if it never had a count, pick up a new count from the messages it hears, and start broadcasting the count messages with the new count. Clement and Nagpal (2003) demonstrated that this mechanism can be used to generate self-repairing lines and other patterns, and even re-route lines and patterns around “dead” regions where particles have stopped functioning.

The relationship with biology flows in the other direction as well: the amorphous algorithm for repair is a model which is not obviously inconsistent with the facts of angiogenesis in the repair of wounds. Although the existence of the algorithm has no bearing on the facts of the matter, it may stimulate systems-level thinking about models in biological research. For example, Patel et al. use amorphous ideas to analyze the growth of epithelial cells (Patel et al. 2006).

## Amorphous Computing Paradigms

Amorphous computing is still in its infancy. Most of linguistic investigations based on the amorphous computing model have been carried out in simulation. Nevertheless, this work has yielded a rich variety of programming paradigms that demonstrate that one can in fact achieve robustness in face of the unreliability of individual particles and the absence of precise organization among them.

### Marker Propagation for Amorphous Particles

Weiss’s Microbial Colony Language (Weiss 2001) is a marker propagation language for programming the particles in an amorphous computer. The program to be executed, which is the same for each particle, is constructed as a set of rules. The state of each particle includes a set of binary markers, and rules are enabled by boolean combinations of the markers. The rules, which have the form (*trigger, condition, action*) are triggered by the receipt of labelled messages from neighboring particles. A rule may test conditions, set or clear various markers, and it broadcast further messages to its neighbors. Each message carries a count that determines how far it will diffuse, and each marker has a lifetime that determines how long its value lasts. Supporting these language’s rules is a runtime system that automatically propagates messages and manages the lifetimes of markers, so that the programmer need not deal with these operations explicitly.

Weiss’s system is powerful, but the level of abstraction is very low. This is because it was motivated by cellular engineering—as something that can be directly implemented by genetic regulatory networks. The language is therefore more useful as a tool set in which to implement higher-level languages such as GPL (see below), serving as a demonstration that in principle, these higher-level languages can be implemented by genetic regulatory networks as well.

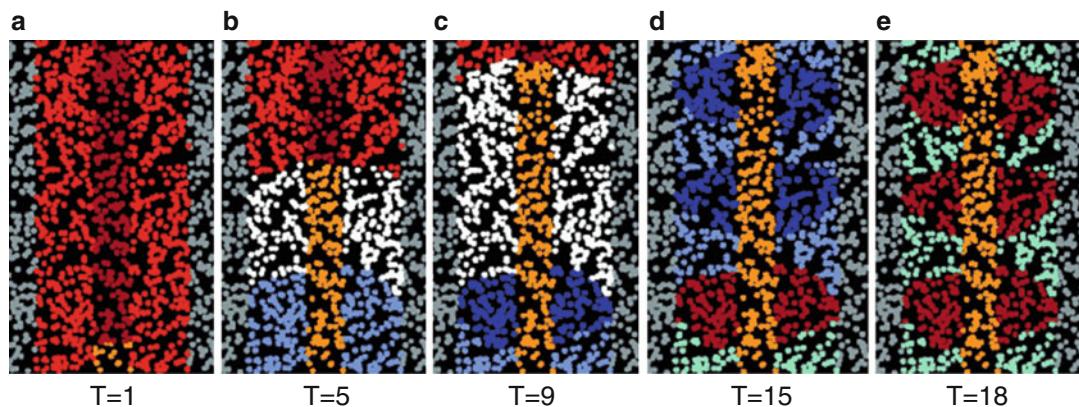
Figure 1 shows an example simulation programmed in this language, that organizes an initially undifferentiated column of particles into a structure with band of two alternating colors: a caricature of somites in developing vertebrae.

### The Growing Point Language

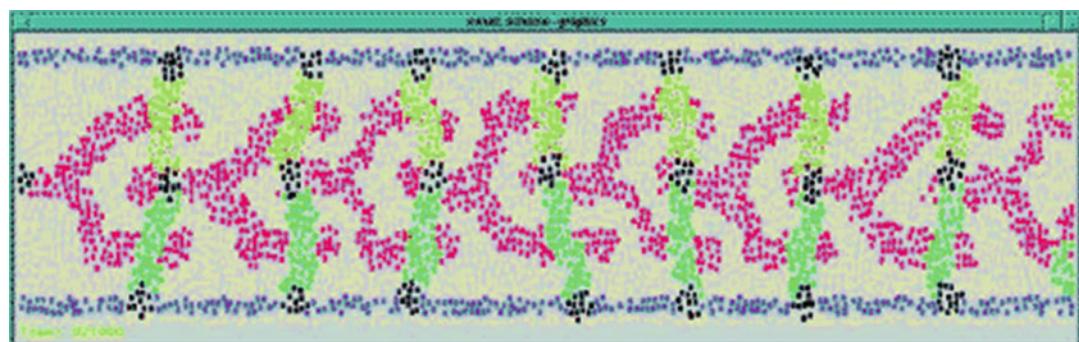
Coore's *Growing Point Language* (GPL) (Coore 1999) demonstrates that an amorphous computer can be configured by a program that is common to all the computing elements to generate highly complex patterns, such as the pattern representing the interconnection structure of an arbitrary electrical circuit as shown in Fig. 2.

GPL is inspired by a botanical metaphor based on growing points and tropisms. A growing point is a locus of activity in an amorphous computer. A growing point propagates through the computer by transferring its activity from one computing element to a neighbor. As a growing point passes through the computer it effects the differentiation of the behaviors of the particles it visits. Particles secrete "chemical" signals whose count-up waves

define gradients, and these attract or repel growing points as directed by programmer-specific "tropisms". Coore demonstrated that these mechanisms are sufficient to permit amorphous computers to generate any arbitrary prespecified graph structure pattern, up to topology. Unlike real biology, however, once a pattern has been constructed, there is no clear mechanism to maintain it in the face of changes to the material. Also, from a programming linguistic point of view, there is no clear way to compose shapes by composing growing points. More recently, Gayle and Coore have shown how GPL may be extended to produce arbitrarily large patterns such as arbitrary text strings(Gayle and Coore 2006). D'Hondt and D'Hondt have explored the use of GPL for geometrical constructions and its relations with



**Amorphous Computing, Fig. 1** A Microbial Colony Language program organizes a tube into a structure similar to that of somites in the developing vertebrate (from (Weiss 2001))



**Amorphous Computing, Fig. 2** A pattern generated by GPL whose shape mimics a chain of CMOS inverters (from (Coore 1999))

computational geometry (D'Hondt and D'Hondt 2001a, b).

### Origami-Based Self-Assembly

Nagpal (2001) developed a prototype model for controlling programmable materials. She showed how to organize a program to direct an amorphous sheet of deformable particles to cooperate to construct a large family of globally-specified predetermined shapes. Her method, which is inspired by the folding of epithelial tissue, allows a programmer to specify a sequence of folds, where the set of available folds is sufficient to create any origami shape (as shown by Huzita's axioms for origami (Huzita and Scimemi 1989)). Figure 3 shows a sheet of amorphous particles, where particles can cooperate to create creases and folds, assembling itself into the well-known origami “cup” structure.

Nagpal showed how this language of folds can be compiled into a low-level program that can be distributed to all of the particles of the amorphous sheet, similar to Coore's GPL or Weiss's MCL. With a few differences of initial state (for example, particles at the edges of the sheet know that they are edge particles) the particles run their copies of the program, interact with their neighbors, and fold up to make the predetermined shape. This technique is quite robust. Nagpal studied the range of shapes that can be constructed using her method, and on their sensitivity to errors of communication, random cell death, and density of the cells.

As a programming framework, the origami language has more structure than the growing point language, because the origami methods allow composition of shape constructions. On the other hand, once a shape has been constructed, there is no clear mechanism to maintain existing patterns in the face of changes to the material.

### Dynamic Recruitment

In Butera (2002)'s “paintable computing”, processes dynamically recruit computational particles from an amorphous computer to implement their goals. As one of his examples Butera uses dynamic recruitment to implement a robust storage system for streaming audio and images

(Fig. 4). Fragments of the image and audio stream circulate freely through the amorphous computer and are marshaled to a port when needed. The audio fragments also sort themselves into a playable stream as they migrate to the port.

To enhance robustness, there are multiple copies of the lower resolution image fragments and fewer copies of the higher resolution image fragments. Thus, the image is hard to destroy; with lost fragments the image is degraded but not destroyed.

Clement and Nagpal (2003) also use dynamic recruitment in the development of active gradients, as described below.

### Growth and Regeneration

Kondacs (2003) showed how to synthesize arbitrary two-dimensional shapes by growing them. These computing units, or “cells”, are identically-programmed and decentralized, with the ability to engage in only limited, local communication. Growth is implemented by allowing cells to multiply. Each of his cells may create a child cell and place it randomly within a ring around the mother cell. Cells may also die, a fact which Kondacs puts to use for temporary scaffolding when building complex shapes. If a structure requires construction of a narrow neck between two other structures it can be built precisely by laying down a thick connection and later trimming it to size.

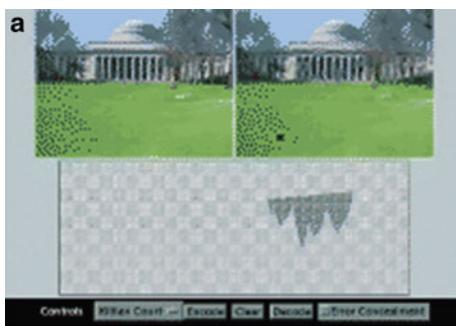
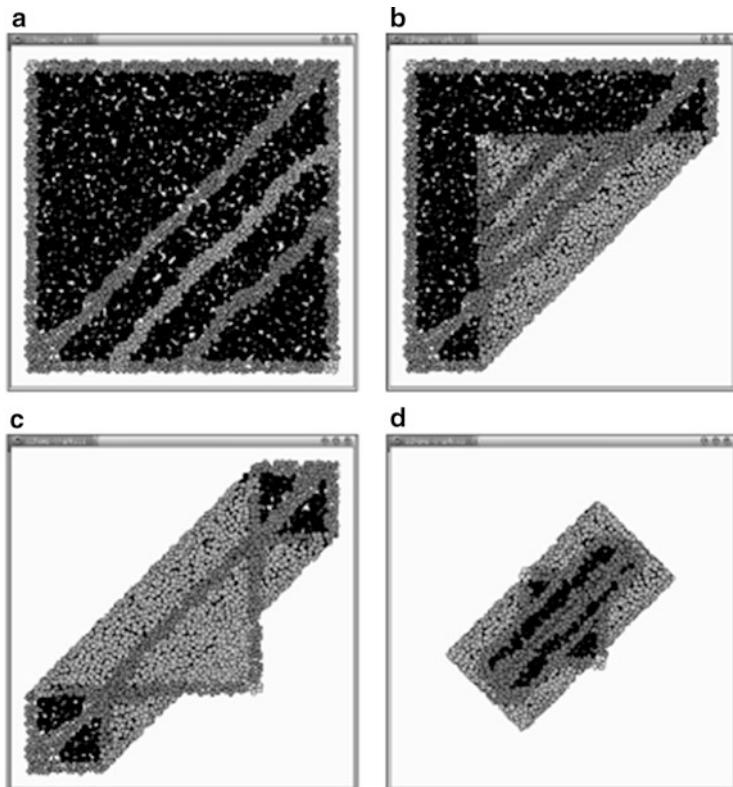
Attributes of this system include scalability, robustness, and the ability for self-repair. Just as a starfish can regenerate its entire body from part of a limb, his system can self-repair in the event of agent death: his sphere-network representation allows the structure to be grown starting from any sphere, and every cell contains all necessary information for reproducing the missing structure.

### Abstraction to Continuous Space and Time

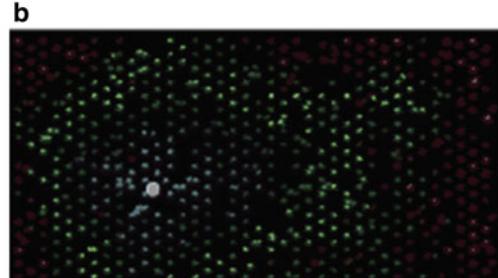
The amorphous model postulates computing particles distributed throughout a space. If the particles are dense, one can imagine the particles as actually filling the space, and create programming abstractions that view the space itself as the object being programmed, rather than the collection of particles. Beal and Bachrach 2006; Bachrach and Beal 2006 pursued this approach by creating a

**Amorphous Computing,**

**Fig. 3** Folding an envelope structure (from (Nagpal 2001)). A pattern of lines is constructed according to origami axioms. Elements then coordinate to fold the sheet using an actuation model based on epithelial cell morphogenesis. In the figure, black indicates the front side of the sheet, grey indicates the back side, and the various colored bands show the folds and creases that are generated by the amorphous process. The small white spots show gaps in the sheet caused by “dead” or missing cells—the process works despite these



Robust Storage



Self-Organization

**Amorphous Computing, Fig. 4** Butera dynamically controls the flow of information through an amorphous computer. In (a) image fragments spread through the computer so that a degraded copy can be recovered from any segment; the original image is on the *left*, the

blurry copy on the *right* has been recovered from the small region shown below. In (b) audio fragments sort themselves into a playable stream as they migrate to an output port; cooler colors are earlier times (from (Butera 2002))

language, Proto, where programmers specify the behavior of an amorphous computer as though it were a continuous material filling the space it occupies. Proto programs manipulate fields of values spanning the entire space. Programming

primitives are designed to make it simple to compile global operations to operations at each point of the continuum. These operations are approximated by having each device represent a nearby chunk of space. Programs are specified in space

and time units that are independent of the distribution of particles and of the particulars of communication and execution on those particles (Fig. 5). Programs are composed functionally, and many of the details of communication and composition are made implicit by Proto’s runtime system, allowing complex programs to be expressed simply. Proto has been applied to applications in sensor networks like target tracking and threat avoidance, to swarm robotics and to modular robotics, e. g., generating a planar wave for coordinated actuation.

Newton’s language Regiment (Newton et al. 2007; Newton and Welsh 2004) also takes a continuous view of space and time. Regiment is organized in terms of stream operations, where each stream represents a time-varying quantity over a part of space, for example, the average value of the temperature over a disc of a given radius centered at a designated point. Regiment, also a functional language, is designed to gather streams of data from regions of the amorphous computer and accumulate them at a single point. This assumption allows Regiment to provide region-wide summary functions that are difficult to implement in Proto.

## Primitives for Amorphous Computing

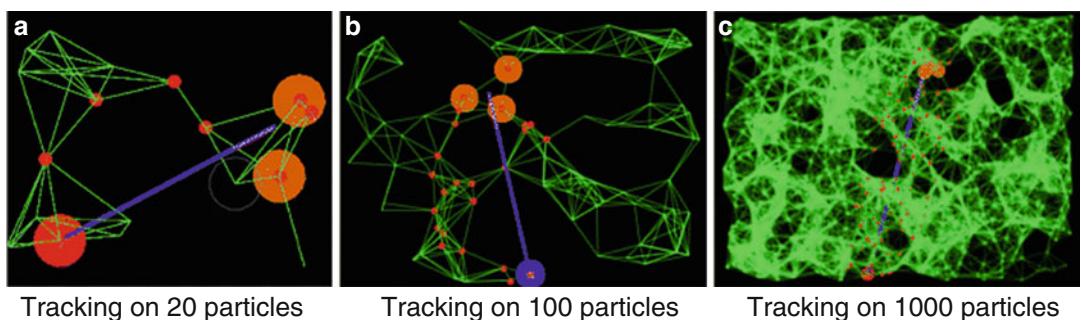
The previous section illustrated some paradigms that have been developed for programming amorphous systems, each paradigm building on some

organizing metaphor. But eventually, meeting the challenge of amorphous systems will require a more comprehensive linguistic framework. We can approach the task of creating such a framework following the perspective in (Abelson et al. 1996), which views languages in terms of primitives, means of combination, and means of abstraction.

The fact that amorphous computers consist of vast numbers of unreliable and unsynchronized particles, arranged in space in ways that are locally unknown, constrains the primitive mechanisms available for organizing cooperation among the particles. While amorphous computers are naturally massively parallel, the kind of computation that they are most suited for is parallelism that does not depend on explicit synchronization and the use of atomic operations to control concurrent access to resources. However, there are large classes of useful behaviors that can be implemented without these tools. Primitive mechanisms that are appropriate for specifying behavior on amorphous computers include gosspp, random choice, fields, and gradients.

### Gossip

Gosspp, also known as epidemic communication (Demers et al. 1987; Ganesan et al. 2002), is a simple communication mechanism. The goal of a gosspp computation is to obtain an agreement about the value of some parameter. Each particle broadcasts its opinion of the parameter to its neighbors, and computation is performed by



**Amorphous Computing, Fig. 5** A tracking program written in Proto sends the location of a target region (orange) to a listener (red) along a channel (small red

dots) in the network (indicated by green lines). The continuous space and time abstraction allows the same program to run at different resolutions

each particle combining the values that it receives from its neighbors, without consideration of the identification of the source. If the computation changes a particle's opinion of the value, it rebroadcasts its new opinion. The process concludes when there are no further broadcasts.

For example, an aggregate can agree upon the minimum of the values held by all the particles as follows. Each particle broadcasts its value. Each recipient compares its current value with the value that it receives. If the received value is smaller than its current value, it changes its current value to that minimum and rebroadcasts the new value.

The advantage of gossipping is that it flows in all directions and is very difficult to disrupt. The disadvantage is that the lack of source information makes it difficult to revise a decision.

### Random Choice

Random choice is used to break symmetry, allowing the particles to differentiate their behavior. The simplest use of random choice is to establish local identity of particles: each particle chooses a random number to identify itself to its neighbors. If the number of possible choices is large enough, then it is unlikely that any nearby particles will choose the same number, and this number can thus be used as an identifier for the particle to its neighbors. Random choice can be combined with gossipping to elect leaders, either for the entire system or for local regions. (If collisions in choice can be detected, then the number of choices need not be much higher than the number of neighbors. Also, using gossipping to elect leaders makes sense only when we expect a leader to be long-lived, due to the difficulty of changing the decision to designate a replacement leader.)

To elect a single leader for the entire system, every particle chooses a value, then gossips to find the minimum. The particle with the minimum value becomes the leader. To elect regional leaders, we instead use gossipping to carry the identity of the first leader a particle has heard of. Each particle uses random choice as a "coin flip" to decide when to declare itself a leader; if the flip comes up heads enough times before the particle hears of another leader, the particle declares itself a leader and broadcasts that fact to its neighbors.

The entire system is thus broken up into contiguous domains of particles who first heard some particular particle declare itself a leader.

One challenge in using random choice on an amorphous computer is to ensure that the particulars of particle distribution do not have an unexpected effect on the outcome. For example, if we wish to control the expected size of the domain that each regional leader presides over, then the probability of becoming a leader must depend on the density of the particles.

### Fields

Every component of the state of the computational particles in an amorphous computer may be thought of as a field over the discrete space occupied by those particles. If the density of particles is large enough this field of values may be thought of as an approximation of a field on the continuous space.

We can make amorphous models that approximate the solutions of the classical partial-differential equations of physics, given appropriate boundary conditions. The amorphous methods can be shown to be consistent, convergent and stable.

For example, the algorithm for solving the Laplace equation with Dirichlet conditions is analogous to the way it would be solved on a lattice. Each particle must repeatedly update the value of the solution to be the average of the solutions posted by its neighbors, but the boundary points must not change their values. This algorithm will eventually converge, although very slowly, independent of the order of the updates and the details of the local connectedness of the network. There are optimizations, such as over-relaxation, that are just as applicable in the amorphous context as on a regular grid.

Katzenelson (1999) has shown similar results for the diffusion equation, complete with analytic estimates of the errors that arise from the discrete and irregularly connected network. In the diffusion equation there is a conserved quantity, the amount of material diffusing. Rauch (1999) has shown how this can work with the wave equation, illustrating that systems that conserve energy and momentum can also be effectively modeled with an amorphous computer. The simulation of the

wave equation does require that the communicating particles know their relative positions, but it is not hard to establish local coordinate systems.

## Gradients

An important primitive in amorphous computing is the gradient, which estimates the distance from each particle to the nearest particle designated as a source. The gradient is inspired by the chemical-gradient diffusion process that is crucial to biological development. Amorphous computing builds on this idea, but does not necessarily compute the distance using diffusion because simulating diffusion can be expensive.

The common alternative is a linear-time mechanism that depends on active computation and relaying of information rather than passive diffusion. Calculation of a gradient starts with each source particle setting its distance estimate to zero, and every other particle setting its distance estimate to infinity. The sources then broadcast their estimates to their neighbors. When a particle receives a message from its neighbor, it compares its current distance estimate to the distance through its neighbor. If the distance through its neighbor is less, it chooses that to be its estimate, and broadcasts its new estimate onwards.

Although the basic form of the gradient is simple, there are several ways in which gradients can be varied to better match the context in which they are used. These choices may be made largely independently, giving a wide variety of options when designing a system. Variations which have been explored include:

### Active Gradients

An active gradient (Clement and Nagpal 2003; Coore 1998; Coore et al. 1997) monitors its validity in the face of changing sources and device failure, and maintains correct distance values. For example, if the supporting sources disappear, the gradient is deallocated. A gradient may also carry version information, allowing its source to change more smoothly.

Active gradients can provide self-repairing coordinate systems, as a foundation for robust construction of patterns. Figure 6 shows the “count-down wave” line described above in the

introduction to this article. The line’s implementation in terms of active gradients provides for self-repair when the underlying amorphous computer is damaged.

### Polarity

A gradient may be set to count down from a positive value at the source, rather than to count up from zero. This bounds the distance that the gradient can span, which can help limit resource usage, but may limit the scalability of programs.

### Adaptivity

As described above, a gradient relaxes once to a distance estimate. If communication is expensive and precision unimportant, the gradient can take the first value that arrives and ignore all subsequent values. If we want the gradient to adapt to changes in the distribution of particles or sources, then the particles need to broadcast at regular intervals. We can then have estimates that converge smoothly to precise estimates by adding a restoring force which acts opposite to the relaxation, allowing the gradient to rise when unconstrained by its neighbors. If, on the other hand, we value adaptation speed over smoothness, then each particle can recalculate its distance estimate from scratch with each new batch of values.

### Carrier

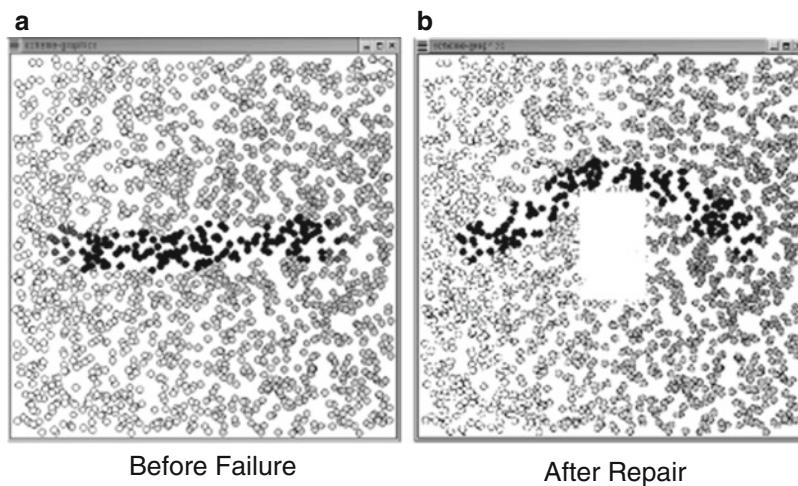
Normally, the distance value calculated by the gradient is the signal we are interested in. A gradient may instead be used to carry an arbitrary signal outward from the source. In this case, the value at each particle is the most recently arrived value from the nearest source.

### Distance Measure

A gradient’s distance measure is, of course, dependent on how much knowledge we have about the relative positions of neighbors. It is sometimes advantageous to discard good information and use only hop-count values, since it is easier to make an adaptive gradient using hop-count values. Non-linear distance measures are also possible, such as a count-down gradient that decays exponentially from the source. Finally, the value of a gradient may depend on more sources than the

### Amorphous Computing,

**Fig. 6** A line being maintained by active gradients, from (Clement and Nagpal 2003). A line (*black*) is constructed between two anchor regions (*dark grey*) based on the active gradient emitted by the right anchor region (*light gray*). The line is able to rapidly repair itself following failures because the gradient actively maintains itself



nearest (this is the case for a chemical gradient), though this may be very expensive to calculate.

### Coordinates and Clusters

Computational particles may be built with restrictions about what can be known about local geometry. A particle may know that it can reliably communicate with a few neighbors. If we assume that these neighbors are all within a disc of some approximate communication radius then distances to others may be estimated by minimum hop count (Kleinrock and Sylvester 1978). However, it is possible that more elaborate particles can estimate distances to near neighbors. For example, the Cricket localization system (Priyantha et al. 2000) uses the fact that sound travels more slowly than radio, so the distance is estimated by the difference in time of arrival between simultaneously transmitted signals. McLurkin's swarmbots (McLurkin 2004) use the ISIS communication system that gives bearing and range information. However, it is possible for a sufficiently dense amorphous computer to produce local coordinate systems for its particles with even the crudest method of determining distances. We can make an atlas of overlapping coordinate systems, using random symmetry breaking to make new starting baselines (Bachrach et al. 2003). These coordinate systems can be combined and made consistent to form a manifold, even if the amorphous computer is not flat or simply connected.

One way to establish coordinates is to choose two initial particles that are a known distance apart. Each one serves as the source of a gradient. A pair of rectangular axes can be determined by the shortest path between them and by a bisector constructed where the two gradients are equal. These may be refined by averaging and calibrated using the known distance between the selected particles. After the axes are established, they may source new gradients that can be combined to make coordinates for the region near these axes. The coordinate system can be further refined using further averaging. Other natural coordinate constructions are bipolar elliptical. This kind of construction was pioneered by Coore (1998) and Nagpal (1999). Katzenelson (1999) did early work to determine the kind of accuracy that can be expected from such a construction.

Spatial clustering can be accomplished with any of a wide variety of algorithms, such as the clubs algorithm (Coore et al. 1997), LOCI (Mittal et al. 2003), or persistent node partitioning (Beal 2003). Clusters can themselves be clustered, forming a hierarchical clustering of logarithmic height.

### Means of Combination and Abstraction

A programming framework for amorphous systems requires more than primitive mechanisms. We also need suitable means of combination, so

that programmers can combine behaviors to produce more complex behaviors, and means of abstraction so that the compound behaviors can be named and manipulated as units. Here are a few means of combination that have been investigated with amorphous computing.

### Spatial and Temporal Sequencing

Several behaviors can be strung together in a sequence. The challenge in controlling such a sequence is to determine when one phase has completed and it is safe to move on to the next. Trigger rules can be used to detect completion locally.

In Coore's Growing Point Language (Coore 1999), all of the sequencing decisions are made locally, with different growing points progressing independently. There is no difficulty of synchronization in this approach because the only time when two growing points need to agree is when they have become spatially coincident. When growing points merge, the independent processes are automatically synchronized.

Nagpal's origami language (Nagpal 2001) has long-range operations that cannot overlap in time unless they are in non-interacting regions of the space. The implementation uses barrier synchronization to sequence the operations: when completion is detected locally, a signal is propagated throughout a marked region of the sheet, and the next operation begins after a waiting time determined by the diameter of the sheet.

With adaptive gradients, we can use the presence of an inducing signal to run an operation. When the induction signal disappears, the operation ceases and the particles begin the next operation. This allows sequencing to be triggered by the last detection of completion rather than by the first.

### Pipelining

If a behavior is self-stabilizing (meaning that it converges to a correct state from any arbitrary state) then we can use it in a sequence without knowing when the previous phase completes. The evolving output of the previous phase serves as the input of this next phase, and once the

preceding behavior has converged, the self-stabilizing behavior will converge as well.

If the previous phase evolves smoothly towards its final state, then by the time it has converged, the next phase may have almost converged as well, working from its partial results. For example, the coordinate system mechanism described above can be pipelined; the final coordinates are being formed even as the farther particles learn that they are not on one of the two axes.

### Restriction to Spatial Regions

Because the particles of an amorphous computer are distributed in space it is natural to assign particular behaviors to specific spatial regions. In Beal and Bachrach's work, restriction of a process to a region is a primitive (Beal and Bachrach 2006). As another example, when Nagpal's system folds an origami construction, regions on different faces may differentiate so that they fold in different patterns. These folds may, if the physics permits, be performed simultaneously. It may be necessary to sequence later construction that depends on the completion of the substructures.

Regions of space can be named using coordinates, clustering, or implicitly through calculations on fields. Indeed, one could implement solid modelling on an amorphous computer. Once a region is identified, a particle can test whether it is a member of that region when deciding whether to run a behavior. It is also necessary to specify how a particle should change its behavior if its membership in a region may vary with time.

### Modularity and Abstraction

Standard means of abstraction may be applied in an amorphous computing context, such as naming procedures, data structures, and processes. The question for amorphous computing is what collection of entities is useful to name.

Because geometry is essential in an amorphous computing context, it becomes appropriate to describe computational processes in terms of geometric entities. Thus there are new opportunities for combining geometric structures and naming the combinations. For example, it is appropriate to compute with, combine, and name regions of

space, intervals of time, and fields defined on them (Beal and Bachrach 2006; Newton and Welsh 2004). It may also be useful to describe the propagation of information through the amorphous computer in terms of the light cone of an event (Bachrach et al. 2007).

Not all traditional abstractions extend nicely to an amorphous computing context because of the challenges of scale and the fallibility of parts and interconnect. For example, atomic transactions may be excessively expensive in an amorphous computing context. And yet, some of the goals that a programmer might use atomic transactions to accomplish, such as the approximate enforcement of conservation laws, can be obtained using techniques that are compatible with an amorphous environment, as shown by Rauch (1999).

## Supporting Infrastructure and Services

Amorphous computing languages, with their primitives, means of combination, and means of abstraction, rest on supporting services. One example, described above, is the automatic message propagation and decay in Weiss's Microbial Colony Language (Weiss 2001). MCL programs do not need to deal with this explicitly because it is incorporated into the operating system of the MCL machine. Experience with amorphous computing is beginning to identify other key services that amorphous machines must supply.

### Particle Identity

Particles must be able to choose identifiers for communicating with their neighbors. More generally, there are many operations in an amorphous computation where the particles may need to choose numbers, with the property that individual particles choose different numbers.

If we are willing to pay the cost, it is possible to build unique identifiers into the particles, as is done with current macroscopic computers. We need only locally unique identifiers, however, so we can obtain them using pseudorandom-number generators. On the surface of it, this may seem problematic, since the particles in the amorphous are assumed to be manufactured identically, with

identical programs. There are, however, ways to obtain individualized random numbers. For example, the particles are not synchronized, and they are not really physically identical, so they will run at slightly different rates. This difference is enough to allow pseudorandom-number generators to get locally out of sync and produce different sequences of numbers. Amorphous computing particles that have sensors may also get seeds for their pseudorandom-number generators from sensor noise.

### Local Geometry and Gradients

Particles must maintain connections with their neighbors, tracking who they can reliably communicate with, and whatever local geometry information is available. Because particles may fail or move, this information needs to be maintained actively. The geometry information may include distance and bearing to each neighbor, as well as the time it takes to communicate with each neighbor. But many implementations will not be able to give significant distance or bearing information. Since all of this information may be obsolete or inaccurately measured, the particles must also maintain information on how reliable each piece of information is.

An amorphous computer must know the dimension of the space it occupies. This will generally be a constant—either the computer covers a surface or fills a volume. In rare cases, however, the effective dimension of a computer may change: for example, paint is three-dimensional in a bucket and two-dimensional once applied. Combining this information with how the number of accessible correspondents changes with distance, an amorphous process can derive curvature and local density information.

An amorphous computer should also support gradient propagation as part of the infrastructure: a programmer should not have to explicitly deal with the propagation of gradients (or other broadcast communications) in each particle. A process may explicitly initiate a gradient, or explicitly react to one that it is interested in, but the propagation of the gradient through a particle should be automatically maintained by the infrastructure.

### Implementing Communication

Communication between neighbors can occur through any number of mechanisms, each with its own set of properties: amorphous computing systems have been built that communicate through directional infrared (McLurkin 2004), RF broadcast (Hill et al. 2000), and low-speed serial cables (Beebe 2007; Raffle et al. 2004). Simulated systems have also included other mechanisms such as signals superimposed on the power connections (Campbell et al. 2005) and chemical diffusion (Weiss 2001).

Communication between particles can be made implicit with a neighborhood shared memory. In this arrangement, each particle designates some of its internal state to be shared with its neighbors. The particles regularly communicate, giving each particle a best-effort view of the exposed portions of the states of its neighbors. The contents of the exposed state may be specified explicitly (Butera 2002; McLurkin 2004; Whitehouse et al. 2004) or implicitly (Beal and Bachrach 2006). The shared memory allows the system to be tuned by trading off communication rate against the quality of the synchronization, and decreasing transmission rates when the exposed state is not changing.

### Lessons for Engineering

As von Neumann remarked half a century ago, biological systems are strikingly robust when compared with our artificial systems. Even today software is fragile. Computer science is currently built on a foundation that largely assumes the existence of a perfect infrastructure. Integrated circuits are fabricated in clean-room environments, tested deterministically, and discarded if even a single defect is uncovered. Entire software systems fail with single-line errors. In contrast, biological systems rely on local computation, local communication, and local state, yet they exhibit tremendous resilience.

Although this contrast is most striking in computer science, amorphous computing can provide lessons throughout engineering. Amorphous computing concentrates on making systems flexible and adaptable at the expense of efficiency. Amorphous computing requires an engineer to

work under extreme conditions. The engineer must arrange the cooperation of vast numbers of identical computational particles to accomplish prespecified goals, but may not depend upon the numbers. We may not depend on any prespecified interconnect of the particles. We may not depend on synchronization of the particles. We may not depend on the stability of the communications system. We may not depend on the long-term survival of any individual particles. The combination of these obstacles forces us to abandon many of the comforts that are available in more typical engineering domains.

By restricting ourselves in this way we obtain some robustness and flexibility, at the cost of potentially inefficient use of resources, because the algorithms that are appropriate are ones that do not take advantage of these assumptions. Algorithms that work well in an amorphous context depend on the average behavior of participating particles. For example, in Nagpal's origami system a fold that is specified will be satisfactory if it is approximately in the right place and if most of the particles on the specified fold line agree that they are part of the fold line: dissenters will be overruled by the majority. In Proto a programmer can address only regions of space, assumed to be populated by many particles. The programmer may not address individual particles, so failures of individual particles are unlikely to make major perturbations to the behavior of the system.

An amorphous computation can be quite immune to details of the macroscopic geometry as well as to the interconnectedness of the particles. Since amorphous computations make their own local coordinate systems, they are relatively independent of coordinate distortions. In an amorphous computation we accept a wide range of outcomes that arise from variations of the local geometry. Tolerance of local variation can lead to surprising flexibility: the mechanisms which allow Nagpal's origami language to tolerate local distortions allow programs to distort globally as well, and Nagpal shows how such variations can account for the variations in the head shapes of related species of *Drosophila* (Nagpal 2001). In Coore's language one specifies the topology of the pattern to be constructed, but only limited information about the geometry. The topology will be

obtained, regardless of the local geometry, so long as there is sufficient density of particles to support the topology. Amorphous computations based on a continuous model of space (as in Proto) are naturally scale independent.

Since an amorphous computer is composed of un-synchronized particles, a program may not depend upon a priori timing of events. The sequencing of phases of a process must be determined by either explicit termination signals or with times measured dynamically. So amorphous computations are time-scale independent by construction.

A program for an amorphous computer may not depend on the reliability of the particles or the communication paths. As a consequence it is necessary to construct the program so as to dynamically compensate for failures. One way to do this is to specify the result as the satisfaction of a set of constraints, and to build the program as a homeostatic mechanism that continually drives the system toward satisfaction of those constraints. For example, an active gradient continually maintains each particle's estimate of the distance to the source of the gradient. This can be used to establish and maintain connections in the face of failures of particles or relocation of the source. If a system is specified in this way, repair after injury is a continuation of the development process: an injury causes some constraints to become unsatisfied, and the development process builds new structure to heal the injury.

By restricting the assumptions that a programmer can rely upon we increase the flexibility and reliability of the programs that are constructed. However, it is not yet clear how this limits the range of possible applications of amorphous computing.

## Future Directions

Computer hardware is almost free, and in the future it will continue to decrease in price and size. Sensors and actuators are improving as well. Future systems will have vast numbers of computing mechanisms with integrated sensors and actuators, to a degree that outstrips our current approaches to system design. When the numbers become large enough, the appropriate programming technology will be

amorphous computing. This transition has already begun to appear in several fields:

Sensor networks	The success of sensor network research has encouraged the planning and deployment of ever-larger numbers of devices. The ad-hoc, time-varying nature of sensor networks has encouraged amorphous approaches, such as communication through directed diffusion (Intanagonwiwat et al. 2000) and Newton's Regiment language (Newton and Welsh 2004).
Robotics	Multi-agent robotics is much like sensor networks, except that the devices are mobile and have actuators. <i>Swarm robotics</i> considers independently mobile robots working together as a team like ants or bees, while <i>modular robotics</i> consider robots that physically attach to one another in order to make shapes or perform actions, working together like the cells of an organism. Gradients are being used to create "flocking" behaviors in swarm robotics (McLurkin 2004; Payton et al. 2001). In modular robotics, Stoy uses gradients to create shapes (Stoy 2003) while De Rosa et al. form shapes through stochastic growth and decay (De Rosa et al. 2006).
Pervasive computing	Pervasive computing seeks to exploit the rapid proliferation of wireless computing devices computing throughout our everyday environment. Mamei and Zambonelli's TOTA system (Mamei and Zambonelli 2004) is an amorphous computing implementation supporting a model of programming using fields and gradients (Mamei and Zambonelli 2005). Servat and Drogoul have suggested combining amorphous computing and reactive agent-based systems to produce something they call "pervasive intelligence" (Servat and Drogoul 2002).
Multicore processors	As it becomes more difficult to increase processor speed, chip manufacturers are looking for performance gains through increasing the number of processing cores per chip. Butera's work (Butera 2002) looks toward a future in which there are thousands of cores per chip and it is no longer reasonable to assume they are all working or have them communicate all-to-all.

While much of amorphous computing research is inspired by biological observations, it is also likely that insights and lessons learned from programming amorphous computers will help elucidate some biological problems (Patel et al. 2006). Some of this will be stimulated by the emerging engineering of biological systems. Current work in synthetic biology (igem 2006; Registry of standard 2007; Weiss et al. 2003) is centered on controlling the molecular biology of cells. Soon synthetic biologists will begin to engineer biofilms and perhaps direct the construction of multicellular organs, where amorphous computing will become an essential technological tool.

## Bibliography

### Primary Literature

- Abelson H, Sussman GJ, Sussman J (1996) Structure and interpretation of computer programs, 2nd edn. MIT Press, Cambridge
- Ashe HL, Briscoe J (2006) The interpretation of morphogen gradients. *Development* 133:385–394
- Bachrach J, Beal J (2006) Programming a sensor network as an amorphous medium. In: DCOSS 2006 posters
- Bachrach J, Nagpal R, Salib M, Shrobe H (2003) Experimental results and theoretical analysis of a self-organizing global coordinate system for ad hoc sensor networks. *Telecommun Syst* 26(2–4):213–233. Special Issue on Wireless System Networks
- Bachrach J, Beal J, Fujiwara T (2007) Continuous space-time semantics allow adaptive program execution. In: IEEE international conference on self-adaptive and self-organizing systems
- Beal J (2003) A robust amorphous hierarchy from persistent nodes. In: Commun System Networks
- Beal J (2004) Programming an amorphous computational medium. In: Unconventional programming paradigms international workshop
- Beal J (2005) Amorphous medium language. In: Large-scale multi- agent systems workshop (LSMAS). Held in Conjunction with AAMAS-05
- Beal J, Bachrach J (2006) Infrastructure for engineered emergence on sensor/actuator networks. In: IEEE intelligent systems
- Beal J, Sussman G (2005) Biologically-inspired robust spatial programming. Technical report AI Memo 2005-001, MIT
- Beebee W (2007) M68hc11 gunk api book. <http://www.swiss.ai.mit.edu/projects/amorphous/HC11/api.html>. Accessed 31 May 2017
- Butera W (2002) Programming a paintable computer. PhD thesis, MIT
- Campbell J, Pillai P, Goldstein SC (2005) The robot is the tether: active, adaptive power routing for modular robots with unary inter-robot connectors. In: IROS
- Clement L, Nagpal R (2003) Self-assembly and self-repairing topologies. In: Workshop on adaptability in multi-agent systems, RoboCup Australian Open
- Codd EF (1968) Cellular automata. Academic, New York
- Coore D (1998) Establishing a coordinate system on an amorphous computer. In: MIT student workshop on high performance computing
- Coore D (1999) Botanical computing: a developmental approach to generating interconnect topologies on an amorphous computer. PhD thesis, MIT
- Coore D, Nagpal R, Weiss R (1997) Paradigms for structure in an amorphous computer. Technical report AI Memo 1614, MIT
- D'Hondt E, D'Hondt T (2001a) Amorphous geometry. In: ECAL
- D'Hondt E, D'Hondt T (2001b) Experiments in amorphous geometry. In: International conference on artificial intelligence
- De Rosa M, Goldstein SC, Lee P, Campbell J, Pillai P (2006) Scalable shape sculpting via hole motion: motion planning in lattice-constrained module robots. In: Proceedings of the 2006 I.E. international conference on robotics and automation (ICRA'06)
- Demers A, Greene D, Hauser C, Irish W, Larson J, Shenker S, Stuygis H, Swinehart D, Terry D (1987) Epidemic algorithms for replicated database maintenance. In: 7th ACM symposium on operating systems principles
- Dust Networks (2007) <http://www.dust-inc.com>. Accessed 31 May 2017
- Ganesan D, Krishnamachari B, Woo A, Culler D, Estrin D, Wicker S (2002) An empirical study of epidemic algorithms in large scale multihop wireless networks. Technical report IRB-TR-02-003, Intel Research Berkeley
- Gayle O, Coore D (2006) Self-organizing text in an amorphous environment. In: ICCS
- Hill J, Szewczyk R, Woo A, Culler D, Hollar S, Pister K (2000) System architecture directions for networked sensors. In: ASPLOS
- Huzita H, Scimemi B (1989) The algebra of paper-folding. In: First international meeting of origami science and technology
- igem (2006) 2006: international genetically engineered machine competition. <http://www.igem2006.com>. Accessed 31 May 2007
- Intanagonwiwat C, Govindan R, Estrin D (2000) Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: Mobile computing and networking, pp 56–67
- Kahn JM, Katz RH, Pister KS J (1999) Mobile networking for smart dust. In: ACM/IEEE international conference on mobile computing and networking (MobiCom 99)
- Katzenelson J (1999) Notes on amorphous computing. (Unpublished Draft)
- Kleinrock L, Sylvester J (1978) Optimum transmission radii for packet radio networks or why six is a magic

- number. In: IEEE national telecommunication conference, pp 4.3.1–4.3.5
- Knight TF, Sussman GJ (1998) Cellular gate technology. In: First international conference on unconventional models of computation (UMC98)
- Kondacs A (2003) Biologically-inspired self-assembly of 2d shapes, using global-to-local compilation. In: International joint conference on artificial intelligence (IJCAI)
- Mamei M, Zambonelli F (2003) Spray computers: Frontiers of self-organization for pervasive computing. In: WOA
- Mamei M, Zambonelli F (2004) Spatial computing: the tota approach. In: WOA, pp 126–142
- Mamei M, Zambonelli F (2005) Physical deployment of digital pheromones through rfid technology. In: AAMAS, pp 1353–1354
- Margolus N (1988) Physics and computation. PhD thesis, MIT
- McLurkin J (2004) Stupid robot tricks: a behavior-based distributed algorithm library for programming swarms of robots. Master's thesis, MIT
- Mittal V, Demirbas M, Arora A (2003) Loci: local clustering service for large scale wireless sensor networks. Technical report OSU-CISRC-2/03-TR07, Ohio State University
- Nagpal R (1999) Organizing a global coordinate system from local information on an amorphous computer. Technical report AI Memo 1666, MIT
- Nagpal R (2001) Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics. PhD thesis, MIT
- Nagpal R, Mamei M (2004) Engineering amorphous computing systems. In: Bergenti F, Gleizes MP, Zambonelli F (eds) Methodologies and software engineering for agent systems, the agent-oriented software engineering handbook. Kluwer, New York, pp 303–320
- Newton R, Welsh M (2004) Region streams: functional macroprogramming for sensor networks. In: First international workshop on data management for sensor networks (DMSN)
- Newton R, Morrisett G, Welsh M (2007) The regiment macroprogramming system. In: International conference on information processing in sensor networks (IPSN'07)
- Patel A, Nagpal R, Gibson M, Perrimon N (2006) The emergence of geometric order in proliferating metazoan epithelia. *Nature* 442:1038–1041
- Payton D, Daily M, Estowski R, Howard M, Lee C (2001) Pheromone robotics. *Auton Robot* 11:319–324
- Priyantha N, Chakraborty A, Balakrishnan H (2000) The cricket location-support system. In: ACM international conference on mobile computing and networking (ACM MOBICOM)
- Raffle H, Parkes A, Ishii H (2004) Topobo: a constructive assembly system with kinetic memory. CHI pp 647–654
- Rauch E (1999) Discrete, amorphous physical models. Master's thesis, MIT
- Registry of standard biological parts. <http://parts.mit.edu>. Accessed 31 May 2007
- Servat D, Drogoul A (2002) Combining amorphous computing and reactive agent-based systems: a paradigm for pervasive intelligence? In: AAMAS
- Stoy K (2003) Emergent control of self-reconfigurable robots. PhD thesis, University of Southern Denmark
- Sutherland A (2003) Towards rseam: resilient serial execution on amorphous machines. Master's thesis, MIT
- von Neumann J (1951) The general and logical theory of automata. In: Jeffress L (ed) Cerebral mechanisms for behavior. Wiley, New York, p 16
- Weiss R (2001) Cellular computation and communications using engineered genetic regular networks. PhD thesis, MIT
- Weiss R, Knight T (2000) Engineered communications for microbial robotics. In: Sixth international meeting on DNA based computers (DNA6)
- Weiss R, Basu S, Hooshangi S, Kalmbach A, Karig D, Mehreja R, Netravali I (2003) Genetic circuit building blocks for cellular computation, communications, and signal processing. *Nat Comput* 2(1):47–84
- Welsh M, Mainland G (2004) Programming sensor networks using abstract regions. In: Proceedings of the first USENIX/ACM symposium on networked systems design and implementation (NSDI'04)
- Werfel J, Bar-Yam Y, Nagpal R (2005) Building patterned structures with robot swarms. In: IJCAI
- Whitehouse K, Sharp C, Brewer E, Culler D (2004) Hood: a neighborhood abstraction for sensor networks. In: Proceedings of the 2nd international conference on Mobile systems, applications, and services

## Books and Reviews

- Abelson H, Allen D, Coore D, Hanson C, Homsky G, Knight T, Nagpal R, Rauch E, Sussman G, Weiss R (1999) Amorphous computing. Technical report AIM-1665, MIT



---

## Reservoir Computing

Zoran Konkoli

Department of Microtechnology and  
Nanoscience - MC2, Chalmers University of  
Technology, Gothenburg, Sweden

### Article Outline

Glossary

Introduction

Why Is Reservoir Computing Useful?

Quality of the Reservoir

A Neural Network as a Reservoir

The Origins of Reservoir Computing

Essential Mathematical Concepts and Definitions

Reservoir as a Filter

Two Models of Reservoir Computing

Future Directions

Bibliography

### Glossary

**System** Contains many parts that interact together. This object can be a mathematical abstraction, but it can also be a real entity, something one can stimulate, observe, or interact with in some other way. Examples: The living cell harbors one million reactions per second. Taken together, the reacting molecules form a system (from a dynamical point of view a rather complicated one); A rock made of atoms is a system (though much simpler than a cell).

**Dynamical system** A system that can evolve in time according to a specific set of rules. The time can be either discrete (abrupt changes at specific time instances) or continuous (smooth changes). Without the loss of generality, for simplicity reasons, only

deterministic systems will be discussed (no stochastic dynamics).

**Configuration or state of the system** The most detailed description of a dynamical system at a given time instance. The appropriate level of such a description depends on what the system is used for. For example, to describe a motion of a rock, 6 degrees of freedom might be sufficient (3 translational and 3 rotational degrees of freedom). However, to specify the individual positions of the atoms in the rock, that would require an astronomical number of variables.

**Configuration or phase space** The collection of all possible states of the system.

**Trajectory** Describes how a dynamical system evolves in time. A trajectory is a “line” in the configuration space. Starting from an initial state at some specific time point that defines the initial time, as time flows, the dynamical system adopts specific configurations according to the internal set of rules. Such a list of configurations is ordered, for a given configuration in the list, it is always clear which configuration precedes it and which one follows.

**Reservoir** A dynamical system with specific set of properties, which make it useful for information processing purposes. These will be specified further in the text. The most important property is the separability property.

**Readout layer** A readout layer is an interface between the reservoir internals and the external world (observer who is trying to realize a computation). An equipment that needs to be engineered to assess the state of the reservoir. This assessment is used to produce an output of the computation.

**Training** A process of adjusting the parameters of the system to achieve some specific computation. Described in a more detail further in the text.

**Filter** An abstract mathematical object that transforms a sequence of numbers into another sequence of numbers. Specified in more detail further in the text.

## Introduction

A reservoir computer consists essentially of two components: a dynamical system that accepts an input signal and a trainable readout layer which is used to produce the output. *The key claim is that if the system is complex enough, in principle, any computation can be achieved by simply training the readout layer for a desired functionality.* Usually, reservoir computing is often used for time data series information processing. The input is applied over a period of time. In strict theoretical computer science terms, the statement is that the universal computation is possible in the context of time data series information processing.

Reservoir computing was independently suggested by Maass et al. (2002), T. Natschler et al. (2002), Jaeger and Haas (2004), and Jaeger (2001, 2010). These studies introduced the concepts of Liquid State Machines (LSMs) and Echo State Networks (ESNs), respectively. These investigations focused on systematically explaining some specific features of neural network dynamics that had been observed in earlier investigations. The field of reservoir computing emerged from these pioneering formalizations of the behaviors relevant for reservoir computing. Further subsequent improvements of the original ideas fully shaped what is now referred to as a field of reservoir computing. Useful reviews of reservoir computing can be found in Lukoievicius et al. (2012) and Lukoievicius and Jaeger (2009).

**The Flow of Information in a Reservoir Computer** In a reservoir computer, the computation is performed essentially in two major steps. First, an input is applied to the system over a time period. The input drives the system to a certain state in the configuration space. The transitions among the states are governed by the rules that specify the dynamics of the system. Second, at the end of the time period, a readout layer is used to quickly assess the state of the system and produce the final output. The observation period is shifted in time, so the system can perform an online computation. *The first step (where the actual computation is performed) comes for “free,” since there is no need to specifically engineer the system*

for reservoir computing. Only the second step needs to be designed carefully which often implies some engineering constraints.

The dynamical system as information transducer: From the information processing point of view, the system operates as an abstract machine that changes the representation of the input. The reservoir transforms the information stored in the input by converting it into a state of the system. The readout layer is used to assess the internal state of the system. In this setup, the readout layer does not perform any substantial computation, since it is only used to quickly check the state of the system and produce a result of that assessment, typically by assigning a real number to each state.

Several examples that exploit different reservoirs can be mentioned: Reservoirs made of memristor networks have been considered for pattern recognition (Kulkarni and Teuscher 2012; Carbajal et al. 2015) or harmonics generation (Konkoli and Wenden 2014); photonic systems as reservoirs have been discussed in Appeltant et al. (2011), Langer et al. (2012), and Mesaritakis et al. (2015).

The question is what is the computing capacity of such devices? A few examples of studies that address such questions can be found in Appeltant et al. (2011), Dambre et al. (2012), Massar and Massar (2013), Goudarzi and Stefanovic (2014), Soriano et al. (2015), and Bennett et al. (2016). It is clear that to be used for universal computation, the dynamical system should exhibit a minimal degree of complexity necessary to ensure a sufficient number of states that can sustain reservoir computing approach. Accordingly, the dynamical system is predominantly viewed as a placeholder of states. *Thus in the context of reservoir computing the term “reservoir” stands for a collection of states (configurations).* This is the reason why a dynamical system used for reservoir computing is often referred to as the reservoir.

**Two Views on Reservoir Computing** Reservoir computing can be viewed in two ways, different but strongly related, as a paradigm of computation and a practical recipe for building computers. As a paradigm of computation, reservoir computing

provides an extensive and rigorous mathematical justification for using an arbitrary dynamical system (for arbitrary computation) without detailed preparation. Seen from another, less abstract angle, reservoir computing is also a remarkably simple and *practical recipe* for turning an arbitrary dynamical system into a computer. Perhaps the biggest advantage of reservoir computing lies in the fact that it can be used without understanding the mathematical justification behind the idea. The recipe is remarkably simple: (a) *choose* a dynamical system (based on intuition or some other considerations), (b) engineer a *simple* readout layer, and (c) train the readout layer.

**Text Organization** The following text has been structured to emphasize the key ideas that have driven the reservoir computing field forward and how they originally emerged. The goal was to transcendent disciplinary boundaries and make the topics accessible to a broader audience. Each statement made in the text can be related to a rigorous mathematical argument. However, whenever possible, the heavy technical details have been presented in more intuitive terms, and the related mathematical rigor has been kept in the background. There are places where mathematical rigor is necessary, but the number of such places has been reduced to the minimum. When such passages do occur, they are very carefully written (the level of mathematics has been kept at the advanced undergraduate level).

Further, the text has been organized to emphasize the narrative. Essentially, the idea behind the structure is that the text should be read in one sweep, without the need to heavily cross-reference among sections. At the end, the reader should have a very good idea what reservoir computing is, how it can be used, and which advantages it offers.

An attempt has been made to provide a correct historical development of ideas, but not at the expense of the narrative and the logical progression of ideas. For example, it is often the case that the historical development of scientific ideas is not necessarily correlated with the best way of presenting them. For most part the logical

progression of ideas and their development through history go hand in hand in the field of reservoir computing, but not always. In such situations, a preference was given to the logical flow of the narrative.

## Why Is Reservoir Computing Useful?

The view on computation adopted in reservoir computing might appear a bit arbitrary, after all, the standard computer functions in a similar way: the external input drives the computation by influencing how the central processing unit changes its state, and the memory or output units contain the final result. Thus the reservoir is related to the central processing unit and the readout layer to an output unit such as a piece of the computer memory connected with the screen or the printer. However, the emphasis on the two-step reservoir computing structure consisting of the information transformation phase (computation), followed by a readout, is extremely useful. The related practical and technological implications are profound.

**Engineering Freedom** If the universal computation is indeed possible this way, it is much easier to build information processing devices. The hard part of building a central processing unit can be avoided, at least in principle, by simply picking up an existing dynamical system. There is a great deal of *engineering freedom* in constructing the reservoir. In practical terms, the difficulties are related to the reservoir selection process, but not necessarily to the process of engineering one.

Most of the technological challenges that might surface in the reservoir computer building process are contained within the procedures of building an interface that can assess the states of the system. Since this interface is not expected to perform any substantial computation, the expectation is that such an interface should be relatively simple to engineer. Having said the above, there is nothing that speaks against engineering the reservoir too. Should this be of interest, the engineering process might be much more flexible than in the case of the traditional microprocessor design.

## Quality of the Reservoir

The question is what makes a good reservoir. A key feature is that determines the quality of the reservoir is the complexity of its configuration space. To sustain a universal computation, a good reservoir needs to exhibit complex and state rich configuration space. How complex should the system be so that it can be used as a reservoir? Indeed, without a precise answer to this question, the key claim might sound rather empty or even unrealistic.

For example, while it would be very hard to turn a rock into a computer it is intuitively clear that doing the same with the living bacterial cell should be easier, though one might not be able to compute much. After all bacteria evolved to solve relatively simple decision problems. If one were to use a human brain as the reservoir, then there would be no difficulty at all in building a very advanced information processing system. Naturally, this will never be done in practice due to the obvious ethical considerations. But as a thought experiment, assuming that a volunteer can be found to serve as a reservoir, one could simply address the person with the relevant questions and observe the answers. To obtain a formal output, as a readout layer, one could simply use a speech recognition apparatus.

The above thought experiments illustrate that the complexity of the system that is used as the reservoir plays an important role. Intuitively, one expects that systems with an increasing degree of complexity will be better reservoirs. What makes reservoir computing useful is the fact that precise mathematical criteria have been identified that guarantee the sufficient amount of complexity. The key requirement is that the reservoir needs to separate inputs. Roughly, different external inputs should drive the system to different regions of the configuration space. This criterion will be defined precisely later on in exact mathematical terms.

## A Neural Network as a Reservoir

The field of reservoir computing emerged as a synthesis of observations which were accumulated

during the process of training the artificial neural network. To understand reservoir computing, it is necessary to have a clear idea how neural networks operate are how they are trained. An illustrative example is presented below that shows how this is done. Readers familiar with neural networks can skip this section. However, the example discussed below prepares the stage for the next section, where some important reservoir computing principles are pointed out.

Assume that the goal is to analyze pictures of flowers, where each picture is digitalized and is represented as a vector of values that specify brightness of each pixel. The goal is to identify a flower type shown in each photograph. Note that many pictures can represent the same flower type. It could have happened that while the photograph was taken, the same flower was rotated, or that picture of several flowers from the same family were taken.

In a typical pattern recognition setup, when a neural network is used to design the pattern recognition device, neurons are divided into three groups, the input neurons, the bulk neurons, and the output or outer neurons. The neurons do not need to be connected in the nearest neighbor fashion, so the usual geometrical analogies do not necessarily apply. An artificial neural network is essentially a geometry free object, and it is hard to define what is *inside* and what is *outside*. In here, *these terms describe the positioning of the neurons with regard to the flow of information through the system*. It is sufficient to define the input and the output neurons. The remaining neurons are the bulk neurons.

The input neurons are attached to the pixels in the picture. In the most optimal case, there is a one-to-one correspondence between the pixels and the neurons. The state of each neuron in this group (e.g., its voltage value) is assigned based on the value of the pixel it is attached to. Another subset of neurons serves as the output layer. In this set each neuron represents a flower label. The number of outer neurons is exactly the same as the number of labels.

The recognition process works as follows: A picture is shown to the network. This is done by initializing the input neurons. For simplicity

reasons, it is assumed that the input neurons are driven by values that are constant in time. Subsequently, the bulk and the output neurons are left to evolve in time and they eventually attain stable voltage values too. If the network works properly, in the most ideal case, only one output neuron should fire (i.e., its potential should be high), all others should not (i.e., their potentials should be low). This would signal that the flower in the picture is clearly of the type that the output neuron corresponds to. It is possible that more than one neuron fires. This indicates that the classification performed by the device is inconclusive.

**Training (Weight Adjustment)** In fact, prior to the process of adjusting the weights, a network with arbitrary weights makes random assignments and repeatedly predicts wrong labels for many flowers. For random weights the error of prediction is large. To reduce this prediction error, the weights need to be carefully adjusted so that the desired pattern recognition behavior is achieved. *The process of the weight adjustment is referred to as the training process.*

The training process is realized by repeatedly showing labeled pictures to the network, which constitute the training set. For each picture shown, the weights can be altered in a systematic manner by trying to achieve a better agreement between the network prediction and the desired result. The usual behavior is that the prediction improves as more pictures are processed. Note that in this context the word training is used in a very special way. The training procedure is in line with the human intuition what the word training or learning means (a repeated exposure to information resulting in a final synthesis). However, if the weights were apriority known, then no such tedious procedure would be needed. Still, with the abuse of terminology, such one-step assignment will also be referred to as training.

## The Origins of Reservoir Computing

The key observation that motivated the reservoir computing idea was the realization that *during the incremental weight improvement processes*

*mostly the weights associated with the outmost layer are altered* (the neurons closest to the output layer). Other weights associated with the bulk neurons usually do not change much. It is exactly this behavior that prevents applying back-propagation training method to deep feed-forward networks. This issue has been addressed recently by the advances of deep learning techniques. However, while being a problem from the deep learning perspective, in the reservoir computing context, this behavior is something one wishes to exploit. A natural way to exploit this behavior is to simply focus on training the outmost layer of neurons.

At this stage the most relevant question is: *Is it possible to achieve the same functionality by only training the weights of the outmost layer of neurons?* The answer is a resounding “yes.” In fact, architecturally, the outer layer resembles the feed-forward network, which is relatively simple to train. Further, in plethora of applications it has been shown that even a linear readout layer is sufficient to achieve any desired information processing behavior. Linear readout layers are simple to train. The least squares method can be used with a great advantage. Complex neural networks have often been used as reservoirs.

*This begs several important follow-up (big) questions: Why is it sufficient to only train the readout layer? Why can such a layer be kept simple? Are there distinct features of the network that naturally sustain reservoir computing? Further, equally important questions are: Is it possible to use other system instead of the neural network as the reservoir? Why are the criteria the system needs to satisfy to be used in such a way? Are these criteria different for different (classes of) hardware realizations, or are there some generic (hardware free) principles that can be identified?*

Some of these questions have been answered over the years, but it is fair to say that the field is still in its infancy. There are many questions that remain open. In the literature, two types of approaches (fully equivalent) have been developed to formalize the answers to some of these questions in a rigorous mathematical fashion, the Echo State Network model and the Liquid State

Machine model, and these are explained in the separate sections. Further, there is a remarkable connection between the field of reservoir computing and philosophy of mind (discussed in the concluding section).

## Essential Mathematical Concepts and Definitions

While the Liquid State Machines and Echo State Networks have been developed independently at the same time, and were motivated by a slightly different set of questions, it is more natural to explain ESNs first and LSMs after that. To do this, some formal mathematical definitions are necessary. This will greatly facilitate the discussions that follow.

The concept of the time data series and the operation on the time data are crucial ingredients of the discussion. Assume that the values of a variable  $q$  that evolves in time are ordered and collected into a sequence  $\mathbf{q}$ . Once the sequence  $\mathbf{q}$  is given, the value of  $q$  at a time  $t$ , to be denoted by  $q_t$ , can be retrieved as

$$q_t = \mathbf{q}(t) \quad (1)$$

One can think of  $\mathbf{q}$  as an array of values indexed by time. However, the index set used to extract the individual variable values is uncountable. Time is a continuous variable, a real number  $t \in \mathbb{R}$ , where  $\mathbb{R}$  stands for the set of real numbers. This implies that it is impossible to list the values in the sequence. Instead, it appears that it is more useful to think about  $q$  as a function of time. Nevertheless, for the purposes of explaining reservoir computing it is better to insist on the sequence representation.

To be able to list the values in  $\mathbf{q}$ , it will be assumed that time is discrete. In practice, this implies that all variables are allowed to change at specific time instances  $t = \hat{t}\tau$  only, with  $\hat{t} \in \mathbb{Z}$  where  $\mathbb{Z}$  is the set of integers,  $\mathbb{Z} \equiv \{\dots, -2, -1, 0, 1, 2, \dots\}$ . Should a need occur, one can exploit the limit  $\tau \rightarrow 0$  to retrieve the continuous time representation, being careful about the transition from a countable index set  $\mathbb{Z}$  that can be ordered to the uncountable index set  $\mathbb{R}$  that cannot.

In the discrete time representation, since the index set is a countable one, the sequence values can be listed  $\mathbf{q} = (\dots, q_{-2\tau}, q_{-\tau}, q_0, q_\tau, q_{2\tau}, \dots)$  and with a change of variables  $q_t = \hat{q}_{\hat{t}}$  the sequence can be simply written as

$$\mathbf{q} = (\dots, \hat{q}_{-2}, \hat{q}_{-1}, \hat{q}_0, \hat{q}_1, \hat{q}_2, \dots) \quad (2)$$

In what follows, the statements that are made are universal in the sense that they hold regardless of the representation (discrete or continuous). To simplify the notation, in the following, the hat symbol above the symbols  $\hat{q}$  and  $\hat{t}$  will be omitted. In the context of reservoir computing, the relevant data series are the input data series  $\mathbf{u} = (\dots, u_{t-1}, u_t, u_{t+1}, \dots)$ , the output data series  $\mathbf{v} = (\dots, v_{t-1}, v_t, v_{t+1}, \dots)$ , and the series that describes how the states of the reservoir evolve in time  $\mathbf{x} = (\dots, x_{t-1}, x_t, x_{t+1}, \dots)$ .

## Reservoir as a Filter

A concept that is extremely relevant for reservoir computing is the one of a filter. In fact, the sole purpose of introducing the sequence notation in the earlier section was to be able to define what a filter is in relatively rigorous but still intuitive fashion.

A filter  $\mathcal{F}$  is a mapping that takes an input sequence  $\mathbf{u}$  and converts it to another sequence  $\mathbf{v}$  as  $\mathbf{v} = \mathcal{F}[\mathbf{u}]$ . The individual values of the output can be retrieved by using the time index

$$v(t) = \mathcal{F}[u](t) \quad (3)$$

Note that equation above does not read  $v(t) = \mathcal{F}[u(t)]$ . That would be wrong since filters operate on whole sequences, and not on numbers, and  $u(t) = u_t$  is a number (a particular value in the sequence).

In the discrete time representation, the dynamics of the reservoir is uniquely defined as

$$x_t = \mathcal{R}(x_{t-1}, u_t) \quad (4)$$

where  $x_t$  denotes the state of the reservoir a particular time instance. If the above mapping is

iterated, one can see that the state at a time  $t$  is defined by the input the system has experienced (so far it can behave as a filter) and the initial state of the system in the infinite past (this is why it is not strictly speaking a filter). To emphasize this we write

$$x_t = \mathcal{R}[u | \xi] \quad (5)$$

where  $\xi = \lim_{t \rightarrow -\infty} x_{-t}$  denotes the initial condition of the system in the infinite past. Further, the output at each time instance is given by

$$y_t = \psi(x_t) \quad (6)$$

Note that we do not write  $y_t = \psi[x]$  which would imply a readout layer that can store and process an infinite sequence of system states  $x = (\dots, x_{t-1}, x_t, x_{t+1}, \dots)$ . The reservoir computing readout layer only cares about the present states.

The most important thing to realize is that the filter only cares about the sequence it is “inspecting.” It does not care about other variables. This might sound trivial, but one should be observant of this fact, for the following reasons. A dynamical system has the potential to define a filter, but whether it does so depends on whether the initial condition is forgotten or not, which in turn depends on the structure of its dynamics. If it is true that in some sense

$$\mathcal{R}[u | \xi] \approx \mathcal{R}[u] \quad (7)$$

then the dynamical system indeed defines a filter. Otherwise, the system defines a filter with an extra parameter. This can be both an opportunity and a problem. In reservoir computing it is mostly a problem.

One might want to exploit the possibility of working with  $\mathcal{R}[u | \xi]$ , e.g., by using  $\xi$  as an extra training parameter, but in the context of online information processing this would be less useful. Prior to any computation, one would have to carefully prepare the initial state of the system. But how does one control the initial state of a complex system? *Clearly if the initial condition*

*has to be trained, advantages of the engineering freedom disappear.* The strategy of training the initial state might work for a simple logical gates, or even a complicated CMOS gate, but not for complex reservoirs. *This is the reason why in the reservoir computing only dynamical systems for which initial condition gets forgotten are useful.* In the field of reservoir computing, this is expressed in two ways, by saying that the system has *the fading memory property* or that the system has *the echo state property*.

## Two Models of Reservoir Computing

### Echo State Networks

In the formulation of the Echo State Network model, neural networks play a central role. The model is formulated by assuming that the artificial neural network is used as the reservoir. The state of the network is defined by the voltage value at each node  $x \equiv (x_1, x_2, \dots, x_n)$ , when there are  $n$  neurons in the system. The update rules for the model are specified as

$$x_t = \sigma(Wx_{t-1} + Vu_t) \quad (8)$$

where  $\sigma$  is a sigmoid-like activation function,  $W$  is an  $n \times n$  matrix, and  $V$  is an  $n$  dimensional vector. The readout at each time instance is given by the standard readout rule given by Eq. (6).

The most important concept in the Echo State Network model is the idea of the “echo.” Inputs to the reservoir echo in the system either forever or until they dissipate. The system is said to have the echo state property if its present state depends on the sequence of inputs it has been exposed to in a unique way.

The original definition of the echo state property is as follows (Jaeger 2010): Assume a fixed time instance  $t$ . Let  $q[-\infty: t] \equiv (\dots, q_{t-2}, q_{t-1}, q_t)$  denote a left infinite sequence obtained by truncating  $(\dots, q_{t-1}, q_t, q_{t+1}, \dots)$  at  $t$ . For any input  $u [-\infty: t]$  that has been used to drive the system (for an infinitely distant past until the time  $t$ ), and for any two trajectories  $x[-\infty: t]$  and  $x'[-\infty: t]$  that are consistent with the dynamic mapping (4), it must be that  $x_t = x'_t$ .

Admittedly, the above definition is hard to appreciate for someone not versed in rigorous mathematical thinking. Essentially the definition states that the state of the system should only depend the trajectory in the input space (i.e., depending on which input sequences has been applied). The definition states that there exists an echo state function  $E$  such that

$$x_t = E(u_t, u_{t-1}, u_{t-2}, \dots) \quad (9)$$

for every time instance. Note that in general one would have to assume the following form of the above equation

$$x_t = E(u_t, u_{t-1}, u_{t-2}, \dots | \xi) \quad (10)$$

where the initial condition in the infinite past has to be explicitly included. Equation (9) states that the information stored in the initial condition is washed out as the system evolves in time. Expressed in the filter notation, the echo state requirement (9) is equivalent to saying that there should exist a filter  $\mathcal{F}$  such that Eq. (7) holds with  $\mathcal{F}$  instead of  $\mathcal{R}$ . In other words, the dynamical system realized by Eq. (8) should realize a filter. This is a nontrivial requirement. This behavior cannot be taken for granted. This condition has been carefully analyzed in the original publication (Jaeger 2010).

*A closely related feature to the echo state is the fading memory property.* A system has the fading memory property if the influence of the initial condition on the dynamics weakens with time. Thus such a system should not be chaotic. It has been shown by Jaeger (2010) that the fading memory property implies the echo state property. The converse seems harder to show as argued in Konkoli (2016).

Instead of discussing rather complex mathematical procedures in the original publication, it is easier to illustrate the echo state condition by providing a few examples where this requirement does not hold. The pedagogical examples listed below were taken from Konkoli (2016):

**Example 1** Consider a system with the following update rule:

$$\mathcal{R}(x_{t-1}, u_t) = x_{t-1} \quad (11)$$

The parameterized filter realized by the system is simply given by

$$\mathcal{R}[u|\xi](t) = \xi \quad (12)$$

and clearly the echo state requirement expressed as a filter identity, given in Eq. (7), does not hold. The system never forgets the initial condition. In fact, it is insensitive to the input  $u$ .

One might be tempted to think that the important thing to observe in this example is that two different drive sequences  $u$  and  $u'$  exist that result in the same state. In fact, this could even hold for a system with the echo-state property. While true for this particular system, this is not the right way to think about the echo state. *The key problem this example demonstrates is that two separate trajectories exist for a given fixed input  $u$ .* To realize such a situation, it is sufficient to start the system from a different initial condition.

**Example 2** Consider any system that has at least two quasi stable states with two (or more) basins of attraction. Let us call them  $B_1$  and  $B_2$ . Here the term “quasi” indicates that the states are easily “disturbed” by the external input.

Such a system can be obtained by considering a particle that moves in the potential with two valleys. By assumption, the input to the system consists of the external force applied to the particle. The dynamical equations of such a system, i.e., the explicit form of  $\mathcal{R}$  in (4) could be obtained by starting from the Newton equation, assuming the overdamped regime (where the acceleration term can be ignored) and implementing the explicit Euler integration scheme with a time increment  $\Delta t$ , and finally set  $\tau \equiv \Delta t$ . Such a model is the one studied in the first year or undergraduate studies in physics curriculum. The mathematical side of it will not be discussed in here. However, the behavior of the model is one that interests us and illustrates an important point.

The key feature of the dynamics is that for “weak” inputs the system never crosses the basins of attraction  $B_1$  and  $B_2$ . Transitions from one basin of attraction to the other can only happen under

the influence of “strong” inputs. This implies that, for *weak* inputs, it is possible to find a weak input  $u$ , under which the system can adopt two different states, depending on where it started, i.e., in which basin of attraction the particle was positioned in the infinite past. Thus the system under consideration does not exhibit the echo-state property.

### Liquid State Machines

The Liquid State Machine (LSM) model emphasizes a slightly different perspective. While the Echo State model focuses on the existence of the filter, the LSM model analyzes the expressive power of reservoir computing. It addresses the question: How much can one compute with a given reservoir?

Liquid State Machine is a model of computation with, in principle, the expressive power of the Turing machine. This result follows from almost a direct application of the Stone-Weierstrass approximation theorem in its most general setting. The theorem strongly emphasizes the concept of the algebra of functions, and in some sense the LSM model is a direct translation of the theorem in the reservoir computing language. The consequences of this direct translation are profound.

**The Expressive Power Statement** The mathematical analysis reveals the most important property of the reservoir. *To achieve universal computation in the fading memory sense, the dynamical system used as reservoir must separate inputs.* This feature is referred as the separability condition. A reservoir that has this property is said to separate points.

The separation requirement is defined as follows. Consider a fixed observation time  $t$ . It can be chosen at will, but once chosen it cannot be changed. It is crucial to keep in mind that a reference time has been chosen and is implicit in the following discussion. After reading the introduction section, the observant reader might conclude that the separability criterion should be stated as follows: When two different inputs are applied the system adopts two different states. Surprisingly, this is not the claim. For example, it is possible that for two different inputs the system

accidentally attains the same state at the time  $t$ . Instead, the definition is as follows.

**The Separability Condition** The reservoir  $\mathcal{R}$  separates inputs when for any two inputs  $u_1$  and  $u_2$  that are different in the time interval  $(-\infty, t]$ , a reservoir  $\mathcal{R}_{12}$  can be found such that  $\mathcal{R}_{12}[u_1](t) \neq \mathcal{R}_{12}[u_2](t)$ . In this definition two sequences are considered to be different in the interval if they differ at least in one position.

The above definition sounds strange, since one is intuitively assuming that there is only one reservoir to choose from. And indeed, the claim of the universal expressive power statement does not automatically hold for one reservoir. *In the expressive power statement, it is implicitly assumed that the reservoir is complex enough so that it can be separated into smaller reservoirs.* This is precisely the reason why complex systems are good reservoirs!

### Future Directions

In the literature Echo State Networks and Liquid State Machines are often treated as one concept. Admittedly, they are strongly related but they are not identical. It is possible that in the future the context-dependent differences will emerge as more applications are considered. For example, the question of the computing capacity of an arbitrary but fixed reservoir is still largely open. Given a reservoir, how much can it compute? Especially from a practical point of view such a question is still very challenging. In fact, there still seems to be no theory of reservoir computing that might be directly relevant for engineering applications. Clearly, the pioneering efforts formalized in the ESN and the LSM models need significant improvements. There are several reasons for that.

The LSM model rests on the shoulders of the Stone-Weierstrass approximation theorem which is clearly a mathematical construct. Because of that, stated in a rigorous mathematical language, the separability condition is rather absolute. Two sequences differ even when they differ by a tiny amount (no matter how small). What if a reservoir cannot be found that separates such sequences

with that very small difference, but exists when larger differences are considered? Thus it could be useful to rephrase the separability criterion in less absolute terms, e.g., by requiring not strict separation of inputs, but separation up to a certain accuracy (resolution).

*The question is whether it is possible to formulate an equally generic separability condition that would allow for some error margins, resolution criteria, or provide some bounds.* This would be much more useful for engineering purposes. Such efforts would greatly enhance functional analysis and the associated mathematical disciplines. The problem of a suitable approximation is probably as old as mathematics itself. To a mathematician, an interesting question, perhaps, might be: given that a class of functions forms an algebra and separates inputs up to a certain accuracy, is there any way to characterize a class of functions that can be represented that way? Which types of theorems could be proven?

Another important aspect, perhaps the most important of all, is that the interplay between the complexity of the reservoir and the readout layer has not been extensively explored. It is not obvious that this is an important issue, but it might be. For example, it is possible that one cannot successfully answer the question on the expressive power without carefully analyzing this interplay. To appreciate this possibility, we turn to Hilary Putnam and his construction of how to use *any* object to implement *any* finite state automaton (Putnam 1988).

Putnam used the construction to show that the notion of computation needs to be restrained when discussing philosophy of mind. An ability to compute does not imply intelligence. The statement is a paradox: even a rock has an intrinsic ability to compute anything. There is an intriguing relation between reservoir computing and Putnam's construction (Konkoli 2015).

The construction by Putnam could be used to turn any object into a universal computer, and the expressive power issue that reservoir computing aims to address would be solved once and for all. For example, if a rock can be made to compute anything, a more complex system surely could do the same. Putnam's construct is essentially a

paradox. This paradox points to the importance of the interplay between the complexity of the system and the readout layer. As argued in Konkoli (2015) and references therein, what makes the rock a bad reservoir is the fact that the complexity and presumably the associated engineering burden of building a suitable readout layer is overwhelming.

The above discussion further implies that it is very important to understand how to interface the reservoir properly, and how to mathematically describe the interface and gauge its computing power versus the computing power of the reservoir. *It is very likely that such efforts would impact various engineering disciplines that target information processing applications.*

In general, due to its obvious interdisciplinary potential, it is surprising that reservoir computing does not attract researches from communities such as physics, biology, or chemistry. For example, it is clear that understanding the dynamic behavior of an arbitrary dynamical system is of paramount importance in the reservoir computing context. There is an obvious broad horizon of research targets that practitioners in complex dynamical systems might find attractive. Further, a possible use of living organisms for biomimetic information processing applications is an area of science that might greatly benefit from the insights that have shaped the field of reservoir computing. It is possible that the field of reservoir computing will serve as a bridge between computer science and all the above disciplines, providing a necessary focus in terms of the relevant questions posed, and the language or terminology used to address the posed questions.

## Bibliography

### Primary Literature

- Appeltant L, Soriano MC, Van der Sande G, Danckaert J, Massar S, Dambre J, Schrauwen B, Mirasso CR, Fischer I (2011) Information processing using a single dynamical node as complex system. Nat Commun 2:468  
 Bennett C, Jesorka A, Wendin G, Konkoli Z (2016) On the inverse pattern recognition problem in the context of the time-series data processing with memristor

- networks. In: Adamatzky J (ed) Advances in Unconventional Computation. Springer, Heidelberg
- Boyd S, Chua LO (1985) Fading memory and the problem of approximating nonlinear operators with volterra series. *IEEE Trans Circuits Syst* 32:1150–1161
- Carbajal JP, Dambre J, Hermans M, Schrauwen B (2015) Memristor models for machine learning. *Neural Comput* 27:725–747
- Dambre J, Verstraeten D, Schrauwen B, Massar S (2012) Information processing capacity of dynamical systems. *Sci Rep* 2:514
- Goudarzi A, Stefanovic D (2014) Towards a calculus of echo state networks. *Procedia Comput Sci* 41:176–181
- Jaeger H (2001) The “echo state” approach to analysing and training recurrent neural networks. Technical Report GDM Report 148 (contains errors), German national research center for information technology
- Jaeger H (2010) The “echo state” approach to analysing and training recurrent neural networks – with an erratum note. Technical Report erratum to GDM Report 148, German national research center for information technology
- Jaeger H, Haas H (2004) Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science* 304(5667):78–80
- Joslin D (2006) Real realization: Dennetts real patterns versus putnams ubiquitous automata. *Mind Mach* 16:29–41
- Kirby K (2009) NACAP 2009 extended abstract: Putnamizing the liquid state. Bloomington, Indiana
- Konkoli Z (2015) A perspective on Putnam’s realizability theorem in the context of unconventional computation. *Int J Unconv Comput* 11:83–102
- Konkoli Z (2016) On reservoir computing: from mathematical foundations to unconventional applications. In: Adamatzky A (ed) Advances in unconventional computing, volume 1. Theory. Springer, Heidelberg
- Konkoli Z, Wendin G (2014) On information processing with networks of nanoscale switching elements. *Int J Unconv Comput* 10(5–6):405–428
- Kulkarni MS, Teuscher C (2012) Memristor-based reservoir computing. In: IEEE/ACM international symposium on Nanoscale Architectures (NANOARCH), ACM, New York, pp 226–232
- Ladyman J (2009) What does it mean to say that a physical system implements a computation? *Theor Comput Sci* 410:376–383
- Larger L, Soriano MC, Brunner D, Appeltant L, Gutierrez JM, Pesquera L, Mirasso CR, Fischer I (2012) Photonic information processing beyond Turing: an optoelectronic implementation of reservoir computing. *Opt Express* 20(3):3241–3249
- Lukočevičius M, Jaeger H (2009) Reservoir computing approaches to recurrent neural network training. *Comput Sci Rev* 3(3):127–149
- Lukoševičius M, Jaeger H, Schrauwen B (2012) Reservoir computing trends. *KI – Knstliche Intelligenz* 26(4):365–371
- Maass W, Natschläger T, Markram H (2002) Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput* 14(11):2531–2560
- Massar M, Massar S (2013) Mean-field theory of echo state networks. *Phys Rev E* 87:042809
- Mesaritakis C, Bogris A, Kapsalis A, Syvridis D (2015) High-speed all-optical pattern recognition of dispersive Fourier images through a photonic reservoir computing subsystem. *Opt Lett* 40:3416–3419
- Natschläger T, Maass W, Markram H (2002) The “liquid computer”: A novel strategy for realtime computing on time series (special issue on foundations of information processing). *TELEMATIK* 8:39–43
- Putnam H (1988) Representation and reality. MIT Press, Cambridge
- Soriano MC, Brunner D, Escalona-Moran M, Mirasso CR, Fischer I (2015) Minimal approach to neuro-inspired information processing. *Front Comput Neurosci* 9:68

## Books and Reviews

- Konkoli (2016); Joslin (2006); Kirby (2009); Boyd and Chua (1985); Putnam (1988); Ladyman (2009)



---

## Unconventional Computational Problems

Selim G. Akl

School of Computing and Department of Mathematics and Statistics, Queen's University, Kingston, ON, Canada

### Article Outline

Glossary

Introduction

Unconventional Computations

Future Directions

Bibliography

### Glossary

**Processor** In a computer, the component in charge of executing the operations of an algorithm.

**Time unit** The length of time required by a processor to perform a *step* of its computation, consisting of three elementary operations: a *read* operation in which it receives a constant number of fixed-size data as input, a *calculate* operation in which it performs a fixed number of constant-time *arithmetic* and *logical* calculations (such as adding two numbers, comparing two numbers, and so on), and a *write* operation in which it returns a constant number of fixed-size data as output.

**Sequential computer** Consists of a single processor.

**Parallel computer** Has  $n$  processors, numbered 1 to  $n$ , where  $n \geq 2$ . Both computers use the same type of processor, and that processor is the fastest possible (Akl 1997). The assumption that the computers on hand, whether sequential or parallel, use the fastest conceivable processor is an important one. This is because the speed of the processor used is what specifies the duration of a

time unit; the faster the processor, the smaller the time unit.

**Principle of simulation** States that any computation that can be performed successfully on a general-purpose computer  $A$  can be performed more or less efficiently but still successfully on another general-purpose computer  $B$  (Harel 1992; Lewis and Papadimitriou 1981; Mandrioli and Ghezzi 1987; Minsky 1967).

**Principle of universality** Follows directly from the principle of simulation and states that there exists a *universal computer*  $U$ , with fixed specifications, capable of performing successfully any computation that can be performed on any other computer (Abramsky et al. 1992; Davis 2000; Denning et al. 1978; Hillis 1998; Hopcroft and Ullman 1969).

**Unconventional computational problem** A computation requiring  $n$  algorithmic steps per time unit, where  $n$  is larger than 1. Such a computation cannot be performed successfully on a computer that can only perform a finite and fixed number  $m$  of algorithmic steps per time unit, where  $m$  is smaller than  $n$  (Akl 2005; Akl and Salay 2015; Burgin 2005; Calude and Paun 2005; Copeland 1998; Denning 2012; Deutsch 1997; Etesi and Németi 2002; Goldin and Wegner 2005; Nagy and Akl 2005, 2012; Siegelmann 1999; Stepney 2004; Toffoli 1982).

### Introduction

The importance of unconventional computations stems from their ability to provide exceptions to the principles of simulation and universality:

1. There exist unconventional computational problems that are demonstrably solvable on a computer capable of  $n$  algorithmic steps per time unit, such as, for example, a parallel computer with  $n$  processors. Simulation of the latter solution on a computer capable of fewer than  $n$  algorithmic steps per time unit is demonstrably impossible.

2. No computer capable of a finite and fixed number of algorithmic steps per time unit can be universal.

Examples of unconventional computational problems are presented in the following section.

## Unconventional Computations

Seven computational paradigms are described in what follows to illustrate the class of unconventional computational problems. Implications of these paradigms are then presented.

### Unconventional Computational Paradigms

Each of the computations described in what follows can be characterized as being *inherently parallel*, due to the fact that it is executed successfully only on a computer capable of  $n$  algorithmic steps per time unit, a feature enjoyed by a parallel computer with  $n$  processors.

**Computations Obeying Mathematical Constraints**  
 There exists a family of computational problems where, given a mathematical object satisfying a certain property, we are asked to transform this object into another which also satisfies the same property. Furthermore, the property is to be maintained throughout the transformation, and be satisfied by every intermediate object, if any. More generally, the computations we consider here are such that every step of the computation must obey a certain predefined mathematical constraint. (Analogies from popular culture include picking up sticks from a heap one by one without moving the other sticks, drawing a geometric figure without lifting the pencil, and so on.)

An example of computations obeying a mathematical constraint is provided by a variant to the problem of sorting a sequence of numbers stored in the memory of a computer. For a positive even integer  $n$ , where  $n \geq 8$ , let  $n$  distinct integers be stored in an array  $A$  with  $n$  locations  $A[1], A[2], \dots, A[n]$ , one integer per location. Thus  $A[j]$ , for all  $1 \leq j \leq n$ , represents the integer currently stored in the  $j$ th location of  $A$ . It is

required to sort the  $n$  integers in place into increasing order, such that:

1. After step  $i$  of the sorting algorithm, for all  $i \geq 1$ , no three consecutive integers satisfy

$$A[j] > A[j+1] > A[j+2], \quad (1)$$

for all  $1 \leq j \leq n - 2$ .

2. When the sort terminates we have

$$A[1] < A[2] < \dots < A[n]. \quad (2)$$

This is the standard sorting problem in computer science, but with a twist. In it, the journey is more important than the destination. While it is true that we are interested in the outcome of the computation (namely, the sorted array, this being the *destination*), in this particular variant we are more concerned with *how* the result is obtained (namely, there is a condition that must be satisfied throughout all steps of the algorithm, this being the *journey*). It is worth emphasizing here that the condition to be satisfied is germane to the problem itself; specifically, there are no restrictions whatsoever on the model of computation or the algorithm to be used. Our task is to find an algorithm for a chosen model of computation that solves the problem exactly as posed. One should also observe that computer science is replete with problems with an inherent condition on how the solution is to be obtained. Examples of such problems include inverting a nonsingular matrix without ever dividing by zero, finding a shortest path in a graph without examining an edge more than once, sorting a sequence of numbers without reversing the order of equal inputs (stable sorting), and so on.

An *oblivious* (that is, input-independent) algorithm for an  $n/2$ -processor parallel computer solves the aforementioned variant of the sorting problem handily in  $n$  steps, by means of predefined pairwise swaps applied to the input array  $A$ , during each of which  $A[j]$  and  $A[k]$  exchange positions (using an additional memory location for temporary storage) (Akl 1997). An input-dependent algorithm succeeds on a computer with  $(n/2) - 1$  processors. However, a sequential computer, and a parallel computer with fewer than

$(n/2) - 1$  processors, both fail to solve the problem consistently, that is, they fail to sort all possible  $n!$  permutations of the input while satisfying, at every step, the condition that no three consecutive integers are such that  $A[j] > A[j+1] > A[j+2]$  for all  $j$ . In the particularly nasty case where the input is of the form

$$A[1] > A[2] > \dots > A[n], \quad (3)$$

any sequential algorithm and any algorithm for a parallel computer with fewer than  $(n/2) - 1$  processors fail after the first swap.

#### Time-Varying Computational Complexity

Here, the computational complexity of the problems at hand depends on *time* (rather than being, as usual, a function of the problem *size*). Thus, for example, tracking a moving object (such as a spaceship racing toward Mars) becomes harder as it travels away from the observer.

Suppose that a certain computation requires that  $n$  independent functions, each of one variable, namely,  $f_1(x_1), f_2(x_2), \dots, f_n(x_n)$ , be computed. Computing  $f_i(x_i)$  at time  $t$  requires  $C(t) = 2^t$  algorithmic steps, for  $t \geq 0$  and  $1 \leq i \leq n$ . Further, there is a strict deadline for reporting the results of the computations: All  $n$  values  $f_1(x_1), f_2(x_2), \dots, f_n(x_n)$  must be returned by the end of the third time unit, that is, when  $t = 3$ .

It should be easy to verify that no sequential computer, capable of exactly one algorithmic step per time unit, can perform this computation for  $n \geq 3$ . Indeed,  $f_1(x_1)$  takes  $C(0) = 2^0 = 1$  time unit,  $f_2(x_2)$  takes another  $C(1) = 2^1 = 2$  time units, by which time three time units would have elapsed. At this point none of  $f_3(x_3), \dots, f_n(x_n)$  would have been computed. By contrast, an  $n$ -processor parallel computer solves the problem handily. With all processors operating simultaneously, processor  $i$  computes  $f_i(x_i)$  at time  $t = 0$ , for  $1 \leq i \leq n$ . This consumes one time unit, and the deadline is met.

#### Rank-Varying Computational Complexity

Suppose that a computation consists of  $n$  stages. There may be a certain precedence among these stages, or the  $n$  stages may be totally independent, in which case the order of execution is of no

consequence to the correctness of the computation. Let the *rank* of a stage be the order of execution of that stage. Thus, stage  $i$  is the  $i$ th stage to be executed. Here we focus on computations with the property that the number of algorithmic steps required to execute stage  $i$  is  $C(i)$ , that is, a function of  $i$  only.

When does rank-varying computational complexity arise? Clearly, if the computational requirements grow with the rank, this type of complexity manifests itself in those circumstances where it is a disadvantage, whether avoidable or unavoidable, to being  $i$ th, for  $i \geq 2$ . For example, the precision and/or ease of measurement of variables involved in the computation in a stage  $s$  may decrease with each stage executed before  $s$ .

The same analysis as in section “[Time-Varying Computational Complexity](#)” applies by substituting the rank for the time.

#### Time-Varying Variables

For a positive integer  $n$  larger than 1, we are given  $n$  functions, each of one variable, namely,  $f_1, f_2, \dots, f_n$ , operating on the  $n$  physical variables  $x_1, x_2, \dots, x_n$ , respectively. Specifically, it is required to compute  $f_i(x_i)$ , for  $i = 1, 2, \dots, n$ . For example,  $f_i(x_i)$  may be equal to  $x_i^2$ . What is unconventional about this computation is the fact that the  $x_i$  are themselves (unknown) functions  $x_1(t), x_2(t), \dots, x_n(t)$  of the time variable  $t$ . It takes one time unit to evaluate  $f_i(x_i(t))$ . The problem calls for computing  $f_i(x_i(t))$ ,  $1 \leq i \leq n$ , at time  $t = t_0$ . Because the function  $x_i(t)$  is unknown, it cannot be inverted, and for  $k > 0$ ,  $x_i(t_0)$  cannot be recovered from  $x_i(t_0 + k)$ . Note that the value of an input variable  $x_i(t)$  changes at the same speed as the processor in charge of evaluating the function  $f_i(x_i(t))$ .

A sequential computer fails to compute all the  $f_i$  as desired. Indeed, suppose that  $x_1(t_0)$  is initially operated upon. By the time  $f_1(x_1(t_0))$  is computed, one time unit would have passed. At this point, the values of the  $n - 1$  remaining variables would have changed. The same problem occurs if the sequential computer attempts to first read all the  $x_i$ , one by one, and store them before calculating the  $f_i$ .

By contrast, a parallel computer consisting of  $n$  independent processors may perform all the computations at once: For  $1 \leq i \leq n$ , and all

processors working at the same time, processor  $i$  computes  $f_i(x_i(t_0))$ , leading to a successful computation.

### Interacting Variables

A physical system has  $n$  variables,  $x_1, x_2, \dots, x_n$ , each of which is to be measured or set to a given value at regular intervals. One property of this system is that measuring or setting one of its variables modifies the values of any number of the system variables uncontrollably, unpredictably, and irreversibly.

A sequential computer measures *one* of the values ( $x_1$ , for example) and by so doing it disturbs an unknowable number of the remaining variables, thus losing all hope of recording the state of the system within the given time interval. Similarly, the sequential approach cannot update the variables of the system properly: Once  $x_1$  has received its new value, setting  $x_2$  may disturb  $x_1$  in an uncertain way.

A parallel computer with  $n$  processors, by contrast, will measure *all* the variables  $x_1, x_2, \dots, x_n$  simultaneously (one value per processor), and therefore obtain an accurate reading of the state of the system within the given time frame. Consequently, new values  $x_1, x_2, \dots, x_n$  can be computed in parallel and applied to the system simultaneously (one value per processor).

### Uncertain Time Constraints

In this paradigm, we are given a computation consisting of three distinct phases, namely, input, calculation, and output, each of which needs to be completed by a certain deadline. However, unlike the standard situation in conventional computation, the deadlines here are not known at the outset. In fact, and this is what makes this paradigm truly unconventional, we do not know at the moment the computation is set to start, *what* needs to be done, and *when* it should be done. Certain physical parameters, from the external environment surrounding the computation, become spontaneously available. The values of these parameters, once received from the outside world, are then used to evaluate two functions,  $f_1$  and  $f_2$ , which tell us precisely *what* to do and *when* to do it, respectively.

The difficulty posed by this paradigm is that the evaluation of the two functions  $f_1$  and  $f_2$  is

itself quite demanding computationally. Specifically, for a positive integer  $n$ , the two functions operate on  $n$  variables (the physical parameters). Only a parallel computer equipped with  $n$  processors can succeed in evaluating the two functions on time to meet the deadlines.

### The Global Variable Paradigm

A computation  $C_0$  consists of two distinct and separate processes  $P_0$  and  $P_1$  operating on a global variable  $x$ . The variable  $x$  is *time critical* in the sense that its value throughout the computation is intrinsically related to real (external or physical) time. Actions taken throughout the computation, based on the value of  $x$ , depend on  $x$  having that particular value at that particular time. Here, time is kept internally by a global clock. Specifically, the computer performing  $C_0$  has a clock that is synchronized with real time. Henceforth, real time is synonymous with internal time. In this framework, therefore, resetting  $x$  artificially, through simulation, to a value it had at an earlier time is entirely insignificant, as it fails to meet the true timing requirements of  $C_0$ . At the beginning of the computation,  $x = 0$ .

Let the processes of the computation  $C_0$ , namely,  $P_0$  and  $P_1$ , be as follows:

```
 $P_0$ : if  $x = 0$  then  $x \leftarrow x + 1$  else loop forever  
end if.
```

```
 $P_1$ : if  $x = 0$  then read  $y$ ;  $x \leftarrow x + y$ ; return  $x$   
else loop forever end if.
```

In order to better appreciate this simple example, it is helpful to put it in some familiar context. Think of  $x$  as the altitude of an airplane and think of  $P_0$  and  $P_1$  as software controllers actuating safety procedures that must be performed at this altitude. The local nonzero variable  $y$  is an integral part of the computation; it helps to distinguish between the two processes and to separate their actions.

The question now is this: on the assumption that  $C_0$  succeeds, that is, that both  $P_0$  and  $P_1$  execute the “**then**” part of their respective “**if**” statements (not the “**else**” part), what is the value of the global variable  $x$  at the end of the computation, that is, when both  $P_0$  and  $P_1$  have halted?

We examine two approaches to executing  $P_0$  and  $P_1$ :

1. **Using a single processor:** Consider a sequential computer equipped, by definition, with a single processor  $p_0$ . The processor executes one of the two processes first. Suppose it starts with  $P_0$ :  $p_0$  computes  $x = 1$  and terminates. It then proceeds to execute  $P_1$ . Because now  $x \neq 0$ ,  $p_0$  executes the nonterminating computation in the “else” part of the “if” statement. The process is uncomputable and the computation fails. Note that starting with  $P_1$  and then executing  $P_0$  would lead to a similar outcome, with the difference being that  $P_1$  will return an incorrect value of  $x$ , namely,  $y$ , before switching to  $P_0$ , whereby it executes a nonterminating computation, given that now  $x \neq 0$ .
2. **Using two processors:** The two processors, namely,  $p_0$  and  $p_1$ , are part of a shared-memory parallel computer in which two or more processors can read from, but not write to, the same memory location simultaneously (Akl 1997). In parallel,  $p_0$  executes  $P_0$  and  $p_1$  executes  $P_1$ . Both terminate successfully and return the correct value of  $x$ , that is,  $x = y + 1$ .

Two observations are in order:

1. The first concerns the sequential (that is, single-processor) solution. Here, no ex post facto simulation is possible or even meaningful. This includes legitimate simulations, such as executing one of the processes and then the other, or interleaving their executions, and so on. It also includes illegitimate simulations, such as resetting the value of  $x$  to 0 after executing one of the two processes, or (assuming this is feasible) an ad hoc rewriting of the code, as, for example,

```
if  $x = 0$  then  $x \leftarrow x + 1$ ; read  $y$ ;  $x \leftarrow x + y$ ;
return  $x$ 

else loop forever

end if.
```

and so on. To see this, note that for either  $P_0$  or  $P_1$  to terminate, the **then** operations of its **if** statement must be executed *as soon as* the global variable  $x$  is found to be equal to 0, and not one time unit later. It is clear that any sequential simulation must be seen to have failed. Indeed:

- A legitimate simulation will not terminate, because for one of the two processes,  $x$  will no longer be equal to 0, while
  - An illegitimate simulation will “terminate” illegally, having executed the “**then**” operations of one or both of  $P_0$  or  $P_1$  too late.
2. The second observation follows directly from the first. It is clear that  $P_0$  and  $P_1$  must be executed simultaneously for a proper outcome of the computation. The parallel (that is, two-processor) solution succeeds in accomplishing exactly this.

A word about the role of time. Real time, as mentioned earlier, is kept by a global clock and is equivalent to internal computer time. It is important to stress here that the time variable is never used explicitly by the computation  $C_0$ . Time intervenes only in the circumstance where it is needed to signal that  $C_0$  has failed (when the “else” part of an “if” statement, either in  $P_0$  or in  $P_1$ , is executed). In other words, time is noticed solely when time requirements are neglected.

To generalize the global variable paradigm, we assume the presence of  $n$  global variables, namely,  $x_0, x_1, \dots, x_{n-1}$ , all of which are time critical, and all of which are initialized to 0. There are also  $n$  nonzero local variables, namely,  $y_0, y_1, \dots, y_{n-1}$ , belonging, respectively, to the  $n$  processes  $P_0, P_1, \dots, P_{n-1}$  that make up  $C_1$ . The computation  $C_1$  is as follows:

```
 $P_0$ : if  $x_0 = 0$  then  $x_1 \leftarrow y_0$  else loop forever
end if.

 $P_1$ : if  $x_1 = 0$  then  $x_2 \leftarrow y_1$  else loop forever
end if.

 $P_2$ : if  $x_2 = 0$  then  $x_3 \leftarrow y_2$  else loop forever
end if.
```

:

$P_{n-2}$ : **if**  $x_{n-2} = 0$  **then**  $x_{n-1} \leftarrow y_{n-2}$  **else**  
loop forever **end if**.

$P_{n-1}$ : **if**  $x_{n-1} = 0$  **then**  $x_0 \leftarrow y_{n-1}$  **else** loop  
forever **end if**.

Suppose that the computation  $C_1$  begins when  $x_i = 0$ , for  $i = 0, 1, \dots, n - 1$ . For every  $i$ ,  $0 \leq i \leq n - 1$ , if  $P_i$  is to be completed successfully, it must be executed, while  $x_i$  is indeed equal to 0, and not at any later time when  $x_i$  has been modified by  $P_{(i-1) \bmod n}$  and is no longer equal to 0. On a parallel computer with  $n$  processors, namely,  $p_0, p_1, \dots, p_{n-1}$ , it is possible to test all the  $x_i$ ,  $0 \leq i \leq n - 1$ , for equality to 0 in one time unit; this is followed by assigning to all the  $x_i$ ,  $0 \leq i \leq n - 1$ , their new values during the next time unit. Thus all the processes  $P_i$ ,  $0 \leq i \leq n - 1$ , and hence the computation  $C_1$ , terminate successfully. A sequential computer has but a single processor  $p_0$  and, as a consequence, it fails to meet the time-critical requirements of  $C_1$ . At best, it can perform no more than  $n - 1$  of the  $n$  processes as required (assuming it executes the processes in the order  $P_{n-1}, P_{n-2}, \dots, P_1$ , then fails at  $P_0$  since  $x_0$  was modified by  $P_{n-1}$ ), and thus does not terminate. A parallel computer with only  $n - 1$  processors,  $p_0, p_1, \dots, p_{n-2}$ , cannot do any better. At best, it too will attempt to execute at least one of the  $P_i$  when  $x_i \neq 0$  and hence fail to complete at least one of the processes on time.

Finally, and most importantly, even a computer capable of an *infinite* number of algorithmic steps per time unit (like an accelerating machine (Fraser and Akl 2008) or, more generally, a Supertask Machine (Davies 2001; Earman and Norton 1996; Steinhart 2007)) would fail to perform the computations required by the global variable paradigm if it were restricted to execute these algorithmic steps *sequentially*.

### Implications

Each of the computational problems described in section “[Unconventional Computational Paradigms](#)” can be readily solved on a computer

capable of executing  $n$  algorithmic steps per time unit but fails to be executed on a computer capable of fewer than  $n$  algorithmic steps per time unit. Furthermore, the problem size  $n$  itself is a variable that changes with each problem instance. As a result, *no* computer, regardless of how many algorithmic steps it can perform in one time unit, can cope with a growing problem size, as long as it obeys the “finiteness condition”, that is, as long as the number of algorithmic steps it can perform per time unit is finite and fixed. This observation leads to a theorem that there does not exist a *finite* computational device that can be called a universal computer. The proof of this theorem proceeds as follows. Suppose there exists a universal computer capable of  $n$  algorithmic steps per time unit, where  $n$  is a finite and fixed integer. This computer will fail to perform a computation *requiring*  $n'$  algorithmic steps per time unit, for any  $n' > n$ , and consequently lose its claim of universality. Naturally, for each  $n' > n$ , another computer capable of  $n'$  algorithmic steps per time unit will succeed in performing the aforementioned computation. However, this new computer will in turn be defeated by a problem requiring  $n'' > n'$  algorithmic steps per time unit. This holds even if the computer purporting to be universal is endowed with an unlimited memory and is allowed to compute for an indefinite amount of time.

The only constraint that is placed on the computer (or model of computation) that aspires to be universal is that the number of operations of which it is capable per time unit be finite and fixed once and for all. In this regard, it is important to note that:

1. The requirement that the number of operations per time unit, or step, be *finite* is necessary for any “reasonable” model of computation; see, for example, (Sipser 1997), p. 141.
2. The requirement that this number be *fixed* once and for all is necessary for any model of computation that claims to be “universal”; see, for example, (Deutsch 1997), p. 210.

These two requirements are fundamental to the theory of computation in general, and to the theory of algorithms, in particular.

It should be noted that computers obeying the finiteness condition include all “reasonable” models of computation, both theoretical and practical, such as the Turing machine, the random access machine, and other idealized models (Savage 1998), as well as all of today’s general-purpose computers, including existing conventional computers (both sequential and parallel), and contemplated unconventional ones such as biological and quantum computers (Akl 2006a). It is true for computers that interact with the outside world in order to read input and return output (unlike the Turing Machine, but like every realistic general-purpose computer). It is also valid for computers that are given unbounded amounts of time and space in order to perform their computations (like the Turing Machine, but unlike realistic computers). Even accelerating machines that increase their speed at every step (such as doubling it, or squaring it, or any such fixed acceleration) at a rate that is set in advance, cannot be universal.

As a result, it is possible to conclude that the only possible universal computer would be one capable of an infinite number of algorithmic steps per time unit *executed in parallel*.

In fact, this work has led to the discovery of computations that can be performed on a quantum computer but that cannot, even in principle, be performed on any classical computer (even one with infinite resources), thus showing for the first time that the class of problems solvable by classical means is a true subset of the class of problems solvable by quantum means (Nagy and Akl 2007a). Consequently, the only possible universal computer would have to be quantum (as well as being capable of an infinite number of algorithmic steps per time unit *executed in parallel*).

## Future Directions

Some approaches have been proposed in an attempt to meet the challenges to universality posed by unconventional computational problems. For example, mathematical logic has been used to address the time-varying variables paradigm (Bringsjord 2017). On another front (Akl

2010), closed timelike curves seem to overcome the difficulties of some of the problems in section “[Unconventional Computational Paradigms](#)”, but not all. In particular, computations that are subject to mathematical constraints appear to have withstood all attacks. It is important to note that in order to salvage the principle of universality, all unconventional computational problems listed in section “[Unconventional Computational Paradigms](#)” (not just some) need to be shown solvable by a computer that obeys the finiteness condition. Alternatively, it appears that the only way to proceed may be to work on developing a universal computer such as the one described in the closing sentence of the previous section (Davies 2001).

## Bibliography

### (A) Primary Literature

- Abramsky S et al (1992) Handbook of logic in computer science. Clarendon Press, Oxford
- Akl SG (1997) Parallel computation: models and methods. Prentice Hall, Upper Saddle River
- Akl SG (2005) The myth of universal computation. In: Trobec R, Zinterhof P, Vajtersic M, Uhl A (eds) Parallel numerics. University of Salzburg/Jozef Stefan Institute, Salzburg/Ljubljana, pp 211–236
- Akl SG (2006a) Three counterexamples to dispel the myth of the universal computer. Parallel Proc Lett 16:381–403
- Akl SG (2006b) Conventional or unconventional: is any computer universal? In: Adamatzky A, Teuscher C (eds) From utopian to genuine unconventional computers. Luniver Press, Frome, pp 101–136
- Akl SG (2007a) Gödel’s incompleteness theorem and non-universality in computing. In: Nagy M, Nagy N (eds) Proceedings of the workshop on unconventional computational problems. Sixth International Conference on Unconventional Computation, Kingston, pp 1–23
- Akl SG (2007b) Even accelerating machines are not universal. Int J Unconv Comput 3:105–121
- Akl SG (2008a) Unconventional computational problems with consequences to universality. Int J Unconv Comput 4:89–98
- Akl SG (2008b) Evolving computational systems. In: Rajasekaran S, Reif JH (eds) Parallel computing: models, algorithms, and applications. Taylor and Francis, Boca Raton, pp 1–22
- Akl SG (2009) Ubiquity and simultaneity: the science and philosophy of space and time in unconventional computation. Keynote address, Conference on the Science and Philosophy of Unconventional Computing, The University of Cambridge, Cambridge

- Akl SG (2010) Time travel: a new hypercomputational paradigm. *Int J Unconv Comput* 6:329–351
- Akl SG (2014) What is computation? *Int J Parallel Emergent Distrib Syst* 29:337–345
- Akl SG (2016) Nonuniversality explained. *Int J Parallel Emergent Distrib Syst* 31:201–219
- Akl SG (2017) Nonuniversality in computation: fifteen misconceptions rectified. In: Adamatzky A (ed) *Advances in unconventional computing*. Springer, Cham, pp 1–31
- Akl SG Universality in computation: some quotes of interest. Technical report no 2006–511, School of Computing, Queen's University. [http://www.cs.queensu.ca/home/akl\\_techreports/quotes.pdf](http://www.cs.queensu.ca/home/akl_techreports/quotes.pdf)
- Akl SG, Nagy M (2009a) Introduction to parallel computation. In: Trobec R, Vajtersic M, Zinterhof P (eds) *Parallel computing: numerics, applications, and trends*. Springer, London, pp 43–80
- Akl SG, Nagy M (2009b) The future of parallel computation. In: Trobec R, Vajtersic M, Zinterhof P (eds) *Parallel computing: numerics, applications, and trends*. Springer, London, pp 471–510
- Akl SG, Salay N (2015) On computable numbers, non-universality, and the genuine power of parallelism. *Int J Unconv Comput* 11:283–297
- Bringsjord S (2017) Is universal computation a myth? In: Adamatzky A (ed) *Emergent computation: a festschrift for Selim G. Akl*. Springer, Cham, pp 19–37
- Burgin M (2005) Super-recursive algorithms. Springer, New York
- Calude CS, Paun G (2004) Bio-steps beyond Turing. *Biosystems* 77:175–194
- Copeland BJ (1998) Super Turing-machines. *Complexity* 4:30–32
- Davies EB (2001) Building infinite machines. *Br J Philos Sci* 52:671–682
- Davis M (2000) The universal computer. W.W. Norton, New York
- Denning PJ (2012) Reflections on a symposium on computation. *Comput J* 55:799–802
- Denning PJ, Dennis JB, Qualitz JE (1978) Machines, languages, and computation. Prentice-Hall, Englewood Cliffs
- Deutsch D (1997) The fabric of reality. Penguin Books, London
- Earman J, Norton JD (1996) Infinite pains: the trouble with supertasks. In: Morton A, Stich SP (eds) *Benacerraf and his critics*. Blackwell, Cambridge, pp 231–261
- Etesi G, Nemeti I (2002) Non-Turing computations via Malament-Hogarth space-times. *Int J Theor Phys* 41:341–370
- Fraser R, Akl SG (2008) Accelerating machines: a review. *Int J Parallel Emergent Distrib Syst* 23:81–104
- Goldin D, Wegner P (2005) The church-Turing thesis: breaking the myth. In: Cooper BS, Lowe B (eds) *New computational paradigms*. Springer, Berlin, pp 152–168
- Harel D (1992) *Algorithmics: the spirit of computing*. Addison-Wesley, Reading
- Hillis D (1998) *The pattern on the stone*. Basic Books, New York
- Hopcroft JE, Ullman JD (1969) *Formal languages and their relations to automata*. Addison-Wesley, Reading
- Lewis HR, Papadimitriou CH (1981) *Elements of the theory of computation*. Prentice Hall, Englewood Cliffs
- Mandrioli D, Ghezzi C (1987) *Theoretical foundations of computer science*. Wiley, New York
- Minsky ML (1967) *Computation: finite and infinite machines*. Prentice-Hall, Englewood Cliffs
- Nagy M, Akl SG (2005) On the importance of parallelism for quantum computation and the concept of a universal computer. In: Calude CS, Dinneen MJ, Paun G, de Perez-Jimenez, M. J., Rozenberg G (eds) *Unconventional computation*. Springer, Heidelberg, pp 176–190
- Nagy M, Akl SG (2006) Quantum measurements and universal computation. *Int J Unconv Comput* 2:73–88
- Nagy M, Akl SG (2007a) Quantum computing: beyond the limits of conventional computation. *Int J Parallel Emergent Distrib Syst* 22:123–135
- Nagy M, Akl SG (2007b) Parallelism in quantum information processing defeats the universal computer. *Par Proc Lett* 17:233–262
- Nagy N, Akl SG (2011) Computations with uncertain time constraints: effects on parallelism and universality. In: Calude CS, Kari J, Petre I, Rozenberg G (eds) *Unconventional computation*. Springer, Heidelberg, pp 152–163
- Nagy N, Akl SG (2012) Computing with uncertainty and its implications to universality. *Int J Parallel Emergent Distrib Syst* 27:169–192
- Savage JE (1998) *Models of computation*. Addison-Wesley, Reading
- Siegelmann HT (1999) Neural networks and analog computation: beyond the Turing limit. Birkhauser, Boston
- Sipser M (1997) *Introduction to the theory of computation*. PWS Publishing Company, Boston
- Steinhart E (2007) Infinitely complex machines. In: Schuster A (ed) *Intelligent computing everywhere*. Springer, New York, pp 25–43
- Stepney S (2004) The neglected pillar of material computation. *Physica D* 237:1157–1164
- Toffoli T (1982) Physics and computation. *Int J Theor Phys* 21:165–175

## (B) Books and Reviews

- Akl SG (2004) Superlinear performance in real-time parallel computation. *J Supercomput* 29:89–111
- Akl SG Non-universality in computation: the myth of the universal computer. School of Computing, Queen's University. <http://research.cs.queensu.ca/Parallel/projects.html>
- Akl SG A computational challenge. School of computing, Queen's University [http://www.cs.queensu.ca/home/akl/CHALLENGE/A\\_Computational\\_Challenge.htm](http://www.cs.queensu.ca/home/akl/CHALLENGE/A_Computational_Challenge.htm)
- Akl SG, Yao W (2005) Parallel computation and measurement uncertainty in nonlinear dynamical systems. *J Math Model Alg* 4:5–15
- Durand-Lose J (2004) Abstract geometrical computation for black hole computation. Research report no

- 2004–15, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Supérieure de Lyon, Lyon
- Einstein A (2009) Letter to Erwin Schrödinger. In: Gilder L (ed) *The age of entanglement*. Vintage Books, New York, p 170
- Fortnow L The enduring legacy of the Turing machine. <http://ubiquity.acm.org/article.cfm?id=1921573>
- Gleick J (2011) *The information: a history, a theory, a flood*. HarperCollins, London
- Hypercomputation. <http://en.wikipedia.org/wiki/Hypercomputation>
- Kelly K (2002) God is the machine. *Wired* 10. <https://www.wired.com/2002/12/holytech/>
- Kleene SC (1952) *Introduction to metamathematics*. North Holland, Amsterdam
- Lloyd S (2006) *Programming the universe*. Knopf, New York
- Lloyd S, Ng YJ (2004) Black hole computers. *Sci Am* 291:53–61
- Rucker R (2005) *The lifebox, the seashell, and the soul*. Thunder's Mouth Press, New York
- Seife C (2006) *Decoding the universe*. Viking Penguin, New York
- Siegfried T (2000) *The bit and the pendulum*. Wiley, New York
- Stepney S (2004) Journeys in non-classical computation. In: Hoare T, Milner R (eds) *Grand challenges in computing research*. BCS, Swindon, pp 29–32
- Tipler FJ (1995) *The physics of immortality: modern cosmology, God and the resurrection of the dead*. Macmillan, London
- Turing AM (1939) Systems of logic based on ordinals. *Proc London Math Soc* 2 45:161–228
- Vedral V (2010) *Decoding reality*. Oxford University Press, Oxford
- Wegner P, Goldin D (1997) Computation beyond Turing machines. *Comm ACM* 40:100–102
- Wheeler JA (1989) Information, physics, quanta: The search for links. In: Proceedings of the third international symposium on foundations of quantum mechanics in light of new technology, Tokyo, pp 354–368
- Wheeler JA (1990) Information, physics, quantum: the search for links. In: Zurek W (ed) *Complexity, entropy, and the physics of information*. Addison-Wesley, Redwood City
- Wheeler JA (1994) *At home in the universe*. American Institute of Physics Press, Woodbury
- Wolfram S (2002) *A new kind of science*. Wolfram Media, Champaign
- Zuse K (1970) Calculating space. MIT Technical Translation AZT-70-164-GEMIT, Massachusetts Institute of Technology (Project MAC), Cambridge



---

## Algorithmic Cognition and the Computational Nature of the Mind

Hector Zenil<sup>1</sup> and Nicolas Gauvrit<sup>2</sup>

<sup>1</sup>Algorithmic Dynamics Lab, Unit of Computational Medicine and SciLifeLab, Center for Molecular Medicine, Department of Medicine Solna, Karolinska Institutet, Stockholm, Sweden

<sup>2</sup>Human and Artificial Cognition Lab, EPHE, Paris, France

### Article Outline

Glossary

Algorithmic Cognition

Algorithmic Bayesian Estimations as a Model of the Mind

Conclusion

Bibliography

### Glossary

**Algorithmic coding theorem (not to confuse with Shannon's coding theorem)** A theorem that formally establishes an inversely proportional relationship between Kolmogorov-Chaitin complexity and algorithmic probability.

**Algorithmic cognition** The study of animal, human, and artificial cognition based on the theory of algorithmic probability.

**Algorithmic information theory** The literature based on the concept of Kolmogorov-Chaitin complexity and related concepts such as algorithmic probability, compression, optimal inference, the Universal Distribution, and Levin's semi-measure.

**Algorithmic probability** The probability to produce an object from a random digital computer program whose program binary digits are chosen by chance. The calculation of algorithmic probability is a lower semi-computable problem.

**Algorithmic randomness** How removed the length of the shortest generating program is from the size of the uncompressed data that such program generates.

**Block decomposition method (BDM)** A divide and conquer method that chunks a string or multidimensional object into smaller pieces for which better lower bounds of algorithmic probability can be produced.

**Causal gap** The gap between objects that Shannon entropy can identify as non-random and the same objects that estimations to Kolmogorov-Chaitin complexity can identify as non-random which is a much larger set than Shannon entropy.

**Coding theorem method (CTM)** A method based on the relationship established by the algorithmic coding theorem to estimate Kolmogorov-Chaitin complexity by way of algorithmic probability.

**Computability** The theory that studies the problems that can be solved by algorithmic means with, for example, digital computers.

**Halting problem** The strongest problem of predictability in algorithms related to whether a computer program will ever halt or not. Turing proved that it is an undecidable problem and is equivalent to the undecidability results by Gödel.

**Invariance theorem** A foundational theorem that establishes that shortest programs in different computer languages differ at most in length by a constant value and thus guarantees that the Kolmogorov-Chaitin complexity of an object converges up to a constant making this measure robust in the face of changes of models (such as a reference universal Turing machine).

**Kolmogorov-Chaitin complexity** Also known as algorithmic complexity of a string, it is the length of the shortest computer program that generates the string. The calculation of the Kolmogorov-Chaitin complexity is an upper semi-computable problem meaning upper bounds by finding short, if not the shortest, computer programs.

**Semi-computability** A semi-computable problem is one that allows approximations from above or below. If from above then is called upper semi-computable, and if from below then is called lower semi-computable.

**Semi-measure** A measure of probability whose sum does not add 1 because some events are undetermined, for example, by the Halting problem in the context of computability theory.

**Shannon entropy** A measure of combinatorial and statistical complexity based on the diversity of symbols used to define an object according to a probability distribution.

**Turing machine** An abstraction of a general-purpose computer introduced by Alan M. Turing.

**Undecidable problem** A problem that cannot be decided by traditional algorithms such as, for example, all those implemented in a Universal Turing machine.

**Universal distribution** The distribution produced by the output of random computer programs whose instruction bits are the result of, e.g., tossing a coin. It is called universal not only because it is the limit distribution of the output of all halting programs running on a Universal Turing machine but also because it is independent of the choice of particular Universal Turing machine or computer language up to a constant.

**Universal Turing machine** A Turing machine that can emulate any other Turing machine by reprogramming the machine using proper inputs.

insurmountable obstacle, and more often than not it merely served as a point of reference.

In recent years, we have been able to create and use more reliable estimates of algorithmic complexity using the coding theorem method (Gauvrit et al. 2014b, 2016). This has made it possible to deploy a precise and quantitative approximation of algorithmic complexity, with applications in many areas of psychology and the behavioral sciences – sometimes rather unexpected applications. Algorithmic complexity has proven useful in the study of human visual perception (Gauvrit et al. 2014a), as predicted some time back by Chater (1999). It has also cropped up in the study of cognitive development over the lifespan (Gauvrit et al. 2017b). More surprisingly, it has helped understand the conspiracy theory mindset (Dieguez et al. 2015), the emergence of grammar in natural languages (Kempe et al. 2015), and aesthetic judgments (Gauvrit et al. 2017a).

New methods, based on the coding theorem, are capable of accounting for known biases in human behavior, thus vindicating a computational algorithmic view of cognition first suggested by Turing, albeit now rooted in the concept of algorithmic probability, which in turn is based on computational universality while being independent of computational model, and which has the virtue of being predictive and testable as a model theory of cognitive behavior (Fig. 3).

While it is unclear whether the human mind has Turing super- or sub-universal capabilities (Zenil and Hernandez-Quiroz 2007), the algorithmic approach to cognition provides a strong formal connection between cognition and computation by means of recursion, which seems to be innate or else is developed over the course of the lifespan (Gauvrit et al. 2017b).

For example, behavioral sequences encoding right and left movements encoded in binary (R and L) (see <http://grahamshawcross.com/2014/07/26/counting-ants/>, Accessed on Dec 23, 2014.) (Reznikova and Ryabko 2011, 2012; Ryabko and Reznikova 2009), related to the manner and the speed at which scout ants communicate when returning to their foraging team in the colony, confirm the original author's suspicions that algorithmic information theory would account for the biases found (Zenil 2013; Zenil et al. 2015a).

## Algorithmic Cognition

The idea that complexity or, its reverse, simplicity are essential concepts for cognitive psychology was already understood in the middle of the twentieth century (Mach 1914), and these concepts have remained salient ever since (Oizumi et al. 2014). As early as the 1990s, the algorithmic theory of information was referenced by some researchers in psychology, who recommended the use of algorithmic complexity as a universal normative measure of complexity. Nevertheless, the noncomputability of algorithmic complexity was deemed an

An early example of an application to cognition of the algorithmic approach: the quantification and validation of Reznikova's famous, if controversial, ant communication experiments (Zenil 2013; Zenil et al. 2015a). It was found that ants communicate simpler strings (related to instructions for getting to food in a maze) in a shorter time, thus establishing an experimental connection between animal behavior, algorithmic complexity, and time complexity.

As shown in (Zenil et al. 2012), if the environment is too predictable, the cost of information-processing is very low. In contrast, if the environment is random, the cost is at a maximum. The results obtained by these information-processing methods suggest that organisms with superior learning capabilities save more energy and therefore have an evolutionary advantage.

In many ways, animal behavior (including, notably, human behavior) suggests that the brain acts as an (algorithmic) data compression device looking for truly algorithmic patterns (as opposed to simply statistical ones) (Fig. 1). For example, despite our very limited memory, it is clear we can retain long strings if they have low algorithmic complexity (e.g., the sequence 123456... see Fig. 4).

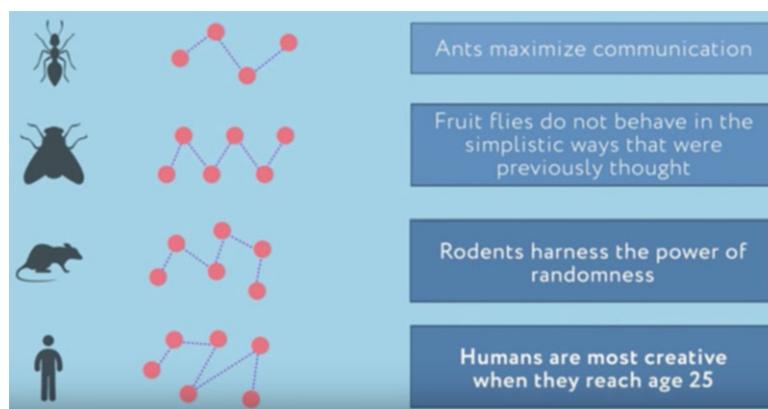
The cognitive and behavioral sciences deal, for the most part, with small series barely exceeding a few tens of values. For such short sequences,

estimating algorithmic complexity is a challenge. Indeed, until now, behavioral science has largely relied on subjective and intuitive accounts of complexity (Fig. 2).

As explained in (Gauvrit et al. 2015), humans display biases in the same *algorithmic* direction, from their motion trajectories (Peng et al. 2014) to their perception of reality (Chater 1999). Indeed, we have shown that cognition, including visual perception and the generation of subjective randomness, shows a bias that can be accounted for with the seminal concept of algorithmic probability (Chekaf et al. 2014; Gauvrit et al. 2014a, b, 2016; Kempe et al. 2015). Using a computer to look at humans in a novel fashion, specifically by using a reverse Turing test where what is assessed is the human mind and an “average” Turing machine or computer program implementing any possible compression algorithm, it was shown that the human mind behaves like a sophisticated universal model-generating algorithm.

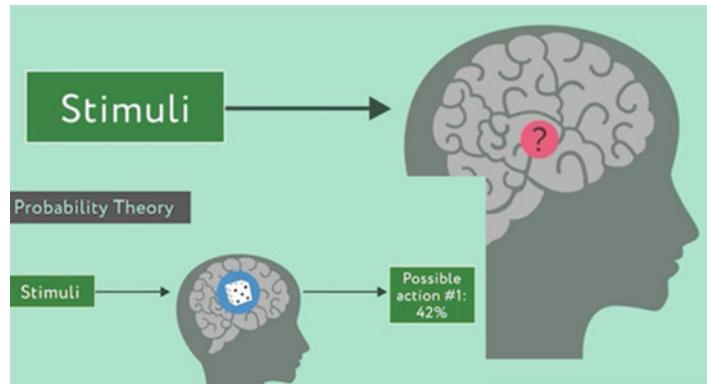
### Unveiling the Mechanistic Inner-Workings of the Mind

Algorithmic complexity can now be approached more reliably and precisely thanks to recent advancements in methodology based on massive calculations implementing a method known as the Coding theorem and the Block Decomposition



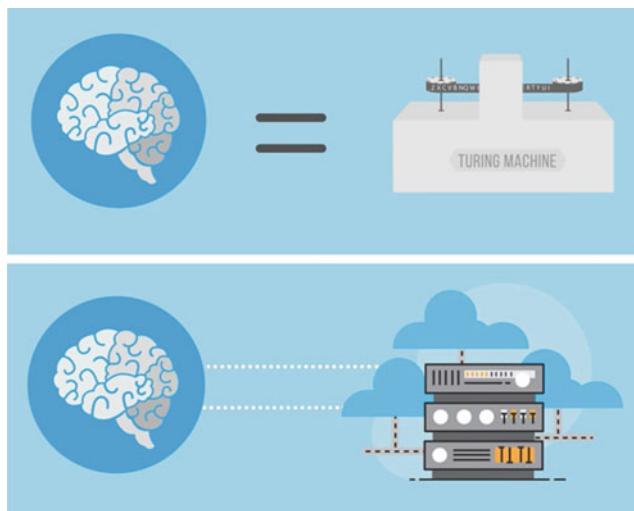
**Algorithmic Cognition and the Computational Nature of the Mind, Fig. 1** It has been found in field experiments that animals display a strong behavioral bias. For example, in Reznikova's communication experiments with ants, simpler instructions were communicated faster and more effectively than complex ones by scout ants

searching for patches of food randomly located in a maze. These findings have been now quantified with new algorithmic tools that would also be able to quantify other landmark experiments in the cognitive and behavioral sciences (Zenil 2013; Zenil et al. 2015a)



**Algorithmic Cognition and the Computational Nature of the Mind, Fig. 2** Determining the way the mind works is a challenge in the cognitive sciences. For a long time it was thought that the mind made certain decisions based on random bias (e.g., equiprobability), and in

many cases based upon previous experiences. But it was suspected to have a strong algorithmic nature, the proof of which, however, remained elusive due to a lack of powerful enough concepts, tools, and methods – until now



**Algorithmic Cognition and the Computational Nature of the Mind, Fig. 3** *Top:* Previous accounts of the human mind that liken them to digital computers (Turing machines) were found to be overly simplistic. *Bottom:* The new account sees the mind as a universal model-generator that generates algorithmic

hypotheses about the world and tests these hypotheses on a continuous basis, feeding back into the model and tweaking it in real-time like an algorithmic Bayesian estimator. The mind, however, has a strong bias that conforms with the theory of Algorithmic Probability

Method (Soler-Toscano et al. 2014; Zenil et al. 2015b, 2016). Today, this algorithmic approach to cognition is used, for example, to study visual perception, inductive reasoning, perception of chance, irrational beliefs, the emergence of linguistic structures, and neuropsychology, as we will see.

For example, in attempting to uncover the sources of our perception of randomness, Hsu and colleagues (Hsu et al. 2010) have suggested that it could be learned from the world. More precisely, according to the authors our two-dimensional randomness approach (Zenil et al.

2015b) could be based on the probability that a given pattern appears in visual natural scenes. They presented a series of patterns, which were  $4 \times 4$  grids with eight black cells, and eight white cells to subjects who had to decide whether these grids were “random” or not. The proportion of subjects answering “random” was used as a measure of subjective randomness. The authors also scanned a series of still nature shots. The pictures were binarized to black and white using the median as threshold, and every  $4 \times 4$  grid was extracted from them. From this dataset, a distribution was computed. The authors found that the probability that a grid appears in random photographs of natural scenes was correlated to the human estimate: the more frequent the grid, the less random it is rated. The authors interpret these results as evidence that we learn to perceive randomness through our eyes. An extension of the Hsu et al. study confirmed the correlation, and found that both subjective randomness and frequency in natural scenes were correlated to the algorithmic complexity of the patterns (Gauvrit et al. 2014a). It was found that natural scene statistics explain, in part, how we perceive randomness/complexity.

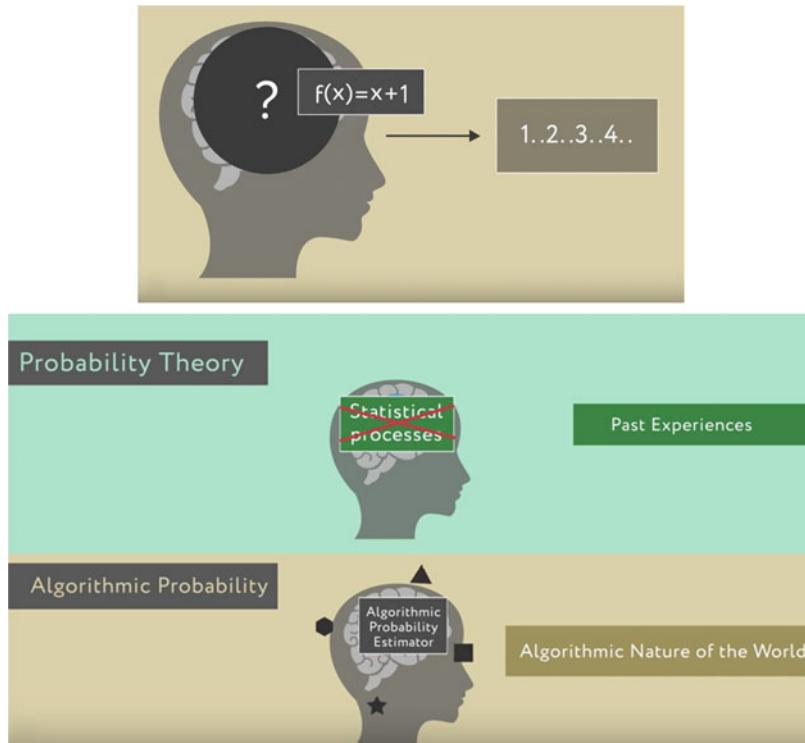
In a more recent study, researchers used two-dimensional patterns of points (Kempe et al. 2015). Twelve tokens were randomly placed on  $10 \times 10$  grids. The first participant had to memorize the pattern in 10 s, and then reproduce it on a blank grid, using new tokens. The second participant then had to reproduce the first participant’s response, and so on. Participants in a chain were sometimes all adults, and sometimes all children. Learnability can be defined as the number of correctly placed tokens. As expected, learnability continuously increases within each transmission chain, at the same rate for children and adults. As a consequence of increasing learnability, each chain converges toward a highly learnable pattern (which also is simple), which varies from one chain to the other.

One method for assessing Kolmogorov-Chaitin complexity, namely lossless compression, as epitomized by the Lempel-Ziv algorithm, has been long and widely used. Such a tool, together with classical Shannon entropy (Ze et al. 2014), has been used recently in neuropsychology to

investigate the complexity of EEG or fMRI data (Casali et al. 2013; Maguire et al. 2014). Indeed, the size of a compressed file gives an indication of its algorithmic complexity. The size of a compressed file is, in fact, an upper bound of true algorithmic complexity. On the one hand, compression methods have a basic flaw: they can only recognize statistical regularities and are therefore implementations of variations of entropic measures, only assessing the rate of entropic convergence based on repetitions of strings of fixed sliding-window size. If lossless compression algorithms work to approximate algorithmic (Kolmogorov-Chaitin) complexity, it is only because compression is a sufficient test for non-randomness, but they clearly fail in the other direction, when it is a matter of ascertaining whether something is the result of or is produced by an algorithmic process (e.g., the digits of the mathematical constant  $\pi$ ). That is, they cannot find structure that takes other forms than simple repetition. On the other hand, compression methods are inappropriate for short strings (of, say, less than a few hundreds of symbols). For short strings, lossless compression algorithms often yield files that are longer than the strings themselves, hence providing results that are very unstable and difficult, if not impossible, to interpret.

An observation led to the notion of *chunking*. It is now believed that humans can divide to-be-recalled lists into simple sub-lists. For instance, when they have to store “1, 2, 3, 4, 5, 3, 3, 3, 1, 1, 1” they can build three “chunks”: “1, 2, 3, 4, 5”; “3, 3, 3”; and “1, 1, 1.” As this example illustrates, chunks are not arbitrary: they are conceived to minimize complexity, and thus act as compression algorithms (Mathy and Feldman 2012). An objective factor showing that chunking does occur is that while recalling the above series, people make longer pauses between the hypothetical chunks than within them. Taking these new concepts into account, it is now believed that the short-term memory span is not roughly seven items, as established using unstructured lists, but more accurately around four chunks (Cowan 2010).

The short-term memory span is thus dependent on the type of list to be memorized (see Fig. 4). With structured series, one can recall more items



**Algorithmic Cognition and the Computational Nature of the Mind, Fig. 4** The new approach suggests and quantifies the way in which the mind can behave more algorithmically than statistically. An obvious example from memorization that displays the algorithmic power of the mind involves the task of learning a rather long sequence that has, however, a short description (or formula) that can generate the longer sequence. In this case (*top*), a sequence of consecutive digits in base 10 can easily be remembered because each digit is the sum of the previous digits plus 1, something that statistical approaches (*middle*) were unable to characterize, but that the new methods (*bottom*) could. This algorithmic nature of the mind is a reflection of the algorithmic nature of the world (Zenil et al. 2012) that the mind harnesses to navigate the world, taking advantage of both its statistical and

algorithmic regularities. A computer program is like a mathematical formula. Our suggestions do not imply, however, that the mind is a number cruncher trying to find the mathematical formula behind the data but that the mind does not only attempt to make statistical sense but also algorithmic sense of the world. That is, we attempt to learn patterns that display either statistical patterns or are algorithmically compressible but this does not imply that the mind always finds neither the shortest, a shorter, or even an efficient algorithm. In fact, the problem is uncomputable even for Turing machines, and it is an open question whether it is for the human mind (Zenil and Hernandez-Quiroz 2007), but the mind is equipped to do so, unlike previous assumptions in practice that would overlook such capabilities beyond simple statistical pattern recognition

but fewer chunks. This apparent contradiction can be overcome by challenging the clear-cut opposition between short-term memory span and working memory span. To overcome the limitations of short-term memory, humans can take advantage of the structure present in some series. Doing so requires using the “processing” component of working memory to analyze the data (Chekaf et al. 2014; Mathy and Feldman 2012). Thus, even simple tasks, where subjects must only store

and recall series of items, do tax the processing part of working memory. Hence according to this hypothesis, working memory works as a compression algorithm, where part of the resource is allocated to data storage while another part is dedicated to the “compression/decompression” program.

Our results in (Gauvrit et al. 2014a) suggest that the mind is intrinsically wired to believe that the world is algorithmic in nature, that what happens in it is likely the output of a random computer program

and not of a process producing uniform classical randomness. To ascertain whether this is the result of biological design or an acquired “skill,” one can look at whether and how people develop this view over the course of a lifetime. Preliminary results suggest that this algorithmic view or filter about the world we are equipped with is constructed over time, reaching a peak of algorithmic randomness production and perception at about age 25 years (see <http://complexitycalculator.com/hmg/>). This means that the mind adapts and learns from experience, and for some reason it develops the algorithmic perception before changing again. We would not have developed such a worldview peaking at reproductive age had it no evolutionary advantage, as it seems unlikely to be a neutral feature of the mind in relation to the world, affecting the way it perceives events, and therefore how it learns and behaves. And this is not exclusive to the human mind but is also characteristic of animal behavior, as shown in (Zenil 2013; Zenil et al. 2015a). All this evidence points in the same direction: that the world is or appears to us to be highly algorithmic in nature, at least transitionally.

### **Algorithmic Bayesian Estimations as a Model of the Mind**

Since the emergence of the Bayesian paradigm in cognitive science, researchers have expressed the need for a formal account of complexity based on a sound complexity measure. They have also struggled to find a way of giving a formal normative account of the probability that a deterministic algorithm produces a given sequence, such as the heads-or-tails string “HTHTHTHT,” which intuitively looks like the result of a deterministic process even if it has the same probability of occurrence as any other sequence of the same length according to classical probability theory, which assigns a probability of  $1/2^8$  to all sequences of size 8 (Fig. 4).

The famous work of Kahneman, Slovic, and Tversky (1982) aimed at understanding how people reason and make decisions in the face of uncertain and noisy information sources. They showed that humans were prone to many errors

about randomness and probability. For instance, people tend to claim that the sequence of heads or tails “HTHTHTHHHT” is more likely to appear when a coin is tossed than the series “HHHHHTTTTT” (see Fig. 3).

In the “heuristics and bias” approach advocated by Kahneman and Tversky, these “systematic” errors were interpreted as biases inhering in human psychology, or else as the result of using faulty heuristics. For instance, it was believed that people tend to say that “HHHHHTTTTT” is less random than “HTHTHTHHHTT” because they are influenced by a so-called representativeness heuristic, according to which a sequence is more random the better it conforms to prototypical examples of random sequences. Human reasoning, it has been argued, works like a faulty computer. Although many papers have been published about these biases, not much is known about their causes.

Another example of a widespread error is the so-called equiprobability bias (Lecoutre 1992), a tendency to believe that any random variable should be uniform (with equal probability for every possible outcome). In the same vein as the seminal investigations by Kahneman et al. (Tversky and Kahneman 1975), this bias too, viz. the erroneous assumption that randomness implies uniformity, has long been interpreted as a fundamental flaw of the human mind.

During the last decades, a paradigm shift has occurred in cognitive science. The “new paradigm” – or Bayesian approach – suggests that the human (or animal) mind is not a faulty machine, but a probabilistic machine of a certain type. According to this understanding of human cognition, we all estimate and constantly revise probabilities of events in the world, taking into account any new pieces of information, and more or less following probabilistic (including Bayesian) rules.

Studies along these lines often try to explain our probabilistic errors in terms of a sound intuition about randomness or probability applied in an inappropriate context. For instance, a mathematical and psychological reanalysis of the equiprobability bias was recently published (Gauvrit 2014). The mathematical theory of randomness, based on algorithmic complexity (or on entropy, as it happens), does in fact imply

uniformity. Thus, claiming that the intuition that randomness implies uniformity is a bias that does not fit with the mathematical theory. On the other hand, if one follows the mathematical theory of randomness, one must admit that a combination of random events is, in general, not random anymore. Thus, the equiprobability bias (which indeed is a bias, since it yields frequent faulty answers in the probability class) is not, we argue, the result of a misconception regarding randomness, but a consequence of the incorrect intuition that random events can be combined without affecting their property of randomness.

We now believe that when we have to compare the probability that a fair coin produces either “HHHHHTTTTT” or any other 10-item long series, we do not really do so. One reason is that the question is unnatural: our brain is built to estimate the probabilities of the causes of observed events, not the *a priori* probability of such events. Therefore, say researchers, when we have participants rate the probability of the string  $s = \text{“HHHHHTTTTT”}$ , for instance, or any other, they do not actually estimate the probability that such a string will appear on tossing a fair coin, which we could write as  $P(s | R)$  where  $R$  stands for “random process,” but the reverse probability  $P(R | s)$ , that is, the probability that the coin is fair (or that the string is genuinely random), given that it produced  $s$ . Such a probability can be estimated using Bayes’ theorem:

$$P(R | s) = \frac{P(s | R)P(R)}{P(s | R)P(R) + P(s | D)P(D)},$$

where  $D$  stands for “not random” (or deterministic).

In this equation, the only problematic element is  $P(s | D)$ , the probability that an undetermined but deterministic algorithmic will produce  $s$ . It was long believed that no normative measure of this value could be assumed, although some authors had the intuition that it was linked to the complexity of  $s$ : simple strings are more likely to appear as a result of an algorithm than complex ones.

The algorithmic theory of information actually provides a formal framework for this intuition. The

algorithmic probability of a string  $s$  is the probability that a randomly chosen program running on a Universal (prefix-free) Turing Machine will produce  $s$  and then halt. It therefore serves as a natural formal definition of  $P(s | D)$ . As we will see below, the algorithmic probability of a string  $s$  is inversely linked to its (Kolmogorov-Chaitin) algorithmic complexity, defined as the length of the shortest program that produces  $s$  and then halts. Simpler strings have a higher algorithmic probability.

One important drawback of algorithmic complexity is that it is not computable. However, there now exist methods to approximate the probability, and thus the complexity, of any string, even short ones (see below), giving rise to a renewed interest in complexity in the cognitive sciences.

Using these methods (Gauvrit et al. 2016), we can compute that, with a prior of 0.5, the probability that the string HHHHHTTTTT is random amounts to 0.58, whereas the probability that HTTHTHHHTT is random amounts to 0.83, thus confirming the common intuition that the latter is “more random” than the former.

## Conclusion

In the cognitive sciences, the study of working memory, of probabilistic reasoning, the emergence of structure in language, strategic response, and navigational behavior is cutting-edge research. In all these areas the algorithmic approach to cognition has made contributions (Chekaf et al. 2014; Gauvrit et al. 2014a, b, 2016; Kempe et al. 2015; Zenil 2013; Zenil et al. 2015a) based upon algorithmic probability as a useful normative tool, shedding light on the algorithmic mechanisms of cognitive processes. The concepts and methods explained here are available through a publicly accessible online calculator at <http://complexitycalculator.com/>.

## Bibliography

- Casali AG, Gosseries O, Rosanova M, Boly M, Sarasso S, Casali KR, Casarotto S, Bruno M-A, Laureys S, Tononi G et al (2013) A theoretically based index of

- consciousness independent of sensory processing and behavior. *Sci Transl Med* 5(198):105
- Chater N (1999) The search for simplicity: a fundamental cognitive principle? *Quart J Exp Psychol A* 52(2): 273–302
- Cheka M, Gauvrit N, Mathy F (2014) Chunking on the fly in working memory and its relationship to intelligence. In: 55th Annual meeting of the psychonomic society
- Cowan N (2010) The magical mystery four: How is working memory capacity limited, and why? *Curr Dir Psychol Sci* 19(1):51–57
- Dieguez S, Wagner-Egger P, Gauvrit N (2015) Nothing happens by accident, or does it? a low prior for randomness does not explain belief in conspiracy theories. *Psychol Sci* 26(11):1762–1770
- Gauvrit N, Kinga M (2014) The equiprobability bias from a mathematical and psychological perspective. *Adv Cogn Psychol* 10(4):119–130
- Gauvrit N, Soler-Toscano F, Zenil H (2014a) Natural scene statistics mediate the perception of image complexity. *Vis Cogn* 22(8):1084–1091
- Gauvrit N, Zenil H, Delahaye J-P, Soler-Toscano F (2014b) Algorithmic complexity for short binary strings applied to psychology: a primer. *Behav Res Methods* 46(3): 732–744
- Gauvrit N, Zenil H, Tegnér J (2015) The information-theoretic and algorithmic approach to human, animal and artificial cognition. *arXiv preprint arXiv:1501.04242*
- Gauvrit N, Singmann H, Soler-Toscano F, Zenil H (2016) Algorithmic complexity for psychology: a user-friendly implementation of the coding theorem method. *Behav Res Methods* 48(1):314–329
- Gauvrit N, Soler-Toscano F, Guida A (2017a) A preference for some types of complexity comment on perceived beauty of random texture patterns: a preference for complexity. *Acta Psychol* 174:48–53
- Gauvrit N, Zenil H, Soler-Toscano F, Delahaye J-P, Brugger P (2017b) Human behavioral complexity peaks at age 25. *PLoS Comput Biol* 13(4):e1005408
- Hsu AS, Griffiths TL, Schreiber E (2010) Subjective randomness and natural scene statistics. *Psychon Bull Rev* 17(5):624–629
- Kahneman D, Slovic P, Tversky A (1982) Judgment under uncertainty: heuristics and biases. Cambridge University Press, New York and Cambridge.
- Kempe V, Gauvrit N, Forsyth D (2015) Structure emerges faster during cultural transmission in children than in adults. *Cognition* 136:247–254
- Lecoutre M-P (1992) Cognitive models and problem spaces in “purely random” situations. *Educ Stud Math* 23(6):557–568
- Mach E (1914) The analysis of sensations, and the relation of the physical to the psychical. Open Court Publishing Company, Chicago
- Maguire P, Moser P, Maguire R, Griffith V (2014) Is consciousness computable? quantifying integrated information using algorithmic information theory. *arXiv preprint arXiv:1405.0126*
- Mathy F, Feldman J (2012) What's magic about magic numbers? chunking and data compression in short-term memory. *Cognition* 122(3):346–362
- Masafumi Oizumi, Larissa Albantakis, and Giulio Tononi. From the phenomenology to the mechanisms of consciousness: integrated information theory 3.0. *PLoS computational biology*, 10(5):e1003588, 2014.
- Peng Z, Genewein T, Braun DA (2014) Assessing randomness and complexity in human motion trajectories through analysis of symbolic sequences. *Front Hum Neurosci* 8:168
- Reznikova Z, Ryabko B (2011) Numerical competence in animals, with an insight from ants. *Behaviour*:405–434
- Reznikova Z, Ryabko B (2012) Ants and bits. *IEEE Inform Theor Soc News* 62(5):17–20
- Ryabko B, Reznikova Z (2009) The use of ideas of information theory for studying “language” and intelligence in ants. *Entropy* 11(4):836–853
- Soler-Toscano F, Zenil H, Delahaye J-P, Gauvrit N (2014) Calculating Kolmogorov complexity from the output frequency distributions of small Turing machines. *PLoS One* 9(5):e96223
- Tversky A, Kahneman D (1975) Judgment under uncertainty: heuristics and biases. In: Utility, probability, and human decision making. Springer, New York, pp 141–162
- Ze W, Li Y, Childress AR, Detre JA (2014) Brain entropy mapping using fmri. *PLoS One* 9(3):e89948
- Zenil H (2013) Algorithmic complexity of animal behaviour: from communication to cognition. In: Theory and practice of natural computing second international conference proceedings, Cáceres, Spain TPNC 2013
- Zenil H, Hernandez-Quiroz F (2007) On the possible computational power of the human mind. In: C. Gershenson, D. Aerts, and B. Edmonds (eds) Worldviews, science and us: philosophy and complexity. World Scientific, Singapore, pp 315–334
- Zenil H, Gershenson C, Marshall JAR, Rosenblueth DA (2012) Life as thermodynamic evidence of algorithmic structure in natural environments. *Entropy* 14(11): 2173–2191
- Zenil H Marshall JAR, Tegnér J (2015a) Approximations of algorithmic and structural complexity validate cognitive-behavioural experimental results. *arXiv preprint arXiv:1509.06338*
- Zenil H, Soler-Toscano F, Delahaye J-P, Gauvrit N (2015b) Two-dimensional kolmogorov complexity and an empirical validation of the coding theorem method by compressibility. *Peer J Comput Sci* 1:e23
- Zenil H, Soler-Toscano F, Kiani NA, Hernández-Orozco S, Rueda-Toicen A (2016) A decomposition method for global evaluation of Shannon entropy and local estimations of algorithmic complexity. *arXiv preprint arXiv:1609.00110*



---

## Approximations to Algorithmic Probability

Hector Zenil

Algorithmic Dynamics Lab, Unit of Computational Medicine and SciLifeLab, Center for Molecular Medicine, Department of Medicine Solna, Karolinska Institutet, Stockholm, Sweden

### Article Outline

#### Glossary

A Computational Theory of Everything

Algorithmic Probability

Conclusion

Bibliography

### Glossary

#### Algorithmic coding theorem (not to confuse with Shannon's coding theorem)

A theorem that formally establishes an inversely proportional relationship between Kolmogorov-Chaitin complexity and algorithmic probability.

**Algorithmic cognition** The study of animal, human, and artificial cognition based on the theory of algorithmic probability.

**Algorithmic information theory** The literature based on the concept of Kolmogorov-Chaitin complexity and related concepts such as algorithmic probability, compression, optimal inference, the Universal Distribution, and Levin's semi-measure.

**Algorithmic probability** The probability to produce an object from a random digital computer program whose program binary digits are chosen by chance. The calculation of algorithmic probability is a lower semi-computable problem.

**Algorithmic randomness** How removed the length of the shortest generating program is

from the size of the uncompressed data that such program generates.

**Bayesian model** A kind of probabilistic approach to inference based on updating assumed distributions considered as prior beliefs.

**Behavioral sequence** A sequence of values representing a behavioral trait of an animal, human, or artificial machine.

**Block decomposition method (BDM)** A divide and conquer method that chunks a string or multidimensional object into smaller pieces for which better lower bounds of algorithmic probability can be produced.

**Coding theorem method (CTM)** A method based on the relationship established by the algorithmic coding theorem to estimate Kolmogorov-Chaitin complexity by way of algorithmic probability.

**Computability** The theory that studies the problems that can be solved by algorithmic means with, for example, digital computers.

**EEG or electroencephalogram** A recording of brain activity. During the test, small sensors are attached to the scalp to pick up the electrical signals produced when brain cells send messages to each other.

**fMRI** Functional magnetic resonance imaging or functional MRI (fMRI) measures brain activity by detecting changes associated with blood flow.

**Invariance theorem** A foundational theorem that establishes that shortest programs in different computer languages differ at most in length by a constant value and thus guarantees that the Kolmogorov-Chaitin complexity of an object converges up to a constant making this measure robust in the face of changes of models (such as a reference universal Turing machine).

**Kolmogorov-Chaitin complexity** Also known as algorithmic complexity of a string, it is the length of the shortest computer program that generates the string. The calculation of the Kolmogorov-Chaitin complexity is an upper semi-computable problem meaning upper

bounds by finding short, if not the shortest, computer programs.

**Mechanistic model** A model that corresponds to the actual inner workings of the mechanism that the model is representing and is thus able to simulate closer step by step.

**Shannon entropy** A measure of combinatorial and statistical complexity based on the diversity of symbols used to define an object according to a probability distribution.

**Subjective randomness** The kind of randomness that can be perceived, even if limited, by animals, humans, or machines with restricted resources.

**Turing machine** An abstraction of a general-purpose computer introduced by Alan M. Turing.

**Universal Turing machine** A Turing machine that can emulate any other Turing machine by reprogramming the machine using proper inputs.

causality, where every observation was assumed to have a cause that was something other than a metaphysical explanation.

Over the last few centuries, science has made a lot of progress explaining more with less effectively suggesting the algorithmic nature of the universe. For a long time, the Ptolemaic geocentric model was thought to explain the strange motion of planets in the sky. When it came to planetary movement, the geocentric theory had the same predictive power as the later heliocentric model. The geocentric model, however, was unable to explain how the outer planets seemed to have a period related to the Sun's period if they had no special relation amongst themselves. Not only was the heliocentric model a better model because it required, overall, fewer constraints or features, such as the many epicycles needed in the geocentric model, but it did not need to be supplemented by additional explanations. Some anomalies remained to be accounted for (having to do with the elliptical orbit, as discovered by Kepler) but it did not need to explain odd regularities that were apparent yet unexplained by the geocentric model. We have replaced the heliocentric by much more sophisticated models, starting with Kepler's and Newton's, that not only endorsed and embraced the heliocentric model but placed at the center of the solar system a center of gravity common to the Sun and the rest of the planets, a center of gravity that turns out to be inside the Sun but not to be exactly the Sun or its center. In other words, under updated models the truth value of the claim that either the planets revolve around the Sun or that the Sun revolves around the planets is relative. With this update on the heliocentric model, Newton was able to link the movement of the planets to the movement of everyday objects on Earth, and to supply a reason as to why objects fall down and why we appear to be stuck to the Earth's surface, an amazing feat where a minor increase in the sophistication of a model yielded explanations of a tremendous number of hitherto unexplained phenomena. But it was not until Einstein's general relativity that we could explain phenomena that none of the previous classical models could, phenomena that were considered strange anomalies, as if they were the

## A Computational Theory of Everything

Physicists have long been searching for a so-called Theory of Everything (ToE). Just as quantum mechanics explains the smallest phenomena, from microwaves to light, and general relativity explains the classical world, from gravity to space and time, a ToE would explain everything in the universe, from the smallest to the largest phenomena, in a single formulation.

Science has operated under the assumption and in light of strong evidence that the world is highly, if not completely, algorithmic in nature. If the world were not structured, our attempts to construct a body of theories from observations, to build what we consider ordered scientific models of the world, would have failed. That they have not is spectacular vindication of the validity of the world's orderly character. We started out believing that the world was ruled by magic, and by irrational and emotional gods. However, thinkers from ancient civilizations such as China and India and, notably, Greece made the first recorded attempts to introduce reason and logic into scholarly discourse, advancing rules that may be seen as the first steps towards a computational theory of

result of the caprices of ancient gods, such as the so-called Venus' perihelion movement, the phenomenon that sees Venus' elongated orbit change significantly over the course of a year. The only indication that Venus' movement was not subject to the mood of an angry god was perhaps its regularity. So both regularity in phenomena such as the movement of stars and planets, and the achievement of superior models able to explain a much greater number of unexplained phenomena with little to no increase in sophistication compared to precursors, are at the center of science.

That we are able to explain more without need of many more assumptions, devising ever more encompassing scientific models, suggests that our complex world tends toward algorithmic simplicity. Indeed, the more simple physics finds the world to be, the more algorithmic it suggests it is, as the universe seems to be generated or explained by simple principles. The chances of uncovering a short computer program producing the universe are thus greater if physical phenomena can be explained by increasingly simple and shorter descriptions and models. This does not come as a surprise. If current cosmological theories are correct, the universe has a common and single origin and is thus deterministically connected from beginning to end.

It is true that laws are only models of natural phenomena and that some later models may become slightly more complex before becoming more simple, but ultimately all models in science have become simpler and simpler with time and further research, so that today we can explain a greater number of phenomena encompassing almost every aspect of the natural world using only two major models, general relativity and the standard model. So both the direction of science and the assumption that the world is algorithmic in nature are solid and definite. We can scientifically explain much more today than we could 50 years ago, and 50 years ago we could explain much more than we could 200 years ago, and so on, using simpler and shorter scientific models of incredible accuracy.

If the world is causal and deterministic, as science empirically shows, general relativity suggests and quantum mechanics does not rule out, such a

theory would have a computational form (Wolfram 2002), and even riding on quantum fluctuations it would appear algorithmic at the macroscopic scale. A computational theory of everything of this type in (Schmidhuber 2002), and based on the works of Solomonoff (1964) and Levin (1974), has also been suggested to be miraculous (Kirchherr et al. 1997), and more recently, the merits of approximating it were underscored by Marvin Minsky. According to him, the underlying theory would only be next in importance (while actually being deeply related) to the results of Kurt Gödel, who found that proof and truth are often disconnected, given that one cannot prove certain truths and that being able to prove every possible truth leads to contradictions. At the *Panel on The Limits of Understanding* Minsky pointed out (Source: <https://youtu.be/DfY-DRsE86s?t=1h30m2s>):

It seems to me that the most important discovery since Gödel was the discovery by Chaitin, Solomonoff and Kolmogorov of the concept called, Algorithmic Probability, which is a fundamental new theory of how to make predictions given a collection of experiences, and this is a beautiful theory. Everybody should learn it, but it's got one problem, that is, that you cannot actually calculate what this theory predicts because it is too hard, it requires an infinite amount of work. However, it should be possible to make practical approximations to the Chaitin, Kolmogorov, Solomonoff theory that would make better predictions than anything we have today. Everybody should learn all about that and spend the rest of their lives working on it.

World Science Festival in New York City, Dec 14, 2014.

### The Power of Mechanistic Models

The first mechanistic model of the solar system did not originate with Claudius Ptolemy, Copernicus, or even Kepler, but with Newton and his theory of gravitation. A mechanistic model is one that posits the causes of certain observables rather than merely describing phenomena. None of the models previous to Newton accounted for what underlay the motion of planets, only offering reasons or hypotheses for their apparent movements. It is not until Newton that we knew the planets were somehow "falling," just as apples fall from a tree. The Sun and the planets in our solar system are attracted to each other and pull toward each other, making them move in the way they do.

So what is a causal and mechanistic model in more general terms? Newton's model is causal and mechanistic because it provides the means to, in principle, reproduce the conditions of the solar system and generate exactly the same outcome. This basically means that Newton's model can be emulated mechanistically to reproduce the movement of the planets and Sun. Indeed, the only difference between Kepler's and Newton's models, given that they are equally accurate in their prediction of planetary motion, is that Newton's is causal and can explain many more phenomena outside the realm of planetary motion. Not all models are of the same type. For example, in a mechanistic model, the sun's cycle may require two coupled gears of size  $20 \times 20$  the gear cycle of the moon; the mechanical model would then suggest such a ratio between size and distance of moon to sun. However, this would have been impossible to figure out by following the epicycles model. An example of a mechanistic model is a Rube Goldberg Machine (see Fig. 5), because such a machine implements itself, that is, every action of the machine corresponds to the actual performance of a task. In other words, the best model of something is itself. But we rarely have the chance to design or control a system to include an accurate representation of itself; instead we produce hypotheses for models that we test.

Let's take as an example a very abstract one, a rational number such as  $\pi$ . The constant  $\pi$  is a computable number and thus can be generated mechanistically. A mechanical model of  $\pi$  is the one suggested by the relationship between the diameter and the circumference of a circle. This, in practice, is quite difficult and is not undertaken physically, but from its abstract calculation, mathematicians have learned many properties beyond the original realm of  $\pi$  itself. For example,  $\pi$  is of low algorithmic (Kolmogorov-Chaitin) complexity because it has many short descriptions, as has been found from many relationships in mathematics.

## Algorithmic Probability

What connects all mechanistic model-driven approaches in science is none other than the computational theory of Algorithmic Probability

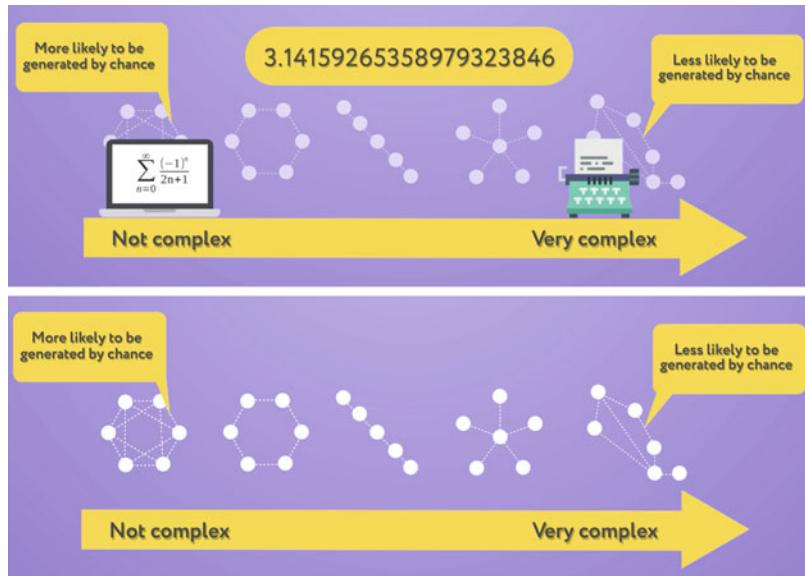
(Levin 1974; Solomonoff 1964), a theory that elegantly connects computation with classical probability in a proper way through a theorem called the *algorithmic Coding theorem* (c.f. below), which for its part establishes that the most frequent outcome of a causal/deterministic system will also be the most simple in the sense of algorithmic complexity, that is, the estimation of the length of the shortest computer program that produces such an output.

This measure defines the probability of data being produced by a random program running on a universal Turing machine (Levin 1974; Solomonoff 1964). The measure is key to understanding data in the best possible scientific way, that is, by exploring hypotheses in search of the most likely genesis of the data, so as to be able to make predictions. Formally,

$$m(x) = \sum_{p:U(p)=x} 1/2^{|p|} < 1 \quad (1)$$

i.e., it is the sum over all the programs that produce  $x$  running on a universal (prefix-free (The group of valid programs forms a prefix-free set (no element is a prefix of any other, a property necessary to keep  $0 < m(x) < 1$ ). For example, program 2 starts with program 1, so if program 1 is a valid program then program 2 cannot be valid. The set of valid programs is said to constitute a prefix-free set, that is, no element is a prefix of any other, a property necessary to keep  $0 < m(x) < 1$ .)) Turing machine  $U$ . The probability measure  $m(x)$  is traditionally called Levin's semi-measure, Solomonoff-Levin's semi-measure, or *algorithmic probability*, and it defines what is known as the *Universal Distribution* (*semi* from *semi-decidable* and the fact that  $m(x)$ , unlike probability measures, does not add up to 1).  $m(x)$  formalizes the concept of Occam's razor (favoring simple – or shorter – hypotheses over complicated ones) (Fig. 1).

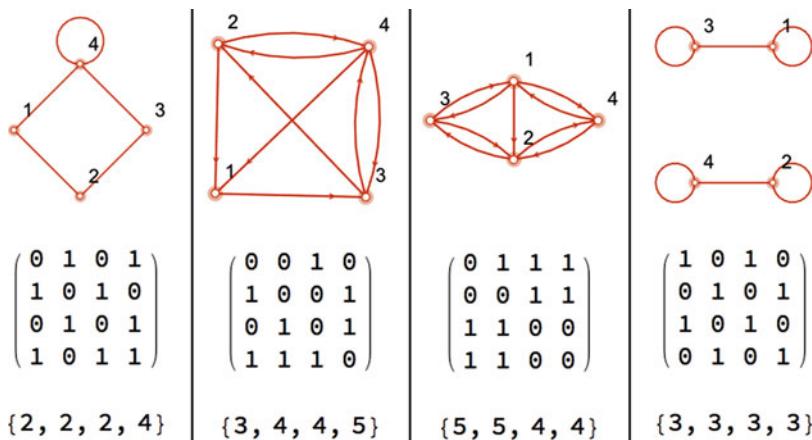
In Zenil et al. (2017), we showed that the use and application of computable measures, in particular Shannon Entropy (Shannon 1948), can lead to weak and contradictory results when



### Approximations to Algorithmic Probability,

**Fig. 1** The mathematical constant  $\pi$  can be more easily generated by typing randomly on a computer than on a typewriter, because there are short computer programs that

produce all the digits of  $\pi$  whereas on a typewriter one would need to type digit by digit. The digits of the constant  $\pi$  is thus algorithmically more likely to occur by chance than random objects that do not have short descriptions



### Approximations to Algorithmic Probability,

**Fig. 2** Objects can always be described in different ways and using different languages. Universal measures of complexity are those robust measures that are invariant to object description up to a constant (the length of the translation program between one description and another). Universal

measures of complexity can only be uncomputable [Zenil 2017] if they are to characterize every possible computable feature. Depicted here are labelled graphs that can be *losslessly* (i.e. without loss of information) described in their visual representation (*top*), as an adjacency matrix (*middle*) or as an ordered sequence of vertex degrees (*bottom*)

evaluating the information content or random typicality of objects, in particular graphs and networks, given that such objects, like any other object, can be described in different ways

without loss of information (see Fig. 2) and, unlike universal measures such as algorithmic complexity, computable measures are not robust to changes in such descriptions (Fig. 2).

## An Alternative Method to Lossless Compression

There is a trend in statistical machine learning fueled by big data and computational power that involves the packing of a set of regression analysis methods into what is called a deep learning and deep neural network. These approaches offer black boxes with incredible capabilities but cannot be farther away from mechanistic models. In some ways, they are sophisticated epicyclic models of some data. They can be extremely useful, can even be as precise as the corresponding mechanistic models of the phenomena they try to capture, but they will fall short at scaling and abstracting general properties into models one can build upon without having to add yet another layer of black boxes. The challenge is how to put together the enormous capabilities of deep learning to synthesize data into a combinatorial set of feature variables with the abstraction capabilities of truly model-based approaches.

Some approaches are unconventional but are not meant to remain so forever. The alternatives for measuring and applying algorithmic complexity other than lossless compression and translating highly abstract theory to highly applied areas have started to produce surprising results that would not be possible had the new numerical methods been unable to go beyond previous traditional schemes. The detailed theory introduced in (Delahaye and Zenil 2012; Soler-Toscano et al. 2014) as a novel alternative to compression (Delahaye and Zenil 2012) has found applications in the study of graphs and networks (Zenil et al. 2014), in molecular biology (Zenil et al. 2016), and animal and human cognition (Gauvrit et al. 2014a, b, 2016, 2017a, b).

The alternative to compression is based upon the algorithmic Coding theorem that describes the connection between the complexity of an object  $K(x)$  and its algorithmic probability of being produced by a random computer program  $m(x)$  (Gregory and Chaitin 1966; Levin 1974) (whose executable code digits are the result of tossing a coin). The theorem established that:

$$K(x) = -\log_2(m(x)) + c \quad (2)$$

where  $c$  is a constant independent of  $x$ .

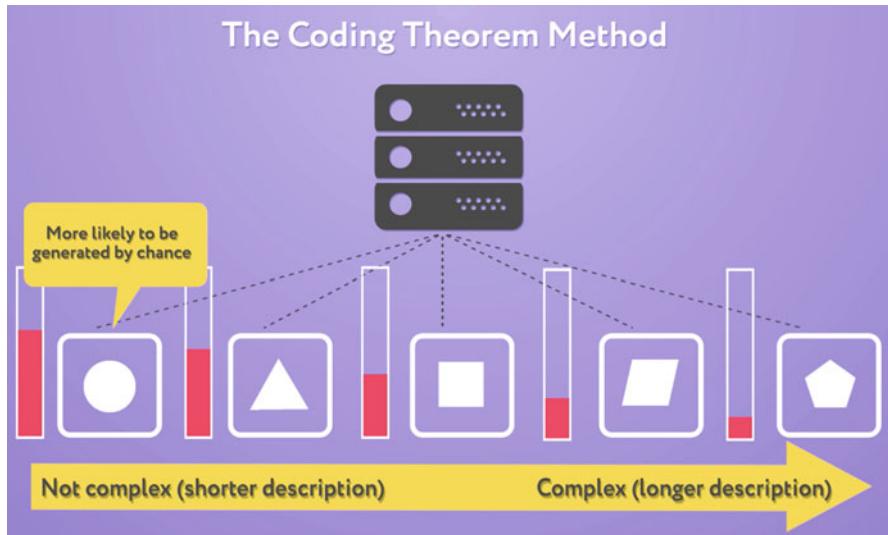
The intuition behind the Coding theorem (2) is a beautiful relation between program-size complexity and frequency of production. If you wished to produce the digits of a mathematical constant like  $\pi$  by throwing digits at random, you would have to produce every digit of its infinite irrational decimal expansion. If you seated a monkey at a typewriter (with, say, 50 keys), the probability of the monkey typing an initial segment of 2400 digits of  $\pi$  by chance is  $(1/50)^{2400}$ .

If, instead, the hypothetical monkey (simply a source of randomness) is seated at a computer, the chances of its producing a program generating the digits of  $\pi$  are on the order of  $1/50^{158}$ , because it would take the monkey only 158 characters to produce the first 2400 digits of  $\pi$  using, for example, C language.

The less random an object, the more likely it is to be produced by a short program. There is a greater probability of producing the program that produces the object, especially if the program producing  $x$  is short. Therefore, if an object is compressible, then the chances of producing the compressed program that produces said object are greater than they would be for a random object, i.e.,  $|p| \ll |x|$  such that  $U(x) = x$ .

## An Unconventional Approach to Computing the Uncomputable

It is one thing to propose that if the world is computational then it has to be generated by some computational rule (Schmidhuber 2002), and a different thing to explore the space of such possible computational rules (Wolfram 2002). However, very little to nothing had been done to actually approximate something like Levin's universal distribution, to plug it back into a possible algorithmic theory of everything and find applications in the sense of Minsky (c.f. quotation above). Moreover, through the algorithmic Coding theorem one would be able to approximate algorithmic (Kolmogorov-Chaitin) complexity rather than, for example, approximating it only by statistical means, by using common lossless compression algorithms that implement measures of entropy rate based on statistical regularities – repetitions – captured by a sliding window traversing a string.



**Approximations to Algorithmic Probability, Fig. 3** Looking at the empirical output distribution of trillions of small computer programs. Small pieces that are often produced by most small computer programs

have *high algorithmic content*, those rarely generated are more *algorithmically random* according to Algorithmic Probability and the algorithmic Coding theorem

Let  $(n, 2)$  be the set of all Turing machines with  $n$  states and 2 symbols, and let  $D(n)$  be the function that assigns to every finite binary string  $x$  the quotient:

$$D(n) = \frac{\text{#of times that a machine } (n, 2) \text{ produces } x}{\text{of machines in } (n, 2)} \quad (3)$$

The function  $D(n, m)$  is an algorithmic measure based on Levin's Universal Distribution  $m(s)$  approximated by finite means (Soler-Toscano et al. 2014), and it can be used to approximate  $K(s)$ , the algorithmic complexity of string  $s$  by using the algorithmic Coding Theorem as follows (Fig. 3):

$$\text{CTM}(s) \sim -\log_2 D(n, m)(s)$$

where CTM stands for *Coding Theorem Method*. The greater the value of  $n$  we use to calculate  $D(n, m)(s)$ , the better the approximation to  $K(s)$ .

$D(n)$  can also be seen as averaging  $K$  over a large set of possible computer languages, each represented by a small Turing machine, in order to reduce the possible impact of the constant from the *invariance theorem*.

The CTM allows us not only to build an empirical distribution of computer program outputs from smaller to larger sizes, but once a string is generated for the first time among all the computer programs of the smallest size, we know which Turing machine is the smallest one producing a string and also the runtime of such a Turing machine. We take the runtime of such a Turing machine as an estimation of another seminal measure of complexity, Bennett's Logical Depth (LD) approximated by CTM. LD is a measure of *sophistication* that assigns low complexity to both simple and random objects and is defined as the computation time that it takes to decompress an object from its (near) greatest compressed version.

Once the tools and methods are to hand, one can see to what extent the world really runs on a universal mechanical computer *algorithmicity* (Zenil 2011; Zenil and Delahaye 2010): how much the outcome of a process resembles the way in which outcomes would be sorted according to the universal distribution, and of *programmability* (Zenil 2013, 2014a, b, 2015); how much such a process can be reprogrammed at will. The more reprogrammable, the more causal, given that a random object cannot be reprogrammed in any practical way other than by changing every possible bit.

### A Hybrid Divide-and-Conquer Method

The Block Decomposition Method (BDM) allows us to extend the power of the Coding theorem by using a combination of Shannon entropy and algorithmic probability. The idea is to divide a piece of data into segments for which we have algorithmic probability approximations, and then count the number of times that a local regularity occurs (Fig. 5). The power of LD can also be extended by a BDM-like formula, a multiplicative variation of the BDM formula.

What would these CTM and BDM methods do with, for example, the binary digital expansion of a mathematical constant such as  $\pi$  that we know is of low complexity but is also random-looking? We know statistical approaches would fail to characterize it. But we need to make a distinction between *principle* and *practice*. The Coding Theorem method (CTM) can, in principle, and to some extent, in practice, find the algorithmic nature of  $\pi$ . CTM would eventually find that subsequences of  $\pi$  have lower complexity than that estimated by, e.g., Shannon Entropy by an uninformed observer (with no access to the knowledge of the deterministic nature of  $\pi$ ). Such an observer would assign maximal entropy to the digits of  $\pi$  because it looks random (there are no apparent repetitions at any level of granularity) where, in contrast, CTM (and thus BDM) would asymptotically find  $\pi$  to be of low algorithmic complexity because it can be generated by short mathematical formulas implemented by short computer programs.

### A Sequence of Causal Models Is Itself a Causal Model

Trillions of small computer programs were run to build an empirical probability distribution (Delahaye and Zenil 2012; Soler-Toscano et al. 2014) that approximates what in the theory of algorithmic probability is called the *universal distribution* (Levin 1974), used as the prior in all sorts of Bayesian calculations that require the framing of hypotheses about generating models or mechanisms that produce an object. Take a binary string of a million 1 s. The theory tells us that the most likely mechanistic generator of 1 s is not print (1 million 1 s) but a recursive program that iterates over print (1) 1 million times, that is,

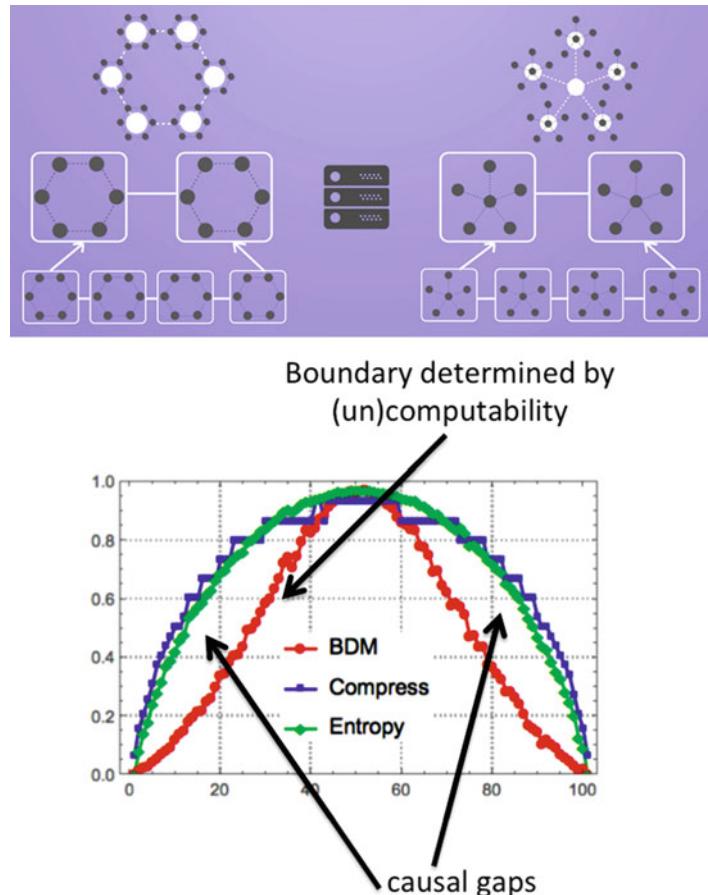
in this case, the shortest possible program. For this sequence the most likely and highest causal generating program can easily be guessed, but for most sequences this is very difficult. However, the more difficult the more random, thus giving us some clues as to the causal content of the object.

The combination of the CTM and BDM algorithms can also be seen as learning the parameters of small Turing machines. The reference machine size will always be a concern, given the invariance theorem that does not guarantee fast convergence. However, we have also shown (Soler-Toscano et al. 2014) that the method is well-behaved and seems to converge because in exploring greater rule spaces according to increasing Turing machine size, it converges in output distribution (Fig. 4).

BDM shows that the best solution to approximating  $K$  is to make the best use of both methods using each where relevant and/or practical and thus combining Entropy and algorithmic complexity (Fig. 5). One can decide to capture more algorithmic content with CTM alone but it becomes infeasible (ultimately uncomputable), and then one switches to Entropy, especially if one has already found that an object is of low Entropy, and thus low  $K$ . There is no need to waste resources only to rediscover that it is of low  $K$ .

For example, while compression algorithms may fail to find the algorithmic content of  $\pi$ , our method may give you a small clue that  $\pi$  is not truly random (maximal  $K$ ) as Entropy would otherwise suggest. Applying these more powerful measures across several strings, and not only testing against  $\pi$ , provides a handle that one would never reach with statistical tools. So if your segment of  $\pi$  is too long, then BDM will behave asymptotically, like Entropy alone if one does not update the pre-computed data from CTM. BDM is optimal for short strings, but for medium strings, from 100 to 1 K bits, it carries the CTM signal to find small signs of algorithmic content, thereby complementing lossless compression algorithms covering the full range of lengths and different information content approximations, showing that the best approaches are hybrid algorithms.

Given that the model machine we use is universal, we know that some machines will

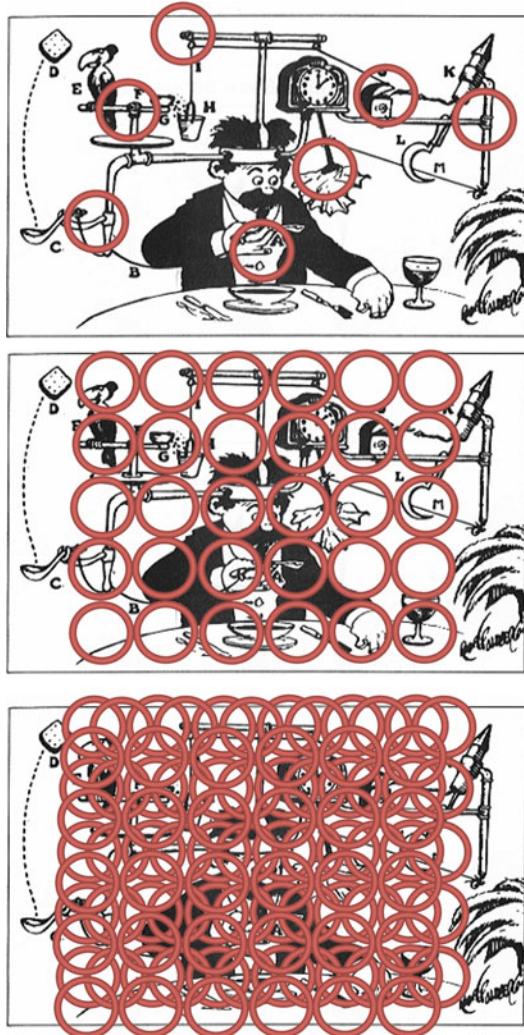


**Approximations to Algorithmic Probability, Fig. 4** *Top:* BDM decomposes objects into smaller constituents for which estimations to their algorithmic content have been pre-calculated by CTM. Estimations are then cleverly summed according to some of the BDM formulae (Zenil et al. 2016) to provide an approximation of the algorithmic content of the whole object. *Bottom:* Comparison of distributions of three different measures. Entropy and Compression produce Bernoulli distributions typical of statistical measures. The approximation of Algorithmic Probability by BDM, however, approximates a two-sided exponential distribution conforming to the theoretical expectation of a process capable of assigning lower

complexity to strings with higher algorithmic information content, even if random-looking. Examples of strings of higher Entropy (and high Entropy rate) but lower  $K$  are 010101110100, 011011010010. Further improvements for larger strings are bounded by uncomputability pushing towards Entropy when finite computable approximations are undertaken, with only infinite computational resources able to keep the gaps wide for longer strings. Approximations, however, provide clues to the algorithmic content of certain strings that no statistical method can reach, thereby constituting a benchmark for approaches attempting to go beyond statistical methods towards model-driven algorithmic/mechanistic models

implement algorithms generating subsequences of  $\pi$  because  $\pi$  is of low  $K$  and thus there are very short programs that can generate it (in full, let alone in pieces). To properly compare Entropy and  $K$  approximated by CTM/BDM, one has to normalize max entropy with max  $K$  and look at the distributions (see Fig. 4 bottom). One will see then

that Entropy approximates the classical Bernoulli distribution, while  $K$  with BDM approximates an exponential, just as the theory predicts (Levin's universal distribution or semi measure (Levin 1974)). The distribution would not yet yield the same order of strings, but one can see the strings that have max Entropy but are not max  $K$  – these



### Approximations to Algorithmic Probability,

**Fig. 5** The Block Decomposition Method (BDM) consists in discovering causal patches for which a computational model is found and thus low algorithmic complexity is assigned. Rube Goldberg machine. A Rube Goldberg machine is a deliberately complex contraption in which a series of devices that perform simple tasks are linked together to produce a domino effect in which activating one device triggers the next device in the sequence. The machine is named after American cartoonist and inventor of such contraptions, Rube Goldberg (1883–1970). c.f. Professor Butts and the Self-Operating Napkin (1931)

are the ones gained by applying our methods versus Entropy or compression. Performing a test for very long sequences of  $\pi$  without updating CTM eventually leads to BDM losing the algorithmic edge given by CTM (Zenil et al. 2016), unless ones reruns CTM for (exponentially) longer periods.

The small machine constitutes a mechanistic model because the Turing machine is mechanistic in nature and explains the small piece of data that it outputs upon halting. By putting all small Turing machines together, each explaining a small piece of the larger system, one can take the whole sequence and use it as a model approximation for the whole. If the small patches are causal and with small programs (Turing machines), the sequence will be shorter than the maximum  $K$  sequence (similar in size to the object itself), thus providing not only a local approximation of algorithmic complexity but also insight into the causal nature (or otherwise) of the larger system.

### Conclusion

We have explained how computability and algorithmic complexity are now at the center of the challenge of causality in science, and while it is more difficult to numerically evaluate, new methods have been advanced to estimate universal measures of complexity. Only universal measures of complexity can characterize any (computable) feature and can be invariant to object description and can encode, in principle, every possible computable function and therefore, potentially, every deterministic system following classical mechanics. It is therefore an extremely powerful measure tantamount to a computational theory of everything, and numerical methods to approximate it give an edge in finding key indications of causal content in a system, which is germane to understanding aspects of evolving causal systems and to developing algorithmic calculi to manipulate them.

### Bibliography

- Delahaye J-P, Zenil H (2012) Numerical evaluation of algorithmic complexity for short strings: a glance into the innermost structure of randomness. *Appl Math Comput* 219(1):63–77  
 Gauvrit N, Singmann H, Soler-Toscano F, Zenil H (2016) Algorithmic complexity for psychology: a user-friendly implementation of the coding theorem method. *Behav Res Methods* 48(1):314–329

- Gauvrit N, Soler-Toscano F, Zenil H (2014a) Natural scene statistics mediate the perception of image complexity. *Vis Cogn* 22(8):1084–1091
- Gauvrit N, Zenil H, Delahaye J-P, Soler-Toscano F (2014b) Algorithmic complexity for short binary strings applied to psychology: a primer. *Behav Res Methods* 46(3):732–744
- Gauvrit N, Zenil H, Soler-Toscano F, Delahaye J-P, Brugge P (2017) Human behavioral complexity peaks at age 25. *PLoS Comput Biol* 13(4):e1005408
- Gauvrit N, Zenil H, Tegnér J (2017) The information-theoretic and algorithmic approach to human, animal and artificial cognition. In: Dodig-Crnkovic G, Giovagnoli R (eds) *Representation and reality: humans, animals and machines*. Springer, New York
- Gregory J, Chaitin GJ (1966) On the length of programs for computing finite binary sequences. *J ACM* 13(4):547–569
- Kirchherr W, Li M, Vitányi P (1997) The miraculous universal distribution. *Math Intell* 19:7–15
- Levin LA (1974) Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Probl Peredachi Inf* 10(3):30–35
- Schmidhuber J (2002) Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *Int J Found Comput Sci* 13(4):587–612
- Shannon CE (1948) A mathematical theory of communication parts i and ii. *Bell Syst Tech J* 27:379–423. and 623–656
- Soler-Toscano F, Zenil H, Delahaye J-P, Gauvrit N (2014) Calculating Kolmogorov complexity from the output frequency distributions of small Turing machines. *PLoS One* 9(5):e96223
- Solomonoff RJ (1964) A formal theory of inductive inference. Part i. *Inf Control* 7(1):1–22
- Tegnér J, Zenil H, Kiani NA, Ball G, Gomez-Cabrero D (2016) A perspective on bridging scales and design of models using low-dimensional manifolds and data-driven model inference. *Phil Trans R Soc A* 374(2080):20160144
- Wolfram S (2002) *A new kind of science*. Wolfram Media, Champaign
- Zenil H (2011) The world is either algorithmic or mostly random, 2011. Winning 3rd place in the international essay context of the FQXi
- Zenil H (2013) A behavioural foundation for natural computing and a programmability test. In: Dodig-Crnkovic G, Giovagnoli R (eds) *Computing nature*. Springer, New York, pp 87–113
- Zenil H (2014a) Programmability: a Turing test approach to computation. In: L. De Mol, G. Primiero, (eds) *Turing in context*, Koninklijke Vlaamse Academie van België voor Wetenschappen en Kunsten (Belgian Academy of Sciences and Arts), Contactforum. Belgian Academy of Sciences and Arts
- Zenil H (2014b) What is nature-like computation? A behavioural approach and a notion of programmability. *Philos Technol* 27(3):399–421
- Zenil H (2015) Algorithmicity and programmability in natural computing with the game of life as in silico case study. *J Exp Theor Artif Intell* 27(1):109–121
- Zenil H, Delahaye J-P (2010) On the algorithmic nature of the world. In: Dodig-Crnkovic G, Burgin M (eds) *Information and computation*. World Scientific, London
- Zenil H, Kiani NA, Tegnér J (2016) Methods of information theory and algorithmic complexity for network biology. *Semin Cell Dev Biol* 51:32–43
- Zenil H, Kiani N, Jesper T (2017) Low algorithmic complexity entropy-deceiving graphs. *Phys Rev E* 96(012308)
- Zenil H, Soler-Toscano F, Dingle K, Louis AA (2014) Correlation of automorphism group size and topological properties with program-size complexity evaluations of graphs and complex networks. *Physica A Stat Mech Appl* 404:341–358
- Zenil H, Soler-Toscano F, Kiani NA, Hernández-Orozco S, Rueda-Toicen A (2016) A decomposition method for global evaluation of Shannon entropy and local estimations of algorithmic complexity. arXiv preprint arXiv:1609.00110
- Zenil H. (2017) Algorithmic Data Analytics, Small Data Matters and Correlation versus Causation. In M. Ott, W. Pietsch, J. Wernecke (eds.), *Berechenbarkeit der Welt? Philosophie und Wissenschaft im Zeitalter von Big Data (Computability of the World? Philosophy and Science in the Age of Big Data)*, Springer Verlag, pp. 453–475



## Grossone Infinity Computing

Yaroslav D. Sergeyev

University of Calabria, Rende, Italy

Lobachevsky State University, Nizhni Novgorod,

Russia

Institute of High Performance Computing and Networking, C.N.R, Rome, Italy

### Article Outline

Glossary

Introduction

Numerical Systems, Their Accuracy, and Numbers

They Can Express

Grossone-Based Numerals

Measuring Infinite Sets and Relations to

Bijections

Concluding Remarks

Bibliography

### Keywords

Numbers and numerals; Grossone-based numerals; Numerical infinities and infinitesimals; Infinite sets

## Glossary

**Extended natural numbers** Positive integers larger than grossone.

**Grossone** The number of elements of the set of natural numbers expressed by the symbol ①. It allows to represent this set as  $\{1, 2, 3, \dots\} = 2, ① - 1, ①\}$ .

**Infinity computer** A new supercomputer patented in USA, Europe, and Russia allowing one to work numerically within finite, infinite, and infinitesimal numbers.

**Infinity computing** Numerical computations on the Infinity Computer using the principle “The

part is less than the whole” applied to all numbers (finite, infinite, and infinitesimal) expressed in the positional numeral system with the base ①.

**Numbers and numerals** A *numeral* is a symbol (or a group of symbols) that represents a *number*. A *number* is a concept that a *numeral* expresses.

**Pirahã** A tribe living in Amazonia nowadays.

These people use an extremely simple numeral system for counting: one, two, and many. For Pirahã, all quantities larger than two are just “many” and such operations as  $2 + 2$  and  $2+1$  give the same result, i.e., “many.” Using their weak numeral system Pirahã are not able to see, for instance, numbers 3, 4, and 5, to execute arithmetical operations with them, and, in general, to say anything about these numbers because in their language there are neither words nor concepts for that.

## Introduction

There exists an important distinction between *numbers* and *numerals*. A *numeral* is a symbol (or a group of symbols) that represents a *number*. A *number* is a concept that a *numeral* expresses. The same number can be represented by different numerals. For example, the symbols “10,” “ten,” “IIIIIIII,” “X,” “=,” and “I” are different numerals, but they all represent the same number. (The last two numerals, = and I, are probably less known. The former belongs to the Maya numeral system where one horizontal line indicates five and two lines one above the other indicate ten. Dots are added above the lines to represent additional units. For instance, = means eleven in this numeral system. The latter symbol, I, belongs to the Cyrillic numeral system derived from the Cyrillic script. This numeral system was developed in the late tenth century and was used by South and East Slavic peoples. The system was used in Russia as late as the early eighteenth century when it was replaced with Arabic numerals. To distinguish numbers from text, a titlo, ~, is drawn over the

symbols showing so that this is a numeral and, therefore, it represents a number and not just a character of text.) Rules used to write down numerals together with algorithms for executing arithmetical operations form a *numeral system*.

In our everyday activities with finite numbers, the *same* finite numerals are used for *different* purposes (e.g., the same numeral 10 can be used to express the number of elements of a set, to indicate the position of an element in a sequence, and to execute practical computations). In contrast, when we face the necessity to work with infinities or infinitesimals, the situation changes drastically. In fact, in this case, *different* numerals are used to work with infinities and infinitesimals in *different* situations. To illustrate this fact, it is sufficient to mention that we use the symbol  $\infty$  in standard analysis,  $\omega$  for working with ordinals, and  $\aleph_0, \aleph_1, \dots$  for dealing with cardinalities.

Many theories dealing with infinite and infinitesimal quantities have a symbolic (not numerical) character. For instance, many versions of nonstandard analysis (see Robinson 1996) are symbolic, since they have no numeral systems to express their numbers by a finite number of symbols (the finiteness of the number of symbols is necessary for organizing numerical computations). Namely, if we consider a finite  $n$ , then it can be taken as  $n = 134$  or  $n = 65$  or any other numeral used to express finite quantities and consisting of a finite number of symbols. In contrast, if we consider a nonstandard infinite  $m$ , then it is not clear which numerals can be used to assign a concrete value to  $m$ .

Analogously, in nonstandard analysis, if we consider an infinitesimal  $h$ , then it is not clear which numerals consisting of a finite number of symbols can be used to assign a value to  $h$  and to write  $h = \dots$ . In fact, very often in nonstandard analysis texts, a *generic* infinitesimal  $h$  is used, and it is considered as a symbol, i.e., only symbolic computations can be done with it. Approaches of this kind leave unclear such issues, e.g., whether the infinite  $1/h$  is an integer or not or whether  $1/h$  is the number of elements of an infinite set. If one wishes to consider two infinitesimals  $h_1$  and  $h_2$ , then it is not clear how to compare them because numeral systems that can express infinitesimals are not provided by nonstandard analysis techniques. In fact, when we work with finite quantities,

then we can compare  $x$  and  $y$  if they assume numerical values, e.g.,  $x = 25$  and  $y = 78$ ; then, by using rules of the numeral system the symbols 25 and 78 belong to, we can compute that  $y > x$ .

Even though there exist codes allowing one to work symbolically with  $\infty$  and other symbols related to the concepts of infinity and infinitesimals, traditional computers work numerically only with finite numbers and situations where the usage of infinite or infinitesimal quantities is required are studied mainly theoretically (see Cantor 1955; Conway and Guy 1996; Gödel 1940; Hardy 1910; Hilbert 1902; Leibniz and Child 2005; Levi-Civita 1898; Robinson 1996; Wallis 1656 and references given therein). The fact that numerical computations with infinities and infinitesimals have not been implemented so far on computers can be explained by several difficulties. Obviously, among them, we can mention the fact that arithmetics developed for this purpose are quite different with respect to the way of computing we use when we deal with finite quantities. For instance, there exist undetermined operations ( $\infty - \infty, \frac{\infty}{\infty}$ , etc.) that are absent when we work with finite numbers. There exist also practical difficulties that preclude an implementation of numerical computations with infinity and infinitesimals. For example, it is not clear how to store an infinite quantity in a finite computer memory.

A computational methodology introduced recently in (Sergeyev 2008, 2010c, 2013b, 2015c) allows one to look at infinities and infinitesimals in a new way and to execute *numerical* computations with infinities and infinitesimals on the Infinity Computer patented in the USA (see Sergeyev 2010a) and other countries (the interested reader is invited to have a look also at the book (Sergeyev 2013a) written in a popular way). Moreover, this approach proposes a numeral system that uses *the same numerals* for several different purposes for dealing with infinities and infinitesimals: for measuring infinite sets; for indicating positions of elements in ordered infinite sequences; for working with functions and their derivatives that can assume different infinite, finite, and infinitesimal values and can be defined over infinite and infinitesimal domains; and for describing Turing machines, etc.

An international scientific community developing a number of interesting theoretical and practical

applications in several research areas by using the new methodology grows rapidly. A number of papers studying connections of the new approach to the historical panorama of ideas dealing with infinities and infinitesimals (see Lolli 2012; Margenstern 2011; Sergeyev and Garro 2010; Montagna et al. 2015) have been published. In particular, metamathematical investigations on the new theory and its noncontradictory can be found in (Lolli 2015). The methodology has been successfully applied in several areas of mathematics and computer science: single and multiple criteria optimization (see Cococcioni et al. 2016; De Cosmis and De Leone 2012; De Leone R 2017; Žilinskas 2012); cellular automata (see D’Alotto 2012, 2013, 2015); Euclidean and hyperbolic geometry (see Margenstern 2012, 2015); percolation (see Iudin et al. 2012, 2015; Vita et al. 2012); fractals (see Sergeyev 2007, 2009a, 2011c, 2016; Vita et al. 2012); infinite series and the Riemann zeta function (see Sergeyev 2009c, 2011b; Zhitljavsky 2012); the first Hilbert problem, Turing machines, and supertasks (see Rizza 2016; Sergeyev 2010b; Sergeyev and Garro 2010, 2013); numerical differentiation and numerical solution of ordinary differential equations (see Sergeyev 2011a, 2013c, 2016; Amodio et al. 2016; Mazzia et al. 2016); etc. Some of these applications will be discussed briefly hereinafter.

## **Numeral Systems, Their Accuracy, and Numbers They Can Express**

It is necessary to remind that different numeral systems can express different sets of numbers and they can be more or less suitable for executing arithmetical operations. Even the powerful positional system is not able to express, e.g., the number  $\sqrt{2}$  by a finite number of symbols (the finiteness is essential for executing numerical computations), and this special numeral,  $\sqrt{2}$ , is deliberately introduced to express the desired quantity. There exist many numeral systems that are weaker than the positional one. For instance, Roman numeral system is not able to express zero and negative numbers, and such expressions as III – VIII or X-X are indeterminate forms in this numeral system. As a result, before appearing,

the positional numeral system and inventing zero mathematicians were not able to create theorems involving zero and negative numbers and to execute computations with them. Thus, numeral systems seriously bound the possibilities of human beings to compute, and developing new, more powerful than existing ones, numeral systems can help a lot both in theory and practice of computations.

Even though Roman numeral system is weaker than the positional one, it is not the weakest numeral system. There exist really feeble numeral systems allowing their users to express very few numbers, and one of them is illuminating for our study. This numeral system is used by a tribe, Pirahā, living in Amazonia nowadays. A study published in *Science* in 2004 (see Gordon 2004) describes that these people use an extremely simple numeral system for counting: one, two, and many. For Pirahā, all quantities larger than two are just “many,” and such operations as  $2 + 2$  and  $2 + 1$  give the same result, i.e., “many.” Using their weak numeral system, Pirahā are not able to see, for instance, numbers 3, 4, and 5, to execute arithmetical operations with them and, in general, to say anything about these numbers because in their language there are neither words nor concepts for that.

It is worthy to mention that the result “many” is not wrong. It is just *inaccurate*. Analogously, when we observe a garden with 343 trees, then both sentences “There are 343 trees in the garden” and “There are many trees in the garden” are correct. However, the accuracy of the former phrase is higher than the accuracy of the latter one. Thus, the introduction of a numeral system having numerals for expressing numbers 3 and 4 leads to a higher accuracy of computations and allows one to distinguish results of operations  $2 + 1$  and  $2 + 2$ .

The poverty of the numeral system of Pirahā leads also to the following results

$$\begin{aligned} \text{“many”} + 1 &= \text{“many,”} & \text{“many”} + 2 &= \text{“many,”} \\ \text{“many”} - 1 &= \text{“many,”} & \text{“many”} - 2 &= \text{“many”} \\ \text{“many”} + \text{“many”} &= \text{“many”} \end{aligned}$$

that are crucial for changing our outlook on infinity. In fact, by changing in these relations “many” with  $\infty$ , we get relations used to work with infinity in the traditional calculus and Cantor’s cardinals:

$$\infty + 1 = \infty, \quad \infty + 2 = \infty, \quad \infty - 1 = \infty, \quad \infty - 2 = \infty, \quad \infty + \infty = \infty \\ \aleph_0 + 1 = \aleph_0, \quad \aleph_0 + 2 = \aleph_0, \quad \aleph_0 - 1 = \aleph_0, \quad \aleph_0 - 2 = \aleph_0, \quad \aleph_0 + \aleph_0 = \aleph_0.$$


---

It should be mentioned that the astonishing numeral system of Pirahā is not an isolated example of this way of counting. In fact, the same counting system, that is, one, two, many, is used by the Warlpiri people, aborigines living in the Northern Territory of Australia (see Butterworth et al. 2008). The Pitjantjatjara people living in the Central Australian desert use numerals one, two, three, and big mob (see Leder 2015), where “big mob” works as “many.” It makes sense to remind also another Amazonian tribe – Mundurukú (see Pica et al. 2004) – who fail in exact arithmetic with numbers larger than 5 but are able to compare and add large approximate numbers that are far beyond their naming range. Particularly, they use the words “some, not many” and “many, really many” to distinguish two types of large numbers. Their arithmetic with “some, not many” and “many, really many” reminds strongly the rules Cantor uses to work with  $\aleph_0$  and  $\aleph_1$ , respectively. For instance, compare

$$\begin{aligned} & \text{“some, not many”} + \text{“many, really many”} \\ &= \text{“many, really many”} \end{aligned}$$

with

$$\aleph_0 + \aleph_1 = \aleph_1.$$

This comparison suggests that our difficulty in working with infinity is not connected to the *nature* of infinity but is a result of inadequate numeral systems used to express infinite numbers. Traditional numeral systems have been developed to express finite quantities, and they simply have no sufficiently high number of numerals to express different infinities (and infinitesimals). In other words, the difficulty we face is not connected to the object of our study – infinity – but is the result of weak instruments, numeral systems, used for our study.

The way of reasoning where the object of the study is separated from the tool used by the investigator is very common in natural sciences where researchers use tools to describe the object of their study and the used instrument influences the results of the observations and determine their accuracy. When a physicist uses a weak lens  $A$  and sees two black dots in his/her microscope, he/she does not say: The object of the observation is two black dots. The physicist is obliged to say: The lens used in the microscope allows us to see two black dots, and it is not possible to say anything more about the nature of the object of the observation until we change the instrument – the lens or the microscope itself – by a more precise one. Suppose that he/she changes the lens and uses a stronger lens  $B$  and is able to observe that the object of the observation is viewed as eleven (smaller) black dots. Thus, we have two different answers: (i) the object is viewed as two dots if the lens  $A$  is used; (ii) the object is viewed as eleven dots by applying the lens  $B$ . Both answers are correct but with the *different accuracies* that depend on the lens used for the observation.

The same happens in mathematics studying natural phenomena, numbers, objects that can be constructed by using numbers, sets, etc. Numeral systems used to express numbers are among the instruments of observations used by mathematicians. As we have illustrated above, the usage of powerful numeral systems gives the possibility to obtain more precise results in mathematics in the same way as usage of a good microscope gives the possibility of obtaining more precise results in Physics.

## Grossone-Based Numerals

In order to increase the accuracy of computations with infinities and infinitesimals, the computational methodology developed in (Sergeyev

2008, 2010c, 2013a) proposes a numeral system that allows one to observe infinities and infinitesimals with a higher accuracy. This numeral system avoids situations similar to “many” + 1 = “many” and  $\infty - 1 = \infty$  providing results ensuring that if  $a$  is a numeral written in this numeral system then for any  $a$  (i.e.,  $a$  can be finite, infinite, or infinitesimal) it follows  $a + 1 > a$  and  $a - 1 < a$ .

The numeral system is based on a new infinite unit of measure expressed by the numeral  $\mathbb{1}$  called *grossone* that is introduced as the number of elements of the set of natural (Notice that nowadays not only positive integers but also zero is frequently included in  $\mathbb{N}$ . However, since zero has been invented significantly later than positive integers used for counting objects, zero is not include in  $\mathbb{N}$  in this text.) numbers

$$\mathbb{N} = \{1, 2, 3, \dots\}. \quad (1)$$

Concurrently with the introduction of  $\mathbb{1}$  in the mathematical language all other symbols (like  $\infty$ , Cantor's  $\omega$ ,  $\aleph_0, \aleph_1, \dots$ , etc.) traditionally used to deal with infinities and infinitesimals are excluded from the language because  $\mathbb{1}$  and other numbers constructed with its help not only can be used instead of all of them but can be used with a higher accuracy. Analogously, when zero and the positional numeral system had been introduced in Europe, Roman numerals I, V, X, etc. had not been involved and new symbols 0, 1, 2, etc. have been used to express numbers. The new element – zero expressed by the numeral 0 – had been introduced by describing its properties in the form of axioms. Analogously,  $\mathbb{1}$  is introduced by describing its properties postulated by the Infinite Unit Axiom added to axioms for real numbers (see Sergeyev 2008, 2010c for a detailed discussion). Let us comment upon some of properties of  $\mathbb{1}$ .

If we consider a finite integer  $k$ , then the number of elements of the set  $\{1, 2, 3, \dots, k-1, k\}$  is its largest element, i.e.,  $k$ . For instance, the number 4 in the set

$$A = \{1, 2, 3, 4\} \quad (2)$$

is the largest element in the set  $A$  and the number of elements of  $A$ . Grossone has been introduced as the

number of elements of the set of natural numbers, and, therefore, we have the same situation as in (Eq. 2), i.e.,  $\mathbb{1} \in \mathbb{N}$ . As a consequence, the introduction of  $\mathbb{1}$  allows us to write down the set of natural numbers as follows:

$$\mathbb{N} = \left\{ 1, 2, \dots, \frac{\mathbb{1}}{2} - 2, \frac{\mathbb{1}}{2} - 1, \frac{\mathbb{1}}{2}, \frac{\mathbb{1}}{2} + 1, \frac{\mathbb{1}}{2} + 2, \dots, \mathbb{1} - 2, \mathbb{1} - 1, \mathbb{1} \right\}. \quad (3)$$

### Infinite natural numbers

$$\dots, \frac{\mathbb{1}}{2} - 2, \frac{\mathbb{1}}{2} - 1, \frac{\mathbb{1}}{2}, \frac{\mathbb{1}}{2} + 1, \frac{\mathbb{1}}{2} + 2, \dots, \mathbb{1} - 2, \mathbb{1} - 1, \mathbb{1} \quad (4)$$

that are invisible if traditional numeral systems are used to observe the set of natural numbers can be viewed now, thanks to the introduction of  $\mathbb{1}$ . The two records (Eqs. 1 and 3) refer to the same set – the set of natural numbers – and infinite numbers (Eq. 4) also take part (This is a difference with respect to nonstandard analysis where infinities it works with do not belong to  $\mathbb{N}$ ) of  $\mathbb{N}$ . Both records (Eqs. 1 and 3) are correct and do not contradict each other. They just use two different numeral systems to express  $\mathbb{N}$ . Traditional numeral systems do not allow us to see infinite natural numbers that we can observe now, thanks to  $\mathbb{1}$ . Thus, we have the same object of observation – the set  $\mathbb{N}$  – that can be observed by different instruments, numeral systems, with different accuracies.

Similarly, Pirahā are not able to see finite natural numbers 3, 4, and 5. In spite of the fact that Pirahā do not see them, these numbers 3, 4, and 5, belong to  $\mathbb{N}$  and are visible if one uses a more powerful numeral system. Even the numeral system of Mundurukú is sufficient to observe 3, 4, and 5. Notice also that the weakness of their numeral system does not allow Pirahā to define the set (Eq. 2) while Mundurukú would be able to do this.

In general, in the new methodology, it is necessary always to indicate a numeral system used for computations and theoretical investigations. For instance, the words “the set of all finite numbers”

do not define a set completely in this methodology. It is always necessary to specify which instruments (numeral systems) are used to describe (and to observe) the required set and, as a consequence, to speak about “the set of all finite numbers expressible in a fixed numeral system.” For instance, for Pirahā and Warlpiri, “the set of all finite numbers” is the set {1, 2}; for the Pitjantjatjara people, “the set of all finite numbers” is the set {1, 2, 3}; and for Mundurukú, “the set of all finite numbers” is the set {1, 2, 3, 4, 5}. We stress again that in Mathematics, as it happens in physics, the instrument used for an observation bounds the possibility of the observation and defines the accuracy of this observation. It is not possible to say how we shall see the object of our observation if we have not clarified which instruments will be used to execute the observation.

Let us see now how one can write down different numerals expressing different infinities and infinitesimals and to execute computations with all of them. Instead of the usual symbol  $\infty$ , different infinite and/or infinitesimal numerals can be used, thanks to ①. Indeterminate forms are not present, and, for example, the following relations hold for infinite numbers ①,  $\mathbb{I}^2$  and  $\mathbb{I}^{-1}$ ,  $\mathbb{I}^{-2}$  (that are infinitesimals), as for any other (finite, infinite, or infinitesimal) number expressible in the new numeral system

$$\begin{aligned} 0.\mathbb{I} &= \mathbb{I}.0 = 0, \mathbb{I} - \mathbb{I} = 0, \frac{\mathbb{I}}{\mathbb{I}} = 1, \mathbb{I}^0 = 1, 1\mathbb{I} = 1, 0\mathbb{I} = 0, \\ 0.\mathbb{I}^{-1} &= \mathbb{I}^{-1}.0 = 0, \mathbb{I}^{-1} > 0, \mathbb{I}^{-2} > 0, \mathbb{I}^{-1} - \mathbb{I}^{-1} = 0, \\ \frac{\mathbb{I}^{-1}}{\mathbb{I}^{-1}} &= 1, (\mathbb{I}^{-1})^0 = 1, \mathbb{I}.\mathbb{I}^{-1} = 1, \mathbb{I}.\mathbb{I}^{-2} = \mathbb{I}^{-1}, \\ \frac{\mathbb{I}^{-2}}{\mathbb{I}^{-2}} &= 1, \frac{\mathbb{I}^2}{\mathbb{I}} = \mathbb{I}, \frac{\mathbb{I}^{-1}}{\mathbb{I}^{-2}} = \mathbb{I}, \mathbb{I}^2.\mathbb{I}^{-1} = \mathbb{I}, \mathbb{I}^2.\mathbb{I}^{-2} = 1. \end{aligned}$$

The introduction of the numeral ① allows us to represent infinite and infinitesimal numbers in a unique framework. For this purpose, a numeral system similar to traditional positional numeral systems was introduced in (Sergeyev 2008, 2013a). To construct a number  $C$  in the numeral positional system with base ①, we subdivide  $C$  into groups corresponding to powers of ①:

$$\begin{aligned} C &= c_{pm}\mathbb{I}^{pm} + \dots + c_{p1}\mathbb{I}^{p1} + c_{p0}\mathbb{I}^{p0} \\ &\quad + c_{p-1}\mathbb{I}^{p-1} + \dots + c_{p-k}\mathbb{I}^{p-k} \end{aligned} \quad (5)$$

Then, the record

$$C = c_{pm}\mathbb{I}^{pm} \dots c_{p1}\mathbb{I}^{p1} c_{p0}\mathbb{I}^{p0} c_{p-1}\mathbb{I}^{p-1} \dots c_{p-k}\mathbb{I}^{p-k} \quad (6)$$

represents the number  $C$ , where all numerals  $c_i \neq 0$ , they belong to a traditional numeral system and are called *grossdigits*. They express finite positive or negative numbers and show how many corresponding units  $\mathbb{I}^{pi}$  should be added or subtracted in order to form the number  $C$ . Note that in order to have a possibility to store  $C$  in the computer memory, values  $k$  and  $m$  should be finite.

Numbers  $p_i$  in (Eq. 6) are sorted in the decreasing order with  $p_0 = 0$

$$\begin{aligned} p_m &> p_{m-1} > \dots > p_1 > p_0 > p_{-1} > \\ &\dots p - (k - 1) > p - k. \end{aligned}$$

They are called *grosspowers*, and they themselves can be written in the form (Eq. 6). In the record (Eq. 6), we write  $\mathbb{I}^{pi}$  explicitly because in the new numeral positional system the number  $i$  in general is not equal to the grosspower  $p_i$ . This gives the possibility to write down numerals without indicating grossdigits equal to zero.

The term having  $p_0 = 0$  represents the finite part of  $C$  since  $c_0\mathbb{I}^0 = c_0$ . Terms having finite positive grosspowers represent the simplest infinite parts of  $C$ . Analogously, terms having negative finite grosspowers represent the simplest infinitesimal parts of  $C$ . For instance, the number  $\mathbb{I}^{-1} = \frac{1}{\mathbb{I}}$  mentioned above is infinitesimal. Note that all infinitesimals are not equal to zero. In particular,  $\frac{1}{\mathbb{I}} > 0$  since it is a result of division of two positive numbers.

A number represented by a numeral in the form (Eq. 6) is called *purely finite* if it has neither infinite nor infinitesimals parts. For instance, 14 is purely finite and  $14 + 5.3\mathbb{I}^{-1.5}$  is not. All grossdigits  $c_i$  are supposed to be purely finite. Purely finite numbers are used on traditional computers and for obvious reasons have a special importance for applications. All of the numbers introduced above can be grosspowers, as well,

**Grossone Infinity Computing, Table 1** Measuring infinite sets using  $\mathbb{I}$ -based numerals allows one in certain cases to obtain more precise answers in comparison with the traditional cardinalities,  $\aleph_0$  and  $C$ , of Cantor

Description of sets	Cardinality	Number of elements
The set of natural numbers $\mathbb{N}$	Countable, $\aleph_0$	$\mathbb{I}$
$\mathbb{N} \cup \{0\}$	Countable, $\aleph_0$	$\mathbb{I} + 1$
$\mathbb{N} \setminus \{3, 5, 10, 23, 114\}$	Countable, $\aleph_0$	$\mathbb{I} - 5$
The set of even numbers $\mathbb{E}$	Countable, $\aleph_0$	$\frac{\mathbb{I}}{2}$
The set of odd numbers $\mathbb{O}$	Countable, $\aleph_0$	$\frac{\mathbb{I}}{2}$
The set of integers $\mathbb{Z}$	Countable, $\aleph_0$	$2\mathbb{I} + 1$
$\mathbb{Z} \setminus \{0\}$	Countable, $\aleph_0$	$2\mathbb{I}$
The set of square natural numbers $\mathbb{G} = \{x : x = n^2, x \in \mathbb{N}, n \in \mathbb{N}\}$	Countable, $\aleph_0$	$\lfloor \sqrt{\mathbb{I}} \rfloor$
The set of pairs of natural numbers $\mathbb{P} = \{(p, q) : p \in \mathbb{N}, q \in \mathbb{N}\}$	Countable, $\aleph_0$	$\mathbb{I}^2$
The set of numerals $\mathbb{Q}' = \left\{-\frac{p}{q}, \frac{p}{q} : p \in \mathbb{N}, q \in \mathbb{N}\right\}$	Countable, $\aleph_0$	$2\mathbb{I}^2$
The set of numerals $\mathbb{Q} = \left\{0 - \frac{p}{q}, \frac{p}{q} : p \in \mathbb{N}, q \in \mathbb{N}\right\}$	Countable, $\aleph_0$	$2\mathbb{I}^2 + 1$
The set of numerals $A_2$	Continuum, $\mathcal{C}$	$2\mathbb{I}$
The set of numerals $A'_2$	Continuum, $\mathcal{C}$	$2^{\mathbb{I}} + 1$
The set of numerals $A_{10}$	Continuum, $\mathcal{C}$	$10^{\mathbb{I}}$
The set of numerals $C_{10}$	Continuum, $\mathcal{C}$	$2 \cdot 10^{\mathbb{I}}$

giving thus a possibility to have various combinations of quantities and to construct terms having a more complex structure.

We conclude this section by emphasizing that different numeral systems, if they have different accuracies, cannot be used together. For instance, the usage of *many* from the language of Pirahã in the record  $5 + \text{many}$  has no any sense because for Pirahã it is not clear what 5 is and for people knowing what 5 is the accuracy of the answer “many” is too low. Analogously, the records of the type  $\mathbb{I} + \omega$ ,  $\mathbb{I} - \aleph_0$ ,  $\mathbb{I}/\infty$ , etc. have no sense because they include numerals developed under different methodological assumptions, in different mathematical contests, for different purposes, and, finally, numeral systems these numerals belong to have different accuracies.

## Measuring Infinite Sets and Relations to Bijections

By using the  $\mathbb{I}$ -based numeral system, it becomes possible to measure certain infinite sets. As we have seen above, relations of the type

“many” + 1 = “many” and  $\aleph_0 - 1 = \aleph_0$  are consequences of the weakness of numeral systems applied to express numbers (finite or infinite). Thus, one of the principles of the new computational methodology consists of adopting the principle “the part is less than the whole” to all numbers (finite, infinite, and infinitesimal) and to all sets and processes (finite and infinite). Notice that this principle is a reformulation of Euclid’s Common Notion 5 saying “the whole is greater than the part.”

Let us show how, in comparison to the traditional mathematical tools used to work with infinity, the new numeral system allows one to obtain more precise answers in certain cases. For instance, Table 1 compares results obtained by the traditional Cantor’s cardinals and the new numeral system with respect to the measure of a dozen of infinite sets (for a detailed discussion regarding the results presented in Table 1 and for more examples dealing with infinite sets, see Lolli 2015; Margenstern 2011; Sergeyev 2010b, c; Sergeyev and Garro 2010). Notice that in  $\mathbb{Q}$  and  $\mathbb{Q}'$ , we calculate different numerals and not numbers. For instance, numerals  $\frac{4}{1}$  and  $\frac{8}{2}$  have been counted two times even though they represent the same number 4. Then, four sets of numerals having

the cardinality of continuum are shown in Table 1 (these results are discussed more in detail in the next section). Among them, we denote by  $A_2$  the set of numbers  $x \in [0, 1)$  expressed in the binary positional numeral system, by  $A'_2$  the set being the same as  $A_2$  but with  $x$  belonging to the closed interval  $[0, 1]$ , and by  $A_{10}$  the set of numbers  $x \in [0, 1)$  expressed in the decimal positional numeral system, and finally we have the set  $C_{10} = A_{10} \cup B_{10}$ , where  $B_{10}$  is the set of numbers  $x \in [1, 2)$  expressed in the decimal positional numeral system. It is worthwhile to notice also that grossone-based numbers from Table 1 can be ordered as follows:

$$\begin{aligned} [\sqrt{\mathbb{1}}] &< \frac{1}{2} < \mathbb{1} - 5 < \mathbb{1} < 2\mathbb{1} < 2\mathbb{1} + 1 < \\ \mathbb{1}^2 &< 2\mathbb{1}^2 + 1 < 2^{\mathbb{1}} < 2^{\mathbb{1}} + 1 < 10^{\mathbb{1}} < 2 \cdot 10^{\mathbb{1}}. \end{aligned}$$

It can be seen from Table 1 that Cantor's cardinalities say only whether a set is countable or uncountable, while the  $\mathbb{1}$ -based numerals allow us to express the exact number of elements of the infinite sets. However, both numeral systems – the new one and the numeral system of infinite cardinals – do not contradict one another. Both Cantor's numeral system and the new one give correct answers, but their answers have *different accuracies*. By using an analogy from physics, we can say that the lens of our new “telescope” used to observe infinities and infinitesimals is stronger and where Cantor's “telescope” allows one to distinguish just two dots (countable sets and the continuum) we are able to see many different dots (infinite sets having different number of elements).

The  $\mathbb{1}$ -based numeral system, as all numeral systems, cannot express all numbers and give answers to all questions. Let us consider, for instance, the set of *extended natural numbers* indicated as  $\hat{\mathbb{N}}$  and including  $\mathbb{N}$  as a proper subset:

$$\begin{aligned} \hat{\mathbb{N}} = \left\{ \underbrace{1, 2, \dots, \mathbb{1} - 1, \mathbb{1}}_{\text{Natural numbers}}, \right. \\ \left. \mathbb{1} + 1, \mathbb{1} + 2, \dots, 2\mathbb{1} - 1, 2\mathbb{1}, 2\mathbb{1}, \right. \\ \left. + 1, \dots, \mathbb{1}^2 - 1, \mathbb{1}^2, \mathbb{1}^2 + 1, \dots, 3\mathbb{1}^{\mathbb{1}} \right. \\ \left. - 1, 3\mathbb{1}^{\mathbb{1}}, 3\mathbb{1}^{\mathbb{1}} + 1, \dots \right\} \quad (7) \end{aligned}$$

What can we say with respect to the number of elements of the set  $\hat{\mathbb{N}}$ ? The introduced numeral system based on ① is too weak to give an answer to this question. It is necessary to introduce in a way a more powerful numeral system by defining new numerals (for instance, ②, ③, etc.).

In order to see how the principle “the part is less than the whole” agrees with traditional views on infinite sets, let us consider two illustrative examples. The first of them is related to the one-to-one correspondence that can be established between the sets of natural and odd numerus. Namely, odd numbers can be put in a one-to-one correspondence with all natural numbers in spite of the fact that  $\mathbb{O}$  is a proper subset of  $\mathbb{N}$

$$\begin{aligned} \text{odd numbers: } & 1, 3, 5, 7, 9, 11, \dots \\ & \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\ \text{natural numbers: } & 1, 2, 3, 4, 5, 6, \dots \end{aligned} \quad (8)$$

The usual conclusion is that both sets are countable and they have the same cardinality  $\aleph_0$ .

Let us see now what we can say from the new methodological positions. We know now that when one executes the operation of counting, the accuracy of the result depends on the numeral system used for counting. Proposing to Pirahā to measure sets consisting of four apples and five apples would give us the answer that both sets of apples have many elements. This answer is correct, but its precision is low due to the weakness of the numeral system used to measure the sets.

Thus, the introduction of the notion of accuracy for measuring sets is very important and should be applied for infinite sets also. Since for cardinal numbers it follows

$$\aleph_0 + 1 = \aleph_0, \quad \aleph_0 + 2 = \aleph_0, \quad \aleph_0 + \aleph_0 = \aleph_0,$$

these relations suggest that the accuracy of the cardinal numeral system of Alephs is not sufficiently high to see the difference with respect to the number of elements of the two sets from (Eq. 8).

In order to look at the record (Eq. 8) using the new numeral system, we need the following fact from (Sergeyev 2013a): the sets of even and odd

numbers have  $\mathbb{1}/2$  elements each, and, therefore,  $\mathbb{1}$  is even. It is also necessary to remind that numbers that are larger than  $\mathbb{1}$  are not natural, but they are extended natural numbers. For instance,  $\mathbb{1} + 1$  is odd but not natural, but it is extended natural; see (Eq. 7). Thus, the last odd natural number is  $\mathbb{1} - 1$ . Since the number of elements of the set of odd numbers is equal to  $\frac{\mathbb{1}}{2}$ , we can write down not only initial (as it is usually done traditionally) but also the final part of (Eq. 8):

$$\begin{array}{ccccccccc} 1, & 3, & 5, & 7, & 9, & 11, & \dots & \mathbb{1}-5, & \mathbb{1}-3, & \mathbb{1}-1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \uparrow \\ 1, & 2, & 3, & 4, & 5, & 6, & \dots & \frac{\mathbb{1}}{2}-2, & \frac{\mathbb{1}}{2}-1, & \frac{\mathbb{1}}{2} \end{array} \quad (9)$$

concluding so (Eq. 8) in a complete accordance with the principle “the part is less than the whole.” Both records (Eqs. 8 and 9) are correct but (Eq. 9) is more accurate, since it allows us to observe the final part of the correspondence that is invisible if (Eq. 8) is used.

The accuracy of the  $\mathbb{1}$ -based numeral system allows us to measure also, for instance, such sets as  $\mathbb{O}' = \mathbb{O} \setminus \{3\}$  and  $\mathbb{O}'' = \mathbb{O} \setminus \{1, \mathbb{1} - 1\}$ . The set  $\mathbb{O}'$  is constructed by excluding one element from  $\mathbb{O}$  and the set  $\mathbb{O}''$  by excluding from  $\mathbb{O}$  two elements. Thus,  $\mathbb{O}'$  and  $\mathbb{O}''$  have  $\frac{\mathbb{1}}{2} - 1$  and  $\frac{\mathbb{1}}{2} - 2$  elements, respectively. In case one wishes to establish the corresponding bijections, starting with natural numbers 1, 2, 3, ..., we obtain for these two sets:

$$\begin{array}{ccccccccc} 1, & 5, & 7, & 9, & 11, & 13, & \dots & \mathbb{1}-5, & \mathbb{1}-3, & \mathbb{1}-1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \uparrow \\ 1, & 2, & 3, & 4, & 5, & 6, & \dots & \frac{\mathbb{1}}{2}-3, & \frac{\mathbb{1}}{2}-2, & \frac{\mathbb{1}}{2}-1 \end{array} \quad (10)$$

$$\begin{array}{ccccccccc} 3, & 5, & 7, & 9, & 11, & 13, & \dots & \mathbb{1}-7, & \mathbb{1}-5, & \mathbb{1}-3 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \uparrow \\ 1, & 2, & 3, & 4, & 5, & 6, & \dots & \frac{\mathbb{1}}{2}-4, & \frac{\mathbb{1}}{2}-3, & \frac{\mathbb{1}}{2}-2 \end{array} \quad (11)$$

In order to become more familiar with natural and extended natural numbers, let us consider one more example where we multiply each element of the set of natural numbers,  $\mathbb{N}$ , by 2. We would like

to study the resulting set that is called  $\mathbb{E}^2$  herein-after to calculate the number of its elements and to specify which among its elements are natural and which ones are extended natural numbers and how many they are.

The introduction of the new numeral system allows us to write down the set,  $\mathbb{N}$ , of natural numbers in the form (Eq. 7). By definition, the number of elements of  $\mathbb{N}$  is equal to  $\mathbb{c}$ . Thus, after multiplication of each of the elements of  $\mathbb{N}$  by 2, the resulting set,  $\mathbb{E}^2$ , will also have grossone elements. In particular, the number  $\frac{\mathbb{1}}{2}$  multiplied by 2 gives us  $\mathbb{1}$  and  $\frac{\mathbb{1}}{2} + 1$  multiplied by 2 gives us  $\mathbb{1} + 2$  that is even extended natural number; see (Eq. 7). Analogously, the last element of  $\mathbb{N}$ , i.e.,  $\mathbb{c}$ , multiplied by 2 gives us  $2\mathbb{1}$ . Thus, the set of even numbers  $\mathbb{E}^2$  can be written as follows:

$$\begin{aligned} \mathbb{E}^2 = & \{2, 4, 6, \dots \mathbb{1}-4, \mathbb{1}-2, \mathbb{1}, \mathbb{1} \\ & + 2, \mathbb{1} + 4, \dots 2\mathbb{1}-4, 2\mathbb{1}-2, 2\mathbb{1}\}, \end{aligned}$$

where numbers

$\{2, 4, 6, \dots, \mathbb{1}-4, \mathbb{1}-2, \mathbb{1}\}$  are even and natural (they are  $\frac{\mathbb{1}}{2}$ ).

and numbers

$\{\mathbb{1} + 2, \mathbb{1} + 4, \dots 2\mathbb{1}-4, 2\mathbb{1}-2, 2\mathbb{1}\}$  are even and extended natural; they also are  $\frac{\mathbb{1}}{2}$ .

## Concluding Remarks

In this entry, infinite sets have been studied using a recently introduced positional numeral system with the infinite radix  $\mathbb{1}$ . It has been shown that in certain cases the new numerals allow one to obtain more precise results in dealing with infinite quantities in comparison to numeral systems traditionally used for this purpose.

The  $\mathbb{1}$ -based numeral system allows us to distinguish new infinite sets that were invisible using traditional instruments both within continuum and numerable sets. Thanks to the  $\mathbb{1}$ -based numerals, it becomes possible to calculate the exact number of elements of old (see Table 1) and new sets and to exhibit sets that were constructed as continuum but are indeed countable bridging so the gap between the two groups

of sets (see Sergeyev 2010b for a detailed discussion). Reminding our example with the microscope, we are able now to see instead of two dots (countable and continuum) many different dots.

In this entry, only one application where ①-based numerals are useful has been discussed: measuring infinite sets. More examples showing how these numerals can be successfully used can be found in the following publications: Euclidean and hyperbolic geometry (see Margenstern 2012, 2015), percolation (see Iudin et al. 2012, 2015; Vita et al. 2012), fractals (see Sergeyev 2007, 2009a, 2011c; Vita et al. 2012), infinite series and the Riemann zeta function (see Sergeyev 2009c, 2011b; Zhigljavsky 2012), the first Hilbert problem and lexicographic ordering (see Sergeyev 2010b, 2015b; Sergeyev and Garro 2010), Turing machines (see Sergeyev and Garro 2010, 2013), and cellular automata (see D'Alotto 2012, 2013, 2015).

In particular, numerical computations with infinities and infinitesimals expressed by ①-based numerals are discussed in the following papers: numerical differentiation, solutions of systems of linear equations, and optimization (see De Cosmis and De Leone 2012; Sergeyev 2009b, 2011a; Žilinskas 2012) and ordinary differential equations (see Amodio et al. 2016; Mazzia et al. 2016; Sergeyev 2011a, 2013c, 2015a; Sergeyev et al. 2016).

## Bibliography

- Amodio P, Iavernaro F, Mazzia F, Mukhametzhanov MS, Sergeyev YaD (2016) A generalized Taylor method of order three for the solution of initial value problems in standard and infinity floating-point arithmetic. *Math Comput Simul.* in press
- Butterworth B, Reeve R, Reynolds F, Lloyd D (2008) Numerical thought with and without words: evidence from indigenous Australian children. *Proc Natl Acad Sci U S A* 105(35):13179–13184
- Cantor G (1955) Contributions to the founding of the theory of trans nite numbers. Dover Publications, New York
- Cococcioni M, Pappalardo M, Sergeyev YaD (2016) Towards lexicographic multiobjective linear programming using grossone methodology. In: Sergeyev YaD, Kvasov DE, Dell' Accio F, Mukhametzhanov MS (eds) *Proceedings of the 2nd international conference “numerical computations: theory and algorithms”, vol 1776*. AIP Publishing, New York, p 090040
- Conway JH, Guy RK (1996) *The book of numbers*. Springer, New York
- D'Alotto L (2012) Cellular automata using infinite computations. *Appl Math Comput* 218(16):8077–8082
- D'Alotto L (2013) A classification of two-dimensional cellular automata using infinite computations. *Ind J Math* 55:143–158
- D'Alotto L (2015) A classification of one-dimensional cellular automata using infinite computations. *Appl Math Comput* 255:15–24
- De Cosmis S, De Leone R (2012) The use of grossone in mathematical programming and operations research. *Appl Math Comput* 218(16):8029–8038
- De Leone R (2017) Nonlinear programming and grossone: quadratic programming and the role of constraint qualifications. *Appl Math Comput.* in press
- Gödel K (1940) *The consistency of the continuum-hypothesis*. Princeton University Press, Princeton
- Gordon P (2004) Numerical cognition without words: evidence from Amazonia. *Science* 306:496–499
- Hardy GH (1910) *Orders of infinity*. Cambridge University Press, Cambridge
- Hilbert D (1902) Mathematical problems: lecture delivered before the international congress of mathematicians at Paris in 1900. *Bull Am Math Soc* 8:437–479
- Iudin DI, Sergeyev YaD, Hayakawa M (2012) Interpretation of percolation in terms of infinity computations. *Appl Math Comput* 218(16):8099–8111
- Iudin DI, Sergeyev YaD, Hayakawa M (2015) Infinity computations in cellular automaton forest-fire model. *Commun Nonlinear Sci Numer Simul* 20(3):861–870
- Leder GC (2015) Mathematics for all? The case for and against national testing. In: Cho SJ (ed) *The proceedings of the 12th international congress on mathematical education: intellectual and attitudinal challenges*. Springer, New York, pp 189–207
- Leibniz GW, Child JM (2005) *The early mathematical manuscripts of Leibniz*. Dover Publications, New York
- Levi-Civita T (1898) Sui numeri transfiniti. *Rend Acc Lincei Series 5a* 11:7–91
- Lolli G (2012) Infinitesimals and infinites in the history of mathematics: a brief survey. *Appl Math Comput* 218(16):7979–7988
- Lolli G (2015) Metamathematical investigations on the theory of grossone. *Appl Math Comput* 255:3–14
- Margenstern M (2011) Using grossone to count the number of elements of infinite sets and the connection with bijections. *p-Adic Numbers Ultrametric Anal Appl* 3(3):196–204
- Margenstern M (2012) An application of grossone to the study of a family of tilings of the hyperbolic plane. *Appl Math Comput* 218(16):8005–8018
- Margenstern M (2015) Fibonacci words, hyperbolic tilings and grossone. *Commun Nonlinear Sci Numer Simul* 21(1–3):3–11
- Mazzia F, Sergeyev YaD, Iavernaro F, Amodio P, Mukhametzhanov MS (2016) Numerical methods for solving ODEs on the Infinity Computer. In: Sergeyev

- YaD, Kvasov DE, Dell'Accio F, Mukhamethanov MS (eds) Proceedings of the 2nd international conference “numerical computations: theory and algorithms”, vol 1776. AIP Publishing, New York, p 090033
- Montagna F, Simi G, Sorbi A (2015) Taking the Pirahã seriously. *Commun Nonlinear Sci Numer Simul* 21(1–3):52–69
- Pica P, Lemer C, Izard V, Dehaene S (2004) Exact and approximate arithmetic in an amazonian indigene group. *Science* 306:499–503
- Rizza D (2016) Supertasks and numeral systems. In: Sergeyev YaD, Kvasov DE, Dell'Accio F, Mukhamethanov MS (eds) Proceedings of the 2nd international conference “numerical computations: theory and algorithms”, vol 1776. AIP Publishing, New York, p 090005
- Robinson A (1996) Non-standard analysis. Princeton University Press, Princeton
- Sergeyev YaD (2007) Blinking fractals and their quantitative analysis using infinite and infinitesimal numbers. *Chaos Solitons Fractals* 33(1):50–75
- Sergeyev YaD (2008) A new applied approach for executing computations with infinite and infinitesimal quantities. *Informatica* 19(4):567–596
- Sergeyev YaD (2009a) Evaluating the exact infinitesimal values of area of Sierpinski's carpet and volume of Menger's sponge. *Chaos Solitons Fractals* 42(5):3042–3046
- Sergeyev YaD (2009b) Numerical computations and mathematical modelling with infinite and infinitesimal numbers. *J Appl Math Comput* 29:177–195
- Sergeyev YaD (2009c) Numerical point of view on Calculus for functions assuming finite, infinite, and infinitesimal values over finite, infinite, and infinitesimal domains. *Nonlinear Anal Ser A Theory Methods Appl* 71(12):e1688–e1707
- Sergeyev YaD (2010a) Computer system for storing infinite, infinitesimal, and finite quantities and executing arithmetical operations with them. US Patent 7,860,914
- Sergeyev YaD (2010b) Counting systems and the first Hilbert problem. *Nonlinear Anal Ser A Theory Methods Appl* 72(3–4):1701–1708
- Sergeyev YaD (2010c) Lagrange lecture: methodology of numerical computations with infinities and infinitesimals. *Rendiconti del Seminario Matematico dell'Università e del Politecnico di Torino* 68(2): 95–113
- Sergeyev YaD (2011a) Higher order numerical differentiation on the Infinity Computer. *Optim Lett* 5(4): 575–585
- Sergeyev YaD (2011b) On accuracy of mathematical languages used to deal with the Riemann zeta function and the Dirichlet eta function. *p-Adic Numbers Ultrametric Anal Appl* 3(2):129–148
- Sergeyev YaD (2011c) Using blinking fractals for mathematical modelling of processes of growth in biological systems. *Informatica* 22(4):559–576
- Sergeyev YaD (2013a) Arithmetic of Infinity, Edizioni Orizzonti Meridionali, CS, 2003, 2nd edn
- Sergeyev YaD (2013b) Numerical computations with infinite and infinitesimal numbers: theory and applications. In: Sorokin A, Pardalos PM (eds) Dynamics of information systems: algorithmic approaches. Springer, New York, pp 1–66
- Sergeyev YaD (2013c) Solving ordinary differential equations by working with infinitesimals numerically on the Infinity Computer. *Appl Math Comput* 219(22): 10668–10681
- Sergeyev YaD (2015a) Numerical infinitesimals for solving ODEs given as a black-box. In: Simos TE, Tsitouras C (eds) AIP proceedings of the international conference on numerical analysis and applied mathematics 2014 (ICNAAM-2014), vol 1648. Melville, New York, p 150018
- Sergeyev YaD (2015b) The olympic medals ranks, lexicographic ordering, and numerical infinities. *Math Intell* 37(2):4–8
- Sergeyev YaD (2015c) Un semplice modo per trattare le grandezze infinite ed infinitesime. *Matematica Nella Società e Nella Cultura: Rivista Della Unione Matematica Italiana* 8(1):111–147
- Sergeyev YaD (2016) The exact (up to infinitesimals) infinite perimeter of the Koch snowflake and its finite area. *Commun Nonlinear Sci Numer Simul* 31(1–3):21–29
- Sergeyev YaD, Garro A (2010) Observability of Turing machines: a refinement of the theory of computation. *Informatica* 21(3):425–454
- Sergeyev YaD, Garro A (2013) Single-tape and multi-tape Turing machines through the lens of the Grossone methodology. *J Supercomput* 65(2):645–663
- Sergeyev YaD, Mukhamethanov MS, Mazzia F, Iavernaro F, Amadio P (2016) Numerical methods for solving initial value problems on the Infinity Computer. *Int J Unconv Comput* 12(1):3–23
- Vita MC, De Bartolo S, Fallico C, Veltri M (2012) Usage of infinitesimals in the Menger's sponge model of porosity. *Appl Math Comput* 218(16):8187–8196
- Wallis J (1656) *Arithmetica in nitorum*
- Zhilgjavsky AA (2012) Computing sums of conditionally convergent and divergent series using the concept of grossone. *Appl Math Comput* 218(16):8064–8076
- Žilinskas A (2012) On strong homogeneity of two global optimization algorithms based on statistical models of multimodal objective functions. *Appl Math Comput* 218(16):8131–8136



## Inductive Turing Machines

Mark Burgin

Department of Mathematics, UCLA, Los Angeles, CA, USA

### Article Outline

Glossary

Introduction/History

Basic Constructions, Theoretical Results, and Applications

Future Directions

Bibliography

### Glossary

**Algorithm** is a compressed exact description of some activity (functioning, behavior, or computation), which allows reproducing this activity (functioning, behavior, or computation).

**Complexity** is a measure of definite resources needed for activity (functioning, behavior, or computation).

**Computation** is goal-oriented information processing, results of which can be accessed (registered) by an observer (user).

**Information** for a system  $R$  is a capacity to change an infological system  $\text{IF}(R)$  of the system  $R$ .

**Computability** is a possibility to compute values of a function or elements of a set.

**Process** is a connected system, e.g., a sequence, of actions or events.

### Introduction/History

Inductive Turing machine, as a rigorous mathematical model of algorithms and computation, was introduced by Mark Burgin in 1983 in the form of an abstract computational device more

powerful than Turing machine (Burgin 1983). It was the first class of rigorously defined abstract automata with this property, i.e., the first super-recursive class of abstract automata. The initial goal was to build automata able to compute Kolmogorov complexity, an important characteristic of information incomputable by Turing machines (Kolmogorov 1965; Li and Vitanyi 1997). The first stage of the inductive Turing machine theory development essentially benefited from the advice given by the great Russian mathematician Andrey Nikolaevich Kolmogorov. In the process of the theory development, it was proved that inductive Turing machines have many other advantages in comparison with Turing machines and other recursive automata and algorithms (Burgin 2005a).

It is interesting that when inductive Turing machines were invented, the inventor did not know about limiting recursive and limiting partial recursive functions as well as about trial and error predicates.

Importance of inductive Turing machines as a new mathematical model of computation was also demonstrated in 1998 when Hintikka and Mutanen rediscovered the first level of the hierarchy of inductive Turing machines building trial-and-error machines, which are functionally and linguistically equivalent to inductive Turing machines of the first order (Hintikka and Mutanen 1998).

It is interesting that the history of inductive Turing machines is very similar to the history of Turing machines. Indeed, Turing machine was the first mathematical model of algorithms in the form of an abstract computational device (Turing 1936). However, before introduction of Turing machines by the outstanding British researcher Alan Turing, two other mathematical models – recursive functions (Gödel 1934) and partial recursive functions (Kleene 1936) as algorithms in the form of constructive functions, as well as  $\lambda$ -calculus as algorithms in the form of a logical calculus (Church 1932/33) – were built. However, those models did not provide a description of a computing device for realization of an algorithm.

As a result, they were not treated as persuasive models of algorithms and computation. Cockshott and Michaelson correctly write (Cockshott and Michaelson 2007):

Calculi are rules for the manipulation of strings of symbols and these rules will not do any calculations unless there is some material apparatus to interpret them. Leave a book on the  $\lambda$ -calculus on a shelf along with a sheet of paper containing a formula in the  $\lambda$ -calculus and nothing will happen. Bring along a mathematician, give them the book and the formula and, given enough paper and pencil the ensemble can compute.

It is interesting that mathematical models of algorithms in the form of functions and logical calculus were created by logicians Gödel, Church, and Kleene, while a model in the form of a computing device was elaborated by a mathematician Turing.

In a similar way, before introduction of inductive Turing machines, two other models – limiting recursive and limiting partial recursive functions having the form of constructive functions (Gold 1965) and trial and error predicates having the form of a logical structure (Putnam 1965) – were constructed. However, those models did not provide a description of a computing device for realization of an algorithm. Naturally, they were not treated as models of algorithms that go beyond the Church-Turing thesis. The first grounded demonstration that inductive Turing machines satisfy the most natural definitions of algorithms and thus, disprove the Church-Turing thesis being more powerful than Turing machines was in Burgin (1987).

It is possible to delineate three roots of the construction of inductive Turing machines:

- *Turing machines* served as a model for simple inductive Turing machines, as well as for the control device of all types of inductive Turing machines.
- *Inductive cognitive processes* influenced elaboration of the formalized mode of inductive computations.
- *Kolmogorov algorithms* (Kolmogorov 1953) inspired construction of the structured memory of inductive Turing machines.

After Turing machines were acknowledged as an absolute model of algorithms and of computation, different authors tried to build models of algorithms more powerful than Turing machines, and for many decades, they were not able to do this. This situation influenced the emergence and popularity of the famous *Church-Turing thesis*, which had different forms and could be more appropriately called the *Church-Turing Conjecture*. Here we consider a Turing's version that states that *the informal notion of algorithm is equivalent to the concept of a Turing machine* and a Church's version that asserts that *any computable function is a partial recursive function*. In other words, an algorithmic form of the Church-Turing thesis states:

Any problem that can be solved by an algorithm can be solved by some Turing machine and any algorithmic computation can be done by some Turing machine.

This thesis was used to prove many undecidability and incomputability results. Mathematicians and computer scientists treated these results as absolute unchangeable achievements. Discovery of inductive Turing machines and other superrecursive algorithms demonstrated relativity of those results making them less important. Besides, computational complexity theory and algorithmic information theory are still mostly based on Turing machines, while acceptance of more powerful algorithms can threaten the existing bias. That is why recognition and understanding of algorithms and their models, which go beyond the Church-Turing thesis such as inductive Turing machines, have been a long and hard process.

## **Basic Constructions, Theoretical Results, and Applications**

### **Structure and Functioning of Inductive Turing Machines**

Any inductive Turing machines have three components, which exactly replicate the structure of a contemporary computer. Namely, an inductive

Turing machine  $M$  has hardware, software, and infware.

The *hardware* of an inductive Turing machine  $M$  consists of three abstract devices:

- The *control device (controller)*  $A$ , which is a finite automaton and controls performance of  $M$
- The *operating device (processor)*  $H$ , which includes three processing units
- The *memory (data and information storage)*  $E$

The *software* of an inductive Turing machine  $M$  consists of rules according to which the operating device  $H$  functions.

The *infware* of an inductive Turing machine  $M$  consists of those symbols, words and languages, which are processes by  $M$ .

In the constructive hierarchy of inductive Turing machines, the simplest devices are simple inductive Turing machines.

The *memory*  $E$  of a simple inductive Turing machine is exactly the same as the memory of a Turing machine with three tapes: one of them is the input tape, another is the working tape, and the third is the output tape. In contrast to Turing machines, which are usually considered with one tape, this structure properly models the structure of computers representing not only the processing device but also input and output devices of a computer.

The *operating device (processor)*  $H$  of a simple inductive Turing machine consists of three heads, which are similar to those that are used in Turing machines, and each head works in its separate tape. The head  $h_1$  works in the input tape, the head  $h_2$  works in the working tape, and the head  $h_3$  works in the output tape.

The *software*  $R$  of a simple inductive Turing machine  $M$  is also a program that consists of three groups (each for a corresponding head) of simple rules:

$$q_h a_i \rightarrow a_j q_k X \quad (1)$$

Here  $q_h$  and  $q_k$  are states of  $A$ ,  $a_i$  and  $a_j$  are symbols of the alphabet of  $M$ , and  $X$  is one of the following symbols: R, L, and S. Each group of

rules determines functioning of one head of  $H$ . The rule (1) means that if the state of the control device  $A$  of  $M$  is  $q_h$  and the corresponding head  $h_t$  ( $t = 1, 2, 3$ ) of the processor  $H$  observes the symbol  $a_i$  in the cell where this head is situated, then the state of  $A$  becomes  $q_k$ , while the head  $h_t$  writes the symbol  $a_j$  in the cell where it is situated and moves right to the next cell from its tape when  $X = R$ , moves left to the next cell from its tape when  $X = L$ , and stays in the same cell when  $X = S$ . Each rule directs one step of computation of the simple inductive Turing machine  $M$ . Rules of the inductive Turing machine  $M$  define the transition function of  $M$  and describe changes of  $A$ ,  $H$ , and  $E$ . Consequently, these rules also determine the transition functions of  $A$ ,  $H$ , and  $E$ .

We see that the rules according to which inductive Turing machines function are the same as the rules of Turing machines.

The machine  $M$  functions in the following way. At the beginning, the head  $h_1$  of the processor  $H$  works in agreement with the head  $h_2$  rewriting the input word from the input tape into the working tape. Then the head  $h_2$  starts working independently. It observes some cell with a symbol  $a_i$  (it may be the symbol  $\Lambda$ , which denotes an empty cell), and the control device  $A$  is in some state  $q_h$ . Then the control device  $A$  (and/or the head  $h_2$ ) chooses from the system  $R$  of rules a rule  $r$  with the left part equal to  $q_h a_i$  and performs the operation prescribed by this rule. In some cases, head  $h_3$  writes some symbol in the output tape. If there is no rule in  $R$  with such a left part, the machine  $M$  stops functioning. If there are several rules with the same left part,  $M$  works as a nondeterministic Turing machine, performing all possible operations. When  $A$  comes to one of the final states from  $F$ , the machine  $M$  also stops functioning. In all other cases, it continues operation without stopping.

Thus, on each step, operation of a simple inductive Turing machine is the same as operation of some Turing machine. The difference is in the definition of the result. The result of a Turing machine is achieved when it comes to a final state. Usually, obtaining the result, the Turing machine halts because for a Turing machine, it is

unreasonable to continue to work after this. A simple inductive Turing machine works in a different way. In some cases, it stops without the result. In other cases, it gives the result coming to a final state just as a Turing machine. However, there are situations, when a simple inductive Turing machine does not halt but gives the result, which is defined by the behavior of the output tape. Namely, when the word in the output tape stops changing, this word becomes the result of the simple inductive Turing machine. This is the *output-stabilizing mode* of computation.

Note that similar to those algorithms that existed before creation of inductive Turing machines, the new algorithms – inductive Turing machines – give their results in finite time, i.e., after making a finite number of steps in all cases when the result is obtained. At the same time, inductive Turing machines realize hyper-computation in a general case (Syropoulos 2008).

There are also other ways to determine results of inductive computations (Burgin 2014a, 2015b). For instance, in the *state-stabilizing mode* of computation, an inductive Turing machine  $M$  gives a *result by state stabilizing* if after some number of steps, the state of the machine  $M$  always remains in the same final group of states.

In the *bistabilizing mode* of computation, an inductive Turing machine  $M$  gives a *result by bistabilizing* if after some number of steps the output of the machine  $M$  stops changing, while the state of  $M$  remains in the same final group of states.

It is proved that all three modes of computation have the same computing and decision power, i.e., they are linguistically and functionally equivalent (Burgin 2014a, 2015b).

Simple inductive Turing machines are situated at the bottom of the hierarchy of all inductive Turing machines being the simplest species in this family. Inductive Turing machines of the first order form the first level of this hierarchy and include simple inductive Turing machines.

Any inductive Turing machine  $M$  has hardware, software, and infware.

The *hardware* of an inductive Turing machine  $M$  consists of three abstract devices:

- The *control device (controller)*  $C$
- The *operating device (processor)*  $H$ , which can include any number of processing units
- The *memory*  $E$

The *control device*  $C$  of all inductive Turing machines is a finite automaton. It controls and regulates processes and parameters of the inductive Turing machine  $M$ : the state of the whole machine  $M$ , the processing of information by the processor  $H$ , and the storage of information in the memory  $E$ . In essence, it is similar to the control device of a Turing machine.

The *software* of an inductive Turing machine  $M$  consists of rules according to which the operating device  $H$  functions.

The *infware* of an inductive Turing machine  $M$  consists of those symbols, words and languages, which are processes by  $M$ .

In the expanded hierarchy of inductive Turing machines, the simplest devices are simple inductive Turing machines.

The *operating device (processor)*  $H$  performs information processing in  $M$  and consists of one or several processing units similar to heads of a multihead Turing machine and to unit processors of a computer. However, in comparison with computers, each processing unit  $P$  from  $H$  performs very simple operations. It can change a symbol in the cell, which it observes, and go from this cell to another one using connections between cells in the memory  $E$  of  $M$ . This allows one to model various real and abstract computing systems: multiprocessor computers; Turing machines with several tapes; networks, grids, and clusters of computers; cellular automata; neural networks; and systolic arrays.

The *software*  $R$  of an inductive Turing machine  $M$  of the first and higher orders is also a program that consists of groups of simple rules:

$$q_h a_i \rightarrow a_j q_k c \quad (2)$$

Here  $q_h$  and  $q_k$  are states of  $A$ ,  $a_i$  and  $a_j$  are symbols of the alphabet of  $M$ , and  $c$  is a type of connection in the memory  $E$ . Each group of rules determines functioning of one processing unit of  $H$ . The rule (2) means that if the state of the

control device  $A$  of  $M$  is  $q_h$  and the corresponding processing unit  $P$  observes the symbol  $a_i$  in its cell, then the state of  $A$  becomes  $q_k$ , while the processing unit  $P$  writes the symbol  $a_j$  in the cell where it is situated and moves to the next cell by a connection (tie) of the type  $c$ . Each rule directs one step of computation of the inductive Turing machine  $M$ . Rules of the inductive Turing machine  $M$  define the transition function of  $M$  and describe changes of  $A$ ,  $H$ , and  $E$ . Consequently, these rules also determine the transition functions of  $A$ ,  $H$ , and  $E$ .

These rules cover only synchronous parallelism of computation. However, it is possible to consider inductive Turing machines of any order in which their processor can perform computations in the concurrent mode.

The main difference between simple inductive Turing machines and inductive Turing machines of the first and higher orders is in the organization of the memory.

The *memory*  $E$  of inductive Turing machines of the first and higher orders consists of cells and is structured by a system of relations and ties that organize memory functioning and provide connections between cells. This structuring delineates three components of the *memory*  $E$ : input registers, the working memory, and output registers. In a general case, cells may be of different types: *binary cells* store bits of information represented by symbols 1 and 0; *byte cells* store information represented by strings of eight binary digits; *symbol cells* store symbols of the alphabet(s) of the machine  $M$  and so on. Thus, the memory is a network of cells, and it is possible to interpret these cells not only as containers of symbols but also as information processing systems, such as neurons in the brain of a human being or computers in the World Wide Web or abstract computing devices in a grid automaton (Burgin 2003b).

In contrast to simple inductive Turing machines, inductive Turing machines of the first and higher orders utilize agent technology, which became popular in the 1990s (Bradshaw 1997; Dinverno and Luc 2001). Namely, an inductive Turing machine  $M$  of the first or higher orders contains another automaton  $A$  (the agent), which

builds the memory of  $M$ . When  $M$  has the first order, the automaton  $A$  is a Turing machine. When  $M$  has the second or higher order, the automaton  $A$  is an inductive Turing machine.

In inductive Turing machines of the first and higher orders, there are agents of three types:

- *Connection agents* build connection (ties or channels) between cells in the memory.
- *Cell agents* build cells in the memory.
- *Unified agents* build cells and connection (ties or channels) between them.

Historical analysis shows that agents emerged in the theory of computation and algorithms in the 1980s, while according to the opinions of experts, agent technology in software industry started only in the 1990s although John Doyle (1983) and Marvin Minsky (1986) used the word “agent” in an informal setting. The notable physicist Freeman Dyson (1972) wrote about missed opportunities in physics and mathematics. The example of inductive Turing machines shows how opportunities provided by the theory of computation and algorithms were missed by computer technology.

Many assume that technology always goes ahead of theory. However, there are colorful examples when theory was far ahead of technology. For instance, Maxwell’s equations had been elaborated before radio waves, one of the cornerstones of contemporary technology, were discovered and utilized. Universal Turing machines had been theoretically constructed before the first electronic computers were built. In a similar way, inductive Turing machines had been theoretically created before agent technology came to software and network technology.

There are different techniques to organize memory construction with the help of computing/constructing agents. The simplest approach called the *sequential strategy* assumes that given some data, e.g., a description of the structure of the memory  $E$  of an inductive Turing machine  $M$ , an automaton  $A$  builds all connections in the memory  $E$  before the machine  $M$  starts its computation. If  $M$  is an inductive Turing machine of the first order,  $A$  is a Turing machine. If  $M$  is an inductive Turing machine of the second or higher

order,  $A$  is an inductive Turing machine, which has order less than  $M$ . For instance, the memory of inductive Turing machines of the second order is constructed by Turing machines of the first order.

According to another methodology called the *concurrent strategy*, memory construction by the machine  $A$  and computations of the machine  $M$  go concurrently, while the machine  $M$  computes, the machine  $A$  constructs connections in the memory  $E$ .

It is also possible to use the *mixed strategy* when some connections in the memory  $E$  are assembled before the machine  $M$  starts its computation, while other connections are formed parallel to the computing process of the machine  $M$ .

These three strategies determine three kinds of the constructed memory:

In the *static memory*  $E$  of the machine  $M$ , all connections are constructed before  $M$  starts working.

In the *growing memory*  $E$  of the machine  $M$ , connections are constructed while  $M$  is working but no connections are deleted.

In the *dynamic memory*  $E$  of the machine  $M$ , when it necessary, some connections are constructed, and when it necessary, some connections are deleted while  $M$  is working.

In addition, there are three types of construction agents  $A$ :

- A *rigid agent*  $A$  always constructs one and the same memory structure for the machine  $M$ .
- An *adaptive agent*  $A$  constructs memory structure for the machine  $M$  taking into account the input data.
- A *dynamic agent*  $A$  constructs memory structure for the machine  $M$  taking into account the process of computation.

Working in the inductive Turing machine  $M$ , its dynamic agent  $A$  can carry out reconfiguration of the system structure and architecture creating order from chaos. This is especially important when the machine  $M$  with the memory  $E$  represents a global computer network such as the existing World Wide Web or the future Grid.

In addition, there is a memory assembly method when the agent machine  $A$  is separate

from the basic machine  $M$  but is controlled by  $M$ . According to another memory assembly technique, the basic machine  $M$  contains the agent machine  $A$  as its part. One more assembly technique uses an agent machine  $A$  completely independent of the basic machine  $M$ . These situations give us three types of agents in inductive Turing machines

- A *componential agent*  $A$  is a part of the basic machine  $M$ .
- A *controlled agent*  $A$  functions under the control of  $M$  when the control device  $C$  of  $M$  is either combined with the control device of  $A$  or regulates functioning of the control device of  $A$ .
- An *autonomous agent*  $A$  works independently of  $M$  although it may use the same data or some information coming from  $M$ .

It is necessary to remark that when both the agent  $A$  and the main machine  $M$  are Turing machines or other kinds of recursive automata, such a recursive agent  $A$  does not increase the computing power of the main machine  $M$ . However, it is proved that  $A$  can essentially amplify the efficiency of the machine  $M$  (Burgin 1999).

At the same time, utilization of inductive agents, e.g., when the agent  $A$  is an inductive Turing machine, can to a great extent increase the computing power of the machine  $M$  (Burgin 2003a).

Application of agents allows building the *constructive hierarchy* of inductive Turing machines in a formal way. On the first level of the constructive hierarchy, we have *inductive Turing machines with recursive memory*, which are also called *inductive Turing machines of the first order*.

To build the constructive hierarchy of inductive Turing machines, we use Turing machines and inductive Turing machines as agents for memory construction.

The memory  $E$  is called *recursive* if all relations that define its structure are recursive meaning that there are Turing machines or other recursive automata that establish the structured memory (Burgin 2004, 2005a, 2016c).

The memory  $E$  is called *n-inductive* if its structure is constructed by an inductive Turing

machine of the order  $n$ . The second inductive Turing machine plays the role of an agent for the first inductive Turing machine.

Inductive Turing machines with  $n$ -inductive memory are called *inductive Turing machines of the order  $n + 1$* . Namely, in inductive Turing machines of order  $n$ , the memory is constructed by Turing machines of order  $n - 1$ . For instance, in inductive Turing machines of the second order, the memory is constructed by Turing machines of the first order.

We denote the class of all inductive Turing machines of the order  $n$  by  $\mathbf{IT}_n$  and obtain the *constructive hierarchy*  $\mathbf{IT} = \bigcup_{n=1}^{\infty} \mathbf{IT}_n$  of inductive Turing machines, in which  $\mathbf{T}$  is the class of all Turing machines and

$$\mathbf{T} \subset \mathbf{IT}_1 \subset \mathbf{IT}_2 \subset \mathbf{IT}_3 \subset \dots \subset \mathbf{IT}_n \subset \dots$$

Thus, if the memory of an inductive Turing machine  $M$  is constructed by an inductive Turing machine  $A$ , then  $A$  is an agent that performs memory structuring for  $M$ . Such a superrecursive agent  $A$  can essentially advance the power of the machine  $M$  because the machine  $M$  has higher order than the machine  $A$  (Burgin 2003a).

Note that the hierarchy  $\mathbf{IT}$  of inductive Turing machines induces a hierarchy of agents that function in  $\mathbf{IT}$ .

Moreover, it is possible to delegate the task of memory building to two or more agents in the form of Turing machines or inductive Turing machines. For instance, given initial data, one of these machines AC computes (designs) a description (schema) of the memory  $E$ , while another machine AB constructs connections between cells in  $E$  using the schema designed by the machines AC. This approach involves two types of agents:

- *Designing agents*
- *Constructing agents*

In a more advanced memory construction schema, memory  $E$  of an inductive Turing machine  $M$  is constructed by three agents in the form of Turing machines or inductive Turing

machines. For instance, the first agent AI collects information, e.g., by data mining, for the schema of the memory  $E$ , the second agent AC computes a description (schema) of the memory  $E$ , and the third agent AB constructs connections between cells in  $E$ . This approach involves three types of agents:

- *Mining agents*
- *Designing agents*
- *Constructing agents*

We see that the schema of inductive Turing machines, which is intrinsically based on agent technology, introduced different types of computational agents before agent approach came to technology.

Other types of inductive Turing machines were suggested by Jürgen Schmidhuber in 2000 and Arto Mutanen in 2004. Schmidhuber called his construction a *general Turing machine* (Schmidhuber 2002), and Mutanen called his construction a *generalized Turing machine* (Mutanen 2004). However, the term *inductive Turing machine* better reflects the essence of automata that model inductive processes.

Existence of different types of inductive Turing machines allows building the following classification:

- Inductive Turing machines with local output stabilization
- Inductive Turing machines with global output stabilization
- Inductive Turing machines with state stabilization
- Inductive Turing machines with table stabilization
- Agent inductive Turing machines
- Table inductive Turing machines

### Theoretical Results: Computational Power, Efficiency, and Other Properties of Inductive Turing Machines

Provision of a possibility to obtain results without halting, makes simple inductive Turing machines more powerful than Turing machines; simple

inductive Turing machines can do everything Turing machines can and even more. For instance, a simple inductive Turing machine can solve the halting problem for all Turing machines (Burgin 2001b), while there is no Turing machine that can do this (Turing 1936). One more function, which cannot be computed by Turing machines (Li and Vitanyi 1997), but is naturally computed by simple inductive Turing machines, is Kolmogorov complexity (Burgin 1983). All this shows that functioning of inductive Turing machines is supercomputation (Syropoulos 2008).

It is proved that simple inductive Turing machines can compute limiting recursive, limiting partial recursive functions (Gold 1965), and trial and error predicates (Putnam 1965). Inductive Turing machines of higher orders can compute iterated limiting recursion (Schubert 1974; Criscuolo et al. 1975).

Inductive Turing machines of the first order have the same computing and decision power as simple inductive Turing machines, but inductive Turing machines of higher orders have much more computing and decision power. In particular, it is proved that inductive Turing machines from the constructive hierarchy described in the previous section can compute and decide the whole arithmetical hierarchy (Burgin 1988, 2003a), while Turing machines can decide only the lowest level and compute only two lowest levels of the arithmetical hierarchy (Rogers 1987).

In addition, inductive Turing machines can prove or disprove, i.e., decide, consistency of an axiomatizable first-order theory, as well as truthfulness of any predicate in such a theory (Mutanen 2004).

However, this growth in power brings one difficulty for users – inductive Turing machines do not always inform the user when the result is obtained. It is important to understand that this does not mean that the user will never know this result. It simply means that the user needs additional information to obtain this knowledge. That is why simple inductive Turing machines may be called machines (automata or algorithms) for intelligent users.

One more advantage of inductive Turing machines is their ability to decrease complexity

of different systems. Namely, Burgin proved algorithmic complexity becomes much smaller if inductive Turing machines are used instead of Turing machines (Burgin 2004). In particular, for infinitely many objects, inductive complexity is much smaller than their Kolmogorov complexity.

However, it is necessary to remark that although simple inductive Turing machines can solve much more problems than Turing machines, there are still algorithmic problems that simple inductive Turing machines cannot solve. For instance, there is no simple inductive Turing machine that can decide (determine) whether an arbitrary simple inductive Turing machine gives the result, i.e., is defined, for a given input or not (Burgin 2005b). This problem is called the definability problem, and the famous Halting problem is the definability problem for Turing machines.

At the same time, inductive Turing machines of the second order can solve this definability problem not only for simple inductive Turing machines but also for all inductive Turing machines of the first order. In general, we have the following result (Burgin 2005b):

Inductive Turing machines of order  $n$  can solve the Definability Problem for inductive Turing machines of order  $n - 1$  but cannot solve the Definability Problem for inductive Turing machines of order  $n$ .

Important theoretical results for simple inductive Turing machines in the form of table inductive Turing machines were obtained by Mutanen, who proved the s-m-n theorem and the recursion theorem for inductively computable functions (Mutanen 2004).

Borodyanskiy and Burgin obtained axiomatic characterization of inductive Turing machines in the class of transrecursive operators and Turing machines in the class of inductive Turing machines (Burgin and Borodyanskiy 1991, 1993b; Borodyanskiy and Burgin 1994).

In the theory of superrecursive algorithms, it is demonstrated that inductive Turing machines have the following advantages.

First, inductive Turing machines, as an innovative model of computations, algorithms, and information processing systems, can compute and

decide much more than Turing machines, i.e., they are superrecursive algorithms, while Turing machines are only recursive algorithms. In particular, inductive Turing machines can solve problems unmanageable by Turing machines providing means for decreasing complexity of computations and decision-making (Burgin 2005a). Consequently, in comparison with Turing machines and other recursive algorithms, such as partial recursive functions or Minsky machines, inductive Turing machines represent the next step in the development of computer science as well as in the advancement of network and computational technology.

Second, inductive Turing machines can essentially decrease complexity of various systems making easier their exploration, modeling, and construction (Burgin 2004, 2005, 2016c).

Third, inductive Turing machines also provide efficient tools for algorithmic information theory, which is one of the indispensable areas in information theory and is based on complexity of algorithms and automata (Chaitin 1977; Burgin 2010a, 2016e).

Before creation of inductive Turing machines refuted the Church-Turing Thesis, the system of all classes of algorithm, the Algorithmic Universe, was closed as it was bounded by the class of Turing machine, which was the most powerful in this universe. Turing machine. Creation of inductive Turing machines made the algorithmic universe open (Burgin and Dodig-Crnkovic 2013) due to the infinite hierarchy of inductive Turing machines and later construction of even more powerful classes of algorithms such as limit Turing machines (Burgin 1992).

### Applications of Inductive Turing Machines

Inductive Turing machines supply models of algorithms, networks, and information processing systems, which are more adequate than recursive algorithms and automata models of computations. As a result, inductive Turing machines have found diverse applications in many areas such as algorithmic information theory and complexity studies

(Burgin 2004, 2007, 2010b, 2016b, 2016e), software testing (Burgin and Debnath 2009; Burgin et al. 2009), high-performance computing (Burgin 1999), evolutionary information theory (Burgin 2013), logic (Mutanen 2004), machine learning (Burgin and Klinger 2004b; Mutanen 2004), artificial intelligence (Borodianskiy and Burgin 1994; Burgin 2016b), sociology (Burgin 1993; Burgin and Borodianskiy 1993a), software engineering (Burgin and Debnath 2004, 2005), computer networks (Burgin 2006; Burgin and Gupta 2012), evolutionary computations (Burgin and Eberbach 2009, 2009a, 2012), and in the study of mathematical problem complexity (Calude et al. 2012; Hertel 2012; Burgin et al. 2013). Inductive Turing machines can perform all types of machine learning – TxtEx-learning, TxtFin-learning, TxtBC-learning, and TxtEx\*-learning with better outcomes than the standard approach to these educational technologies (Beros 2013). While the traditional approach to machine learning models learning processes using functions, e.g., limiting partial recursive functions (Gold 1967), inductive Turing machines are automata, which can compute values of the modeling functions and perform other useful operations, while learning models based on functions only describe such operations. Application of inductive Turing machines to computer modeling and simulation is discussed in Burgin (2001a), Ades and Burgin (2007), and Burgin and Ades (2009).

Inductive Turing machines provide better possibilities for modeling biological computers than conventional models of computation. For instance, Adamatzky writes the plasmodium never stops modifying its protein network and thus, results obtained by plasmodium computers are akin to an inductive Turing machine results as it also does not stop to produce the result of computation (Adamatzky 2010). Roglic demonstrates that inductive Turing machine will reflect natural biological system functioning allowing to develop new kinds of computational evolutionary systems (Roglic 2007, 2011).

Agent technology developed for inductive Turing machines provides for further development of network architecture serving as a theoretical foundation for the innovative distributed

intelligent managed element (DIME) network architecture (DNA), which represents a phase transition in the development of global networks, their software, and functioning (Mikkilineni et al. 2012; Burgin and Mikkilineni 2014; Burgin et al. 2016). Working in a network treated as an extremely large dynamic memory, the DNA agent organizes connections in this network in the same way as one inductive Turing machine builds connections in the memory of another inductive Turing machine.

Various applications of inductive Turing machines, which are described above, are in contrast with opinions of some researchers, who expressed doubts about possibility of practical utilization of inductive Turing machines. To explain their delusion, we remind that inductive Turing machines can do everything what Turing machines can do. They can do more, but in many cases, inductive Turing machines do not inform the user that they already got the necessary result. However, even in these situations, an intelligent user can obtain the result observing the behavior of the machine. There are several approaches to this problem.

The first approach suggests using additional information and/or knowledge, e.g., background knowledge in the sense of Mutanen (2004), which the user has about the problem being solved by the machine, about the process of solution, and/or about the situation, in which the process goes.

The second approach is based on probabilistic techniques and consideration, which are used in the practice of randomized algorithms (Hromkovic 2005; Burgin 2006b). For instance, a Monte Carlo simulation algorithm cannot be modeled by a Turing machine or any other recursive algorithm but allows relevant representation by inductive Turing machines (Burgin and Ades 2009).

The third approach recommends treating inductive Turing machines and inductive computations as anytime algorithms, which are able to return a partial answer as an approximation of the exact answer (Boddy and Dean 1989; Zilberstein 1996). For instance, Manin (2012) suggested applying this technique for solving the Halting Problem for Turing machines.

## Future Directions

To understand the progress in computer science and mathematics achieved by introduction of inductive Turing machines, let us consider processes modeled by Turing machines and processes modeled by inductive Turing machines.

In his pioneering paper published in 1936, Turing clearly explains that his a-machine, later called Turing machine, mathematically models the work of a human computer. We can also add that it models the work of an accountant. In both cases, the goal of the process is computation of values of functions according to exact (mechanical) rules and stopping when the result is obtained:

The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer. The human computer is supposed to be following fixed rules; he has no authority to deviate from them in any detail. We may suppose that these rules are supplied in a book, which is altered whenever he is put on to a new job. He has also an unlimited supply of paper on which he does his calculations. He may also do his multiplications and additions on a ‘desk machine’, but this is not important (Turing 1950).

The work of a scientist or mathematician is essentially different because it is exploration, which can include calculation but cannot be reduced to it. Indeed, in science, e.g., in physics or biology, we can observe the following process of research:

First, scientists learn something about results of other scientists in their area.

Second (this step is sometimes skipped), scientists conduct some experiments and collect experimental data.

Third, scientists elaborate a hypothesis  $L$ .

Fourth, scientists conduct new experiments and check the hypothesis  $L$ .

Fifth, if a sufficient number of experiments support their hypothesis, scientists call  $L$  a law of nature.

As the time goes, the following situations are possible:

- (a) Whenever all further experiments related to the law  $L$  support  $L$ , this law  $L$  is accepted forever as a law of nature.
- (b) A new experiment contradicts  $L$ . In this case, either it is declared that  $L$  is not a law of nature or it is assumed that  $L$  is not valid in the initial domain. In both cases,  $L$  is rejected as a law of nature (either completely or for the initial domain) and scientists start searching for the new law, which more correctly, than  $L$  describes the experimental data.

We can see that this process exactly reflects how an inductive Turing machine is functioning. Note that if inductive Turing machines obtain their results, they do this in finite time, i.e., making only a finite number of steps. They mathematically describe and formalize inductive reasoning prevalent in science and mathematics. Note that induction used in science is not mathematical induction, which reduces induction to deduction.

Thus, we come to the following conclusion.

Turing machine formalizes the work of an accountant or a human computer.

Inductive Turing machine formalizes the work of a scientist and functioning of science.

Hence, it is natural that inductive Turing machines can do much more than Turing machines and this feature of inductive Turing machines is mathematically proved (Burgin 2003a, 2005a).

As a supportive evidence (not a proof) for the above statement, it is possible to take what Gödel wrote in a note entitled “A philosophical error in Turing’s work”:

Turing gives an argument which is supposed to show that mental procedures cannot go beyond mechanical procedures. However, this argument is inconclusive. What Turing disregards completely is the fact that mind, in its use, is not static, but constantly developing, i.e., we understand abstract terms more and more precisely as we go on using them...though at each stage the number and precision of the abstract terms at our disposal may be finite, both...may converge toward infinity.... (Gödel 1990)

Implementing both induction and deduction, inductive Turing machines provide more efficient tools for artificial intelligence (AI) in comparison with Turing machines or other recursive

algorithms, which are not adequate tools for AI (Sloman 2002). Thus, the principal problem is application of inductive Turing machines to solving those problems where conventional algorithms do not give satisfactory results or do not give any results.

Mutanen introduced inductive (generalized) Turing machines with oracles (Mutanen 2004). In the theory of traditional, i.e., recursive, computability, Turing machines with oracles form the base for relative recursive computability (Feferman 1992). Soare (2015) even argues that actually relative computations are in the center and form the main content of theoretical computer science. Thus, one more direction for future research is the development of relative inductive computability.

Inductive computability, recognizability, and decidability for sets and functions are studied for simple inductive Turing machines in (Burgin 2016a). It would be interesting to study these problems for inductive Turing machines of the second and higher orders.

There are other classes of abstract automata that also perform inductive computations: inductive cellular automata (Burgin 2015a), inductive evolutionary machines (Burgin and Eberbach 2009, 2009a, 2012), and periodic Turing machines (Burgin 2014b). An important and interesting problem is the further development of the theory of these classes of abstract automata.

## Bibliography

- Adamatzky A (2010) Physarum machines: computers from slime mould. World Scientific Publishing Co., Hackensack
- Ades MJ, Burgin MS (2007) Evolutionary processes in modeling and simulation. In: Proceedings of the 2007 Spring Simulation MultiConference (SpringSim '07), March 25–29, 2007, Norfolk, pp 133–138
- Barzdin JM, Freivald RV (1972) On the prediction of general recursive functions. Soviet Math Dokl 13:1224–1228
- Beros AA (2013) Learning theory in the arithmetical hierarchy, Preprint in mathematics, math.LO/1302.7069, 2013 (electronic edition: <http://arXiv.org/>)
- Blum L, Blum M (1975) Toward a mathematical theory of inductive inference. Inf Control 28:125–155

- Boddy M, Dean T (1989) Solving time-dependent planning problems. Technical Report: CS-89-03, Brown University
- Borodyanskiy YM, Burgin M (1994) Problems of artificial intelligence and trans-recursive operators, Visnik of the National Academy of Sciences of Ukraine, No. 11–12, pp 29–34 (in Ukrainian)
- Bradshaw J (ed) (1997) Software agents. AAAI Press/MIT Press, Menlo Park
- Burgin M (1983) Inductive turing machines. Not Acad Sci USSR 270(6):1289–1293. translated from Russian, v. 27, No. 3
- Burgin M (1984) Inductive Turing machines with multiple heads and Kolmogorov algorithms. Not Acad Sci USSR 275(2):280–284. (translated from Russian)
- Burgin M (1987) The notion of algorithm and the Church-Turing thesis. VIII International Congress on logic, methodology and philosophy of science, Moscow, vol 5, pt. 1, pp 138–140
- Burgin M (1988) Arithmetic hierarchy and inductive Turing machines. Not Acad Sci USSR 299(3):390–393. (translated from Russian)
- Burgin M (1992) Universal limit Turing machines. Not Russian Acad Sci 325(4):654–658. (translated from Russian)
- Burgin M (1993) Procedures of sociological measurements. In: Catastrophe, chaos, and self-organization in social systems. University of Koblenz-Landau, Koblenz, pp 125–129
- Burgin M (1999) Super-recursive algorithms as a tool for high performance computing. In: Proceedings of the high performance computing symposium, San Diego, pp 224–228
- Burgin M (2000) Theory of super-recursive algorithms as a source of a new paradigm for computer simulation. In: Proceedings of the business and industry simulation symposium, Washington, DC, pp 70–75
- Burgin M (2001a) Mathematical models for computer simulation. In: Proceedings of the business and industry simulation symposium, SCS, Seattle, pp 111–118
- Burgin M (2001b) How we know what technology can do. Commun ACM 44(11):82–88
- Burgin M (2001c) Topological algorithms. In: Proceedings of the ISCA 16th international conference “Computers and their applications”, ISCA, Seattle, pp 61–64
- Burgin M (2003a) Nonlinear phenomena in spaces of algorithms. Int J Comput Math 80(12):1449–1476
- Burgin M (2003b) Cluster computers and grid automata. In: Proceedings of the ISCA 17th international conference “Computers and their applications”, International Society for Computers and their Applications, Honolulu, pp 106–109
- Burgin M (2004) Algorithmic complexity of recursive and inductive algorithms. Theor Comput Sci 317(1/3):31–60
- Burgin M (2005a) Super-recursive algorithms. Springer, New York
- Burgin M (2005b) Superrecursive hierarchies of algorithmic problems. In: Proceedings of the 2005 international conference on foundations of computer science, CSREA Press, Las Vegas, pp 31–37
- Burgin M (2006a) Algorithmic control in concurrent computations. In: Proceedings of the 2006 international conference on foundations of computer science, CSREA Press, Las Vegas, pp 17–23
- Burgin M (2006b) Book Review: Juraj Hromkovic “Design and analysis of randomized algorithms: introduction to design paradigms”, Series: Texts in theoretical computer science, EATCS Series, Springer, (2005), Comput J 49(2): 249–250
- Burgin M (2007) Algorithmic complexity as a criterion of unsolvability. Theor Comput Sci 383(2/3):244–259
- Burgin M (2010a) Theory of information: Fundamentality, diversity and unification, World Scientific, New York/London/Singapore
- Burgin M (2010b) Algorithmic complexity of computational problems. Int J Comput Inform Technol 2(1):149–187
- Burgin M (2011) Super-recursive Kolmogorov Komplexity, Glossarium – BITri, (electronic edition: <http://glossarium.bitrum.unileon.es>)
- Burgin M (2013) Evolutionary information theory, information, vol 4, No.2, pp. 224–268
- Burgin M (2014a) Functioning of inductive Turing machines. Int J Unconv Comput (IJUC) 10(1–2):19–35
- Burgin M (2014b) Periodic Turing machines. J Comput Technol Appl (JoCTA) 5(3):6–18
- Burgin M (2015a) Inductive cellular automata. Int J Data Structures Algorithms 1(1):1–9
- Burgin M (2015b) Properties of stabilizing computations. Theory Appl Math Comput Sci 5(1):71–93
- Burgin M (2015c) Super-recursive algorithms and modes of computation. In: Proceedings of the 2015 European conference on software architecture workshops, Dubrovnik/Cavtat, 7–11 Sept 2015, ACM, pp 10:1–10:5
- Burgin M (2016a) Inductively computable sets and functions. J Comput Technol Appl (JoCTA) 7(1):12–22
- Burgin M (2016b) Inductively computable Hierarchies and inductive algorithmic complexity. Global J Comp Sci Technol., Ser. H: Inf Technol 16(1): 35–45
- Burgin M (2016c) Decreasing complexity in inductive computations. In: Advances in Unconventional computing, series emergence, complexity and computation, vol 22, Springer, Switzerland, pp 183–203
- Burgin M (2016d) On the power of oracles in the context of hierarchical intelligence. J Artif Intell Res Adv 3(2):6–17
- Burgin M (2016e) Inductive complexity and Shannon entropy. In: Information and complexity. World Scientific, New York/London/Singapore, pp 16–32
- Burgin M, Ades M (2009) Monte Carlo methods and superrecursive algorithms. In: Proceedings of the spring simulation multiconference (ADS, BIS, MSE, and MSEng), Society for Modeling and Simulation International, San Diego, pp 289–294
- Burgin M, Borodyanskiy YM (1991) Infinite processes and super-recursive algorithms. Not Acad Sci USSR 321(5):876–879. translated from Russian: 1992, v.44, No. 1

- Burgin MS, Borodyanskiy YM (1993a) Social processes and limit computations. In: Catastrophe, chaos, and self-organization in social systems. University of Koblenz-Landau, Koblenz, pp 117–123
- Burgin M, Borodyanskiy YM (1993b) Alphabetic operators and algorithms. *Cybern Syst Anal* (3): 42–57
- Burgin M, Debnath, N (2004) Measuring software maintenance. In: Proceedings of the ISCA 19th international conference “Computers and their applications”, ISCA, Seattle, pp 118–121
- Burgin M, Debnath N (2005) Complexity Measures for software engineering. *J Comput Methods Sci Eng* 5(1):127–143
- Burgin M, Debnath N (2009) Super-recursive algorithms in testing distributed systems. In: Proceedings of the ISCA 24th international conference “Computers and their applications” (CATA-2009), ISCA, New Orleans, pp 209–214
- Burgin M, Dodig-Crnkovic G (2013) From the closed classical algorithmic universe to an open World of algorithmic constellations. In: Computing nature, studies in applied philosophy, epistemology and rational ethics, vol 7. Springer, Berlin/Heidelberg, pp 241–254
- Burgin M, Eberbach E (2009a) Universality for Turing machines, inductive Turing machines and evolutionary algorithms. *Fundam Inf* 91(1):53–77
- Burgin M, Eberbach E (2009b) On Foundations of evolutionary computation: an evolutionary automata approach. In: Mo H (ed) Handbook of research on artificial immune systems and natural computing: applying complex adaptive technologies. IGI Global, Hershey, pp 342–360
- Burgin M, Eberbach E (2012) Evolutionary automata: expressiveness and convergence of evolutionary computation. *Comput J* 55(9):1023–1029
- Burgin M, Gupta B (2012) Second-level algorithms, Superrecursivity, and recovery problem in distributed systems. *Theory Comput Syst* 50(4):694–705
- Burgin M, Klinger A (2004a) Three aspects of super-recursive algorithms and hypercomputation or finding black swans. *Theor Comput Sci* 317(1/3):1–11
- Burgin M, Klinger A (2004b) Experience, generations, and limits in machine learning. *Theor Comput Sci* 317(1/3):71–91
- Burgin M, Mikkilineni R (2014) Semantic network organization based on distributed intelligent managed elements. In: Proceeding of the 6th international conference on advances in future Internet, Lisbon, pp 16–20
- Burgin M, Mikkilineni R (2016) Agent technology, super-recursive algorithms and DNA as a tool for distributed clouds and grids. In: Proceedings of the 25th IEEE international conference on enabling technologies: infrastructure for collaborative enterprises (WETICE 2016), Paris, 12–15 June, pp 89–94
- Burgin M, Shmidksi Y (1996) Is it possible to compute non-computable or why programmers need the theory of algorithms. *Comput Softw* 5:4–8
- Burgin M, Debnath N, Lee HK (2009) Measuring testing as a distributed component of the software life cycle. *J Comput Methods Sci Eng* 9(1/2)(Suppl 2):211–223
- Burgin M, Calude CS, Calude E (2011) Inductive complexity measures for mathematical problems, *Int J Found Comput Sci* 24(4): 487–500, 2013
- Burgin M, Mikkilineni R, Morana G (2016) Intelligent organization of semantic networks, DIME network architecture and grid automata. *Int J Embed Syst (IJES)* 8(4) pp 352–366
- Calude CS, Calude E (2012) Algorithmic complexity of mathematical problems: an overview of results and open problems, CDMTCS Research Report 410
- Calude CS, Calude E, Queen MS (2012) Inductive complexity of P versus NP problem. In: Unconventional computation and natural computation. Lecture notes in computer science, vol 7445. Springer, New York, pp 2–9
- Chaitin GJ (1977) Algorithmic information theory. *IBM J Res Dev* 21(4):350–359
- Church A (1932/33) A set of postulates for the foundations of logic. *Ann Math* 33: 346–366; 34: 839–864
- Cockshott P, Michaelson G (2007) Are there new models of computation? Reply to Wegner and Eberbach. *Comput J* 50(2):232–247
- Criscuolo G, Minicozzi E, Tratteur G (1975) Limiting recursion and the arithmetic hierarchy. *Rev Francaise Informat Recherche Opérationnelle* (Dec. 1975) 9:5–12
- Dinverno M, Luck M (eds) (2001) Understanding Agent Systems. Springer, New York
- Doyle J (1983) What is rational psychology? Toward a modern mental philosophy. *AI Mag* 4(3):50–53
- Dyson FJ (1972) Missed opportunities. *Bull Amer Math Soc* 78:635–652
- Eberbach E, Burgin M (2009) Evolutionary automata as foundations of evolutionary computation: Larry Fogel was right. In: Proceedings of 2009 congress on evolutionary computation (CEC’2009), Trondheim, pp 2149–2156
- Feferman S (1992) Turing’s ‘Oracle’: From absolute to relative computability - and back, in *The Space of Mathematics*, Walter de Gruyter, Berlin, pp 314–348
- Gödel K (1934) On undecidable propositions of Formal mathematical Systems. Lectures given at the Institute for Advanced Studies, Princeton, in *The Undecidable*, Raven Press, 1965, pp 39–71
- Gödel K (1990) Collected works, Vol. II, Publications 1938–1974. Oxford University Press, New York
- Gold EM (1965) Limiting recursion. *J Symb Log* 30(1):28–46
- Gold EM (1967) Language identification in the limit. *Inf Control* 10:447–474
- Hertel J (2012) Inductive complexity of Goodstein’s theorem. Unconventional computation and natural computation. Lecture notes in computer science, vol 7445. Springer, New York, pp 141–151

- Hintikka Ja, Mutanen A (1998) An alternative concept of computability. In: Language, truth, and logic in mathematics. Springer, Dordrecht, pp 174–188
- Hromkovic J (2005) Design and analysis of randomized algorithms. Springer, New York
- Kleene S (1936) General recursive functions of natural numbers. *Math Ann* 112(5):727–729
- Kolmogorov AN (1953) On the concept of algorithm. *Uspehi Mat Nauk* 8(4):175–176
- Kolmogorov AN (1965) Three approaches to the definition of the quantity of information. *Probl Inf Transm* 1(1):3–11
- Kugel P (2004) Toward a theory of intelligence. *Theor Comput Sci* 317(1/3):13–30
- Kugel P (2005) It's time to think outside the computational box. *Commun ACM* 48(11):32–37
- Li M, Vitanyi P (1997) An introduction to Kolmogorov complexity and its applications. Springer, New York
- Manin Y (2012) Renormalisation and computation II: time cut-off and the halting problem. *Comput Sci* 22:729–751
- McCarthy T, Shapiro S (1987) Turing projectability. *Notre Dame J Formal Logic* 28:520–535
- Mikkilineni R, Comparini A, Morana G (2012) The Turing o-machine and the DIME network architecture: injecting the architectural resiliency into distributed computing. In: Turing-100, The Alan Turing Centenary, EasyChair Proceedings in Computing, [www.easychair.org/publications/?page=877986046](http://www.easychair.org/publications/?page=877986046)
- Minsky M (1986) The Society of Mind. Simon and Schuster, New York
- Mutanen A (2004) From computation to truth via learning. Dissertation, University of Helsinki
- Putnam H (1965) Trial and error predicates and the solution to a problem of Mostowski. *J Symbolic Logic* 30(1):49–57
- Rogers H (1987) Theory of recursive functions and effective computability. MIT Press, Cambridge, MA
- Roglic D (2007) The universal evolutionary computer based on super-recursive algorithms of evolvability. Preprint in Computer Science, <http://arxiv.org/ftp/arxiv/papers/0708/0708.2686.pdf>
- Roglic D (2011) Super-recursive features of evolutionary processes and the models for computational evolution. In: Information and computation. World Scientific, Singapore, pp 331–379
- Schmidhuber J (2002) Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *Int J Found Comput Sci* 3(4):587–612
- Schubert LK (1974) Iterated limiting recursion and the program minimization problem. *J Assoc Comput Mach* 21:436–445
- Sloman A (2002) The irrelevance of Turing machines to AI (<http://www.cs.bham.ac.uk/~axs/>)
- Soare RI (2015) Turing Oracle machines, online computing, and three displacements in computability theory. *Ann Pure Appl Logic* 160:368–399
- Syropoulos A (2008) Hypercomputation: computing beyond the Church-Turing barrier. Springer, New York
- Thomas WJ (1979) A simple generalization of Turing computability. *Notre Dame J Formal Logic* 20:95–102
- Turing A (1936) On computable numbers with an application to the Entscheidungsproblem. *Proc Lond Math Soc, Ser 2* 42: 230–265
- Turing A (1950) Computing machinery and intelligence, mind 59:433–460
- Zilberstein S (1996) Using anytime algorithms in intelligent systems. *AI Mag* 17(3):73–83

---

# Index

## A

Accuracy, 3  
Acellular slime, 431  
Activator, 195  
Adaptive self-organization, 21  
Address decoding, 382  
Adiabatic quantum computing, 138–140  
Algorithmic coding theorem, 654  
Algorithmic cognition, 642  
Algorithmic probability, 654  
 $\alpha$ -shape, 431, 438  
Amoeboid robots, 442  
Amorphous computing, 402, 601–616  
Analog equation solvers, 45  
Analog reaction–diffusion chip, 232–233  
Analog very large scale integration, 9  
Ant colony optimization (ACO), 549, 554–555, 565, 568–569  
    principles and applications, 569–573  
Ants sleeping model (ASM), 570  
Arithmetic and logic operations, 584–585  
Artificial chemistry (AC), 401, 577–596  
    arithmetic and logic operations, 584–585  
    assembler automata, 587–588  
    autocatalytic polymer chemistries, 585  
    cellular automata, 588–589  
    chemical differential equation, 581–582  
    discrete event simulation, 582  
    explicitly simulated collisions, 582  
    extrinsic evolution, 593–594  
    information processing, 596  
    intrinsic evolution, 594–595  
    lambda calculus, 583–584  
    lattice molecular systems, 588–589  
    mechanical, 589–590  
    molecules, 579–580  
    reaction rules, 580–581  
    semi realistic molecular structures and graph rewriting, 589–590  
    space (*see* Space)  
    syntactic and semantic closure, 595–596  
    theory, 591–593  
    Turing machines, 585–587

Artificial immune networks, 509–510

Artificial immune system, 506–507  
Artificial negative selection, 510  
Artificial neural networks, 96  
Artificial neuron, 98  
Artificial organisms, 586  
Assembler automata, 587  
Autocatalytic polymer chemistries, 585  
Autocatalytic set, 591–593  
Autopoiesis, 592

## B

Babbage’s analytical engine, 48  
Babbage’s difference engine, 47–48  
Bacterial computing, 491–501  
Ballistic logical gates, 439–440  
Bat algorithm, 549, 556  
Bees-inspired algorithms, 549  
Belousov–Zhabotinsky (BZ) reaction, 171, 228  
Benenson’s Finite Automaton, 332–335  
Billiard ball computers, 40  
Billiard ball model, 463, 474–476  
Biological Quantum Computing, 142–143  
Block decomposition method, 658  
Broadcast-and-weight protocol, 105, 106  
Brownian machines, 40–41  
BZ processor, 175

## C

Cell Matrix, 405  
Cellular ants, 566  
    computing, 565–575  
    in data clustering, 570–571  
    in swarm robotics, 571–573  
Cellular automata, 38, 400–401, 565, 588–589  
    fundamentals, 567–568  
Cellular automaton, 373  
Cellular Automaton Machine (CAM), 566  
Cellular computing hardware, 403–405  
Cellular neural networks, 377  
Chemical differential equation, 581

Chemical nanocomputers, 360  
 Chemical sensors, 210–214  
 Church–Turing Conjecture, 676  
 Church–Turing computation, 3  
 Church–Turing thesis, 25, 676  
 Clockwork computer, 41  
 Cognitive immunology, 511–512  
 Coherence, 59–60  
 Collision-based computing, 40  
 Collision-free path, 178–180  
 Computability theory, 396  
 Computational completeness, 519  
 Computational complexity classes, 37  
 Computational complexity theory, 396  
 Computational efficiency, 530–531  
 Computational particle, 601  
 Computing power, 529–530  
 Concave hull, 431, 438  
 Conducting polymer, 454–455  
 Construction universality, 379  
 Continuous space machine, 57, 68–72  
 Continuous-time models, 25  
 Continuous-time quantum computing, 136–138  
 Continuous-time recurrent neural networks  
   (CTRNNs), 99  
 Controlled NOT, 465  
 Core network, 587  
 Counter machine, 484  
 Cross-gain modulation (XGM), 99  
 CrossNets, 385  
 Cuckoo search, 549, 558

**D**  
 Danger theory, 510–511  
 Dark silicon strategies, 87  
 Data clustering, 570–571  
 DC operational amplifier, 6  
 Delaunay triangulation, 431, 438  
 Delay-insensitive circuits, 367  
 Deoxyribozyme-based automaton, 327  
 Deoxyribozyme-based logic gate, 327  
 Difference engine, 47  
 Differential analyzers, 6, 45–46  
 Digital biosensors, 296–298  
 Digital CMOS reaction–diffusion chips,  
   228–232  
 Discrete event simulation, 582–583  
 Dissipative models, 26–27  
 Divide-and-Conquer Method, 658  
 DNA  
   binary counter, 327  
   circuit, 327  
   computing, 307, 327  
   logic gate, 327  
   self-assembly, 335–338  
   Sierpinski triangle, 327  
   tiling assemblies, 51  
 Domino tiling problems, 50

**E**  
 Echo State Networks, 620, 625–627  
 Efficiency in interconnection complexity, 59  
 Electroactive polymers, 454–455  
 Electromechanical computing, 49  
 Electronic nanocomputers, 360  
 Embryonics, 403–404  
 Energy efficiency, 59  
 Enigma Machines, 49  
 Enzymatic machine, 41  
 Enzyme-based logic gates, 267–271  
 E/O subcircuit, 99  
 Error correction, 125  
 Evacuation, 436–437  
 Evolution in materio, 447–461  
 Excitability, 195  
 Excitable behavior, 101  
 Excitable medium, 171  
 Experimental archeology, 436  
 EXPTIME, 37  
 Extended analog computer, 3

**F**  
 FANOUT, 152–154  
 Fault-tolerance, 370–373  
 Field programmable analog arrays, 9  
 Field programmable gate arrays, 404–405  
 Firefly algorithm, 549, 556–558  
 Fredkin gate, 465  
 Fredkin’s billiard ball model (BBM), 376  
 Frequency based logical gates, 440

**G**  
 Garbage-less computation, 477  
 Generalized communicating P systems, 528–529  
 General-purpose analog computer, 3  
 Genetic circuits, 21–22, 495–496  
 Geometry, 437–438  
 Gillespie’s algorithm, 582  
 Grossone, 667

**H**  
 Hardware Oregonator model, 232–233  
 Harmonic analyzers, 45  
 Harmonic synthesizers, 45  
 HICANN chip, 109  
 Hybrid quantum computing, 136

**I**  
 Immune computing, 503–514  
 Immune networks, 509  
 Inductive cognitive processes, 676  
 Inductive Turing machine, 675  
 Interacting variables, 634  
 Interference, 66

**J**

The Jevons' logic piano, 49

**K**

Kinodynamic planning, 42  
Kirchhoff–Łukasiewicz machines, 455–456  
Kolmogorov algorithms, 676  
Kolmogorov–Uspensky machine, 441–442

**L**

*Lac* operon, 494–495  
Langmuir–Blodgett films, 455  
Lattice molecular systems, 588  
Leaky integrate-and-fire (LIF), 97  
Learning, 96  
Lehmer's number sieve computer, 49  
Liquid crystal, 454  
Liquid State Machines, 620, 627  
Logic, 381  
    with memory, 418–419  
Lossless compression, 656

**M**

Machine learning, 504  
Manakov solitons, 150–151  
Manakov system, 147, 149  
Margolus' reversible cellular automaton, 376  
Mass action kinetics, 582  
Mass Migration, 436  
Matrix multiplication chemistry, 584–585  
Maze, 432  
McCulloch–Pitts neural model, 96  
Mechanical artificial chemistries, 589  
Mechanical cipher devices, 49  
Mechanical nanocomputers, 360  
Mechanistic model, 653–654  
Membrane algorithms, 540–544  
Membrane computing, 401, 519–532, 536  
Membrane structure, 519  
Memory, 382  
Memristor, 411, 412  
MEMS Computing Devices, 49  
Metaheuristic, 549  
Microelectromechanical systems, 49  
Micro-fluidic logical gates, 440–441  
Microring resonator (MRR), 106  
Minority-carrier transport, 234–235  
Mnemotrix, 411  
Molecular automata, 327–352  
Molecular automaton, 328  
Molecular finite-state automaton, 328  
Molecular Mealy automaton, 328  
Molecule cascades, 377  
Moore's law, 86  
Motion planning, 43–44  
Multiple-in multiple-out (MIMO) systems, 112

Multiplication of images, 67

Multiset, 519  
    rewriting, 519–520  
Multistable soliton collision, 159

**N**

Napier's bones, 47  
Negative selection, 510  
Networkable photonic device, 98  
Neural computation, 20–21  
Neural network, 383–384  
Neuromorphic computing, 84, 87–88  
Neuromorphic engineering, 384  
Neuromorphic microelectronics, 89–90  
Neuromorphic photonics, 84, 85, 90–96  
    challenges, 95–96  
Neuromorphic platforms comparison, 109–111  
No-free-lunch theorems, 559–560  
Nomograph, 3  
Nonlinear programming, 112–113  
Nonlinear Schrodinger equation, 147  
NP, 37  
Numbers, 663  
Numerals, 663

**O**

O/E/O PNN nonlinearities, 100  
Optical computer, 57  
Optical pattern recognition, 62  
Optimization, 550, 553  
Optoelectronics logical gates, 440  
Ordinary differential equation, 3  
Organic memristive device, 413–416, 418–423  
Oscillator, 416

**P**

P, 37  
Palladium processor, 171, 174  
Parallel actuators, 182–184  
Partial differential equation, 3  
Particle-like waves, 40  
Particle swarm optimization, 550, 555  
Pascaline, 47  
Pavlov's dog learning, 418  
PEO–PANI fibrillar networks, 424  
Perceptron, 411, 419–423  
Photonic integrated circuit (PIC), 84  
    fabrication, 92  
Photonic neural network architecture, 105–109  
Photonic neuron, 96–101  
Photonic reservoir computing, 94–95  
Photonic weight banks  
    controlling, 107–108  
    scalability with, 108–109  
*Physarum*, 442  
    *P. polycephalum*, 431

Physical computation, 447–449  
 Pinwheel calculators, 47  
 Polyaniline (PANI), 411, 412  
 Polyethylene oxide, 411  
 Polynomial time reduction, 37  
 Poly(styrene sulfonic acid)-b-poly-(ethylene oxide)-b-poly(styrene sulfonic acid) (PSS-b-PEO-b-PSS), 424  
 Pond snail nervous system, 417  
 Population-based algorithm, 550  
 Potentiometer, 3  
 Precipitation reaction, 172  
 Precision, 3, 17–18  
 PSPACE, 37  
 Pulse generator, 537

**Q**

Quadratic programming (QP), 113  
 Quantum annealing, 138–140  
 Quantum communications, 124  
 Quantum computing, 38–39, 119–143  
 Quantum dense coding, 124  
 Quantum dot cellular automaton, 376  
 Quantum key distribution, 124  
 Quantum limit, 360  
 Quantum mechanics, 121  
 Quantum nanocomputers, 360  
 Quantum superposition, 38  
 Quantum teleportation, 124  
 Quantum walks, 127–129, 140

**R**

Randomized Turing machines, 38  
 Rank-Varying Computational Complexity, 633  
 Rate coding, 97  
 Ray tracing, 42  
 Reaction–diffusion chemical computers, 46  
 Reaction–diffusion circuit, 228  
 Reaction network, 580  
 Reaction rules, 580–581  
 Real-time radio frequency processing, 111–112, 114  
 Recurrent neural networks, 26  
 Reduced Instruction Set Computers (RISC), 385  
 Register machine, 520  
 Regulated rewriting, 520  
 Reservoir computing, 94–95, 619–628  
 Reversible cellular automata, 376  
 Reversible cellular automaton, 463, 485  
 Reversible computing, 463–486  
 Reversible counter machine, 484  
 Reversible finite automaton, 484–485  
 Reversible logic element, 465–476  
     with memory, 463, 467  
 Reversible logic gate, 463, 465–467  
 Reversible Turing machine, 38, 463, 476–484  
 Ring Memory, 214–218  
 Robotic hand, 182

Rotary element, 468–475  
 Routing, 381  
 Rubel’s extended analog computer, 24

**S**

Scaling, 18–20  
 Self maintaining, 591, 592  
 Self-Timed Cellular Automata (STCA), 379  
 Self-timed circuits, 367  
 Semiconductor excitable lasers, 103–105  
     classification of, 99  
 Semiconductor optical amplifiers (SOAs), 95  
 Sequential machine, 467  
 Sequential-time models, 25–26  
 Shamir’s TWINKLE, 49  
 Shannon entropy, 654  
 Shor’s algorithm, 126–127  
 Shortest path, 42, 432  
 Signal cascability, 84  
 Silicon photonics, 84  
 Simulated annealing, 385  
 Single-electron circuit, 442  
 Single-electron reaction–diffusion system, 235–239  
 Slide rules, 47  
 Slime mold computing, 431–443  
 Social algorithms, 550, 549–561  
 Soliton, 147  
     computers, 40  
 Space  
     modeling space, techniques for, 589–590  
     phenomena in, 591  
 Space exploration, 437  
 Spanning tree, 431, 433–434  
 Spatial light modulators, 65  
 Spatial soliton collisions, 167–168  
 Speed-independent circuits, 367  
 Speed-of-light limit, 360  
 Spiking model, 88  
 Spiking neural networks (SNNs), 84, 89  
 Spiking neural P systems, 526–528  
 Spiking neuron, 97  
 Steiner tree, 442  
 Stepped drum calculators, 47  
 Stochastic networks, 423–427  
 Stoichiometric matrix, 580  
 Structure-to function mapping, artificial chemistry,  
     *see Artificial chemistry (AC)*  
 Sub-excitable BZ processor, 175  
 Subexcitable medium, 172  
 Surface plasmon resonance, 278–279  
 Swarm intelligence, 550  
 Symport/antiport, 520

**T**

Temporal coding, 97  
 The informal notion of algorithm is equivalent to concept  
     of a Turing machine, 676  
 Therapeutic and diagnostic automata, 340–343

- Thermal limit, 359  
Thermodynamics, 252–255  
Threshold multiplication, 584  
Time-division multiplexing (TDM), 88  
Time efficiency, 59  
Time gating, 155–156  
Toffoli gate, 465  
Towers of Hanoi, 432–433  
Transport networks, 434–436  
Travelling salesman problem, 433  
Turing machine, 36, 676  
Typogenetics, 586

**U**

- Ultra-wideband (UWB) radio system, 112  
Uncertain time constraints, 634

- Unconventional computational problem, 631  
Universal Distribution, 654  
Universality, 520  
Universal Turing machine, 36

**V**

- Vector solitons, 149–150  
Voltage-Controlled Colloids, 455  
Voronoi diagram, 172, 176–178, 229, 431, 437, 438

**W**

- Wavelength-division multiplexing (WDM)  
networking, 99  
protocol, 106