

**Giancarlo Mauri  
Alberto Dennunzio  
Luca Manzoni  
Antonio E. Porreca (Eds.)**

**LNCS 7956**

# **Unconventional Computation and Natural Computation**

**12th International Conference, UCNC 2013  
Milan, Italy, July 2013  
Proceedings**



 **Springer**

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*TU Dortmund University, Germany*

Madhu Sudan

*Microsoft Research, Cambridge, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max Planck Institute for Informatics, Saarbruecken, Germany*

Giancarlo Mauri Alberto Dennunzio  
Luca Manzoni Antonio E. Porreca (Eds.)

# Unconventional Computation and Natural Computation

12th International Conference, UCNC 2013  
Milan, Italy, July 1-5, 2013  
Proceedings

## Volume Editors

Giancarlo Mauri

Alberto Dennyunzio

Luca Manzoni

Antonio E. Porreca

Università degli Studi di Milano-Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Viale Sarca 336/14, 20126 Milan, Italy

E-mail: {mauri, dennunzio, luca.manzoni, porreca}@disco.unimib.it

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-642-39073-9

e-ISBN 978-3-642-39074-6

DOI 10.1007/978-3-642-39074-6

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013940604

CR Subject Classification (1998): F.1, F.2, I.1-2, C.1.3, C.1, J.2-3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

© Springer-Verlag Berlin Heidelberg 2013

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. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

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.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

*Typesetting:* Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

# Preface

Starting in 2012, the conference series previously known as Unconventional Computation (UC) changed its name to Unconventional Computation and Natural Computation (UCNC). The name change was initiated to reflect the evolution in the variety of fields in the past decade or so. The series is genuinely interdisciplinary and it covers theory as well as experiments and applications. It is concerned with computation that goes beyond the classic Turing model, such as human-designed computation inspired by nature, and with the computational properties of processes taking place in nature.

The topics of the conference typically include: quantum, cellular, molecular, neural, DNA, membrane, and evolutionary computing; cellular automata; computation based on chaos and dynamical systems; massive parallel computation; collective intelligence; computation based on physical principles such as relativistic, optical, spatial, collision-based computing; amorphous computing; physarum computing; hypercomputation; fuzzy and rough computing; swarm intelligence; artificial immune systems; physics of computation; chemical computation; evolving hardware; the computational nature of self-assembly, developmental processes, bacterial communication, and brain processes.

The first venue of the UCNC (previously UC) series was Auckland, New Zealand, in 1998. Subsequent sites of the conference were Brussels, Belgium, in 2000; Kobe, Japan, in 2002; Seville, Spain, in 2005; York, UK, in 2006; Kingston, Canada, in 2007; Vienna, Austria, in 2008; Ponta Delgada, Portugal, in 2009; Tokyo, Japan, in 2010; Turku, Finland, in 2011 and Orléans, France, in 2012. Each meeting was accompanied by its own proceedings<sup>1</sup>.

The 12<sup>th</sup> conference in the series, UCNC 2013, was organized in 2013 in Milan (Italy) by the Department of Informatics, Systems and Communication (DISCO) on the beautiful campus of the University of Milano-Bicocca during July 1–5, 2013.

Milan is situated in the north of Italy, in the middle of the vast area of the Padan Plain, in a truly strategic position for the paths that lead to the heart of Europe. It is the Italian capital of finance and advanced tertiary sector.

Milan is truly one of the few “complete” Italian cities, able to reconcile economic and social realities. It is active in many fields of culture and research. It is a busy and advanced metropolis that attracts millions of people every year, offering a multitude of opportunities in the fields of education, employment, entertainment, and tourism.

---

<sup>1</sup> See <http://www.cs.auckland.ac.nz/CDMTCS/conferences/uc/uc.html>.

The roots of Milan are planted in a past that has bestowed on us an artistic and cultural heritage; this is not rare for towns in Italy, but not all of them have so much to offer:

- The world-famous *L'Ultima Cena* (*The Last Supper*) by Leonardo Da Vinci
- The Opera House, La Scala
- The Sforza Castle
- The numerous museums and art galleries: many of the treasures of Milan are hidden to the less attentive eyes of its inhabitants, but it is all there, waiting to be discovered

Milan also has a rich calendar of events to cater for all tastes, be they cultural, recreational or sports; the city certainly has something to offer for everyone.

UCNC 2013 was co-located with Computability in Europe 2013 (CiE 2013), with three common invited speakers: Gilles Brassard (Université de Montréal), Grzegorz Rozenberg (Leiden Institute of Advanced Computer Science and University of Colorado at Boulder), and Endre Szemerédi (Hungarian Academy of Sciences, Rutgers University). Other invited speakers were Enrico Formenti (Université Nice Sophia Antipolis, France), John V. Tucker (Swansea University, UK), and Xin Yao (University of Birmingham, UK).

There were 46 submissions from 26 countries including Austria, Bangladesh, Canada, Finland, France, Germany, Hungary, India, Iran, Italy, Japan, Latvia, Malaysia, Moldova, Morocco, New Zealand, Norway, Philippines, Poland, Portugal, Romania, Spain, Sweden, Turkey, UK, and the USA. Each paper was reviewed by three referees and discussed by the members of the Program Committee. Finally, 20 regular papers were selected for presentation at the conference. In addition, there were eight posters on display at the conference.

We warmly thank all the invited speakers and all the authors of the submitted papers. Their efforts were the basis of the success of the conference. We would like to thank all the members of the Program Committee and the external referees. Their work in evaluating the papers and comments during the discussions was essential to the decisions on the contributed papers. We would also like to thank all the members of the UCNC Steering Committee, for their ideas and efforts in forming the Program Committee and selecting the invited speakers.

We wish to thank the conference sponsors: the University of Milano-Bicocca, the Italian Chapter of the European Association for Theoretical Computer Science, and Micron Foundation.

The conference has a long history of hosting workshops. The 2013 edition in Milan hosted three workshops:

- CoSMoS 2013, the 6th International Workshop on Complex Systems Modelling and Simulation<sup>2</sup> (Monday, July 1)

---

<sup>2</sup> <http://www.cosmos-research.org/cosmos2013.html>

- BioChemIT 2013, the Third COBRA Workshop on Biological and Chemical Information Technologies<sup>3</sup> (Friday, July 5)
- WIVACE 2013, the Italian Workshop on Artificial Life and Evolutionary Computation<sup>4</sup> (July 1-2)

July 2013

Giancarlo Mauri  
Alberto Dennunzio  
Luca Manzoni  
Antonio E. Porreca

---

<sup>3</sup> [http://www.cobra-project.eu/biochemit2013\\_call.html](http://www.cobra-project.eu/biochemit2013_call.html)

<sup>4</sup> <http://wivace2013.disco.unimib.it/>

# Organization

## Program Committee

Andrew Adamatzky	University of the West of England, UK
Selim G. Akl	Queen's University, Canada
Olivier Bournez	École Polytechnique, France
Cristian S. Calude	University of Auckland, New Zealand
José Félix Costa	Technical University of Lisbon, Portugal
Erzsébet Csuhaj-Varjú	Eötvös Loránd University, Hungary
Alberto Dennunzio	University of Milano-Bicocca, Italy
Michael J. Dinneen	University of Auckland, New Zealand
Marco Dorigo	Université Libre de Bruxelles, Belgium
Jérôme Durand-Lose	Université d'Orléans, France
Masami Hagiya	University of Tokyo, Japan
Oscar H. Ibarra	University of California, Santa Barbara, USA
Jarkko Kari	University of Turku, Finland
Lila Kari	University of Western Ontario, Canada
Viv Kendon	University of Leeds, UK
Giancarlo Mauri	University of Milano-Bicocca, Italy (Chair)
Mario J. Pérez-Jiménez	Universidad de Sevilla, Spain
Kai Salomaa	Queen's University, Canada
Hava Siegelmann	University of Massachusetts Amherst, USA
Susan Stepney	University of York, UK
Hiroshi Umeo	Osaka Electro-Communication University, Japan
Leonardo Vanneschi	Universidade Nova de Lisboa, Portugal
Damien Woods	California Institute of Technology, USA

## Steering Committee

Thomas Back	Leiden University, The Netherlands
Cristian S. Calude	University of Auckland, New Zealand (Founding Chair)
Lov K. Grover	Bell Labs, Murray Hill, New Jersey, USA
Nataša Jonoska	University of South Florida, USA (Co-chair)
Jarkko Kari	University of Turku, Finland (Co-chair)
Lila Kari	University of Western Ontario, Canada
Seth Lloyd	Massachusetts Institute of Technology, USA
Giancarlo Mauri	University of Milano-Bicocca, Italy
Gheorghe Păun	Institute of Mathematics of the Romanian Academy, Romania



Grzegorz Rozenberg	Leiden University, The Netherlands (Emeritus Chair)
Arto Salooma	University of Turku, Finland
Tommaso Toffoli	Boston University, USA
Carme Torras	Institute of Robotics and Industrial Informatics, Barcelona, Spain
Jan van Leeuwen	Utrecht University, The Netherlands

## Organizing Committee

Alberto Dennunzio	University of Milano-Bicocca, Italy
Teresa Gallicchio	University of Milano-Bicocca, Italy
Luca Manzoni	University of Milano-Bicocca, Italy
Giancarlo Mauri	University of Milano-Bicocca, Italy
Antonio E. Porreca	University of Milano-Bicocca, Italy (Chair)
Pamela Pravettoni	University of Milano-Bicocca, Italy
Silvia Robaldo	University of Milano-Bicocca, Italy
Mariella Talia	University of Milano-Bicocca, Italy

## Additional Reviewers

James Aspnes	Diogo Poças
Florent Becker	Alexandru Popa
Ed Blakey	Antonio E. Porreca
Mike Domaratzki	Simon Poulding
Claudio Ferretti	Giuseppe Prencipe
Yuan Gao	Afroza Rahman
Zsolt Gazdag	Bala Ravikumar
Sama Goliaei	John Reif
Emmanuel Hainry	Andrei Romashchenko
Stuart Hasting	Francisco J. Romero-Campero
Mika Hirvensalo	Louis Rose
Tarik Kaced	Adam Sampson
Yoshihiko Kakutani	Shinnosuke Seki
Ibuki Kawamata	José M. Sempere
Yun-Bum Kim	Amirhossein Simjour
Satoshi Kobayashi	Yasuhiro Suzuki
Steffen Kopecki	Krisztián Tichler
Raluca Lefticaru	Nicholas Tran
Roberto Leporini	Kuai Wei
Friedhelm Meyer auf der Heide	Abuzer Yakaryilmaz
Julian Miller	Claudio Zandron
Valtteri Niemi	Rosalba Zizza
Marco S. Nobile	

## Sponsors

We deeply thank the sponsors that made UCNC 2013 possible:



**Università degli Studi  
di Milano-Bicocca**



**Dipartimento di Informatica,  
Sistemistica e Comunicazione**



**European Association for  
Theoretical Computer Science  
Italian Chapter**



**Micron Foundation**

# Table of Contents

## Invited Papers

Asymptotic Dynamics of (Some) Asynchronous Cellular Automata (Abstract) .....	1
<i>Enrico Formenti</i>	
Processes Inspired by the Functioning of Living Cells: Natural Computing Approach (Abstract) .....	3
<i>Andrzej Ehrenfeucht and Grzegorz Rozenberg</i>	

## Regular Papers

On the Power of Threshold Measurements as Oracles .....	6
<i>Edwin Beggs, José Félix Costa, Diogo Poças, and John V. Tucker</i>	
Size Lower Bounds for Quantum Automata .....	19
<i>Maria Paola Bianchi, Carlo Mereghetti, and Beatrice Palano</i>	
Population Protocols on Graphs: A Hierarchy .....	31
<i>Olivier Bournez and Jonas Lefèvre</i>	
Spectral Representation of Some Computationally Enumerable Sets with an Application to Quantum Provability .....	43
<i>Cristian S. Calude and Kohtaro Tadaki</i>	
On the Power of P Automata .....	55
<i>Erzsébet Csuhaj-Varjú and György Vaszil</i>	
Array Insertion and Deletion P Systems .....	67
<i>Henning Fernau, Rudolf Freund, Sergiu Ivanov, Markus L. Schmid, and K.G. Subramanian</i>	
Boolean Logic Gates from a Single Memristor via Low-Level Sequential Logic .....	79
<i>Ella Gale, Ben de Lacy Costello, and Andrew Adamatzky</i>	
Light Ray Concentration Reduces the Complexity of the Wavelength-Based Machine on PSPACE Languages .....	90
<i>Sama Goliaei and Mohammad-Hadi Foroughmand-Araabi</i>	
Small Steps toward Hypercomputation via Infinitary Machine Proof Verification and Proof Generation .....	102
<i>Naveen Sundar Govindarajulu, John Licato, and Selmer Bringsjord</i>	

Secure Information Transmission Based on Physical Principles . . . . .	113
<i>Dima Grigoriev and Vladimir Shpilrain</i>	
Hypergraph Automata: A Theoretical Model for Patterned Self-assembly . . . . .	125
<i>Lila Kari, Steffen Kopecki, and Amirhossein Simjour</i>	
Modeling Heart Pacemaker Tissue by a Network of Stochastic Oscillatory Cellular Automata . . . . .	138
<i>Danuta Makowiec</i>	
Reaction Systems Made Simple: A Normal Form and a Classification Theorem . . . . .	150
<i>Luca Manzoni and Antonio E. Porreca</i>	
Voting with a Logarithmic Number of Cards . . . . .	162
<i>Takaaki Mizuki, Isaac Kobina Asiedu, and Hideaki Sone</i>	
Asynchronous Signal Passing for Tile Self-assembly: Fuel Efficient Computation and Efficient Assembly of Shapes . . . . .	174
<i>Jennifer E. Padilla, Matthew J. Patitz, Raul Pena, Robert T. Schweller, Nadrian C. Seeman, Robert Sheline, Scott M. Summers, and Xingsi Zhong</i>	
Control Languages Associated with Tissue P Systems . . . . .	186
<i>Ajeesh Ramanujan and Kamala Krithivasan</i>	
Geometric Methods for Analysing Quantum Speed Limits: Time-Dependent Controlled Quantum Systems with Constrained Control Functions . . . . .	198
<i>Benjamin Russell and Susan Stepney</i>	
Numerical Analysis of Quantum Speed Limits: Controlled Quantum Spin Chain Systems with Constrained Control Functions . . . . .	209
<i>Benjamin Russell and Susan Stepney</i>	
Combinatorial Optimization in Pattern Assembly . . . . .	220
<i>Shinnosuke Seki</i>	
Towards Computation with Microchemomechanical Systems . . . . .	232
<i>Andreas Voigt, Rinaldo Greiner, Merle Allerdießen, and Andreas Richter</i>	

**Posters**

Evolutionary Programming Using Distribution-Based and Differential Mutation Operators . . . . .	244
<i>Md. Tanvir Alam Anik and Saif Ahmed</i>	

A P System Parsing Word Derivatives . . . . .	246
<i>Artiom Alhazov, Elena Boian, Svetlana Cojocaru,</i> <i>Alexandru Colesnicov, Ludmila Malahov, Mircea Petic, and</i> <i>Yurii Rogozhin</i>	
Computational Power of Protein Interaction Networks . . . . .	248
<i>Bogdan Aman and Gabriel Ciobanu</i>	
Towards an All-Optical Soliton FFT in the 3NLS-Domain . . . . .	250
<i>Anastasios G. Bakaoukas</i>	
Quantum Random Active Element Machine . . . . .	252
<i>Michael Stephen Fiske</i>	
Simulating Metabolic Processes Using an Architecture Based on Networks of Bio-inspired Processors . . . . .	255
<i>Sandra Gómez Canaval, José Ramón Sánchez, and Fernando Arroyo</i>	
On String Reading Stateless Multicounter $5' \rightarrow 3'$ Watson-Crick Automata . . . . .	257
<i>László Hegedüs and Benedek Nagy</i>	
Relating Transition P Systems and Spiking Neural P Systems . . . . .	259
<i>Richelle Ann B. Juayong, Nestine Hope S. Hernandez,</i> <i>Francis George C. Cabarle, and Henry N. Adorna</i>	
<b>Author Index</b> . . . . .	261

# Asymptotic Dynamics of (Some) Asynchronous Cellular Automata<sup>\*</sup>

## (Abstract)

Enrico Formenti

Université Nice Sophia Antipolis, Laboratoire I3S,  
2000 Route des Colles, 06903 Sophia Antipolis, France  
`enrico.formenti@unice.fr`

Cellular automata are a well-known discrete model for complex systems characterized by local interactions. Indeed, these local interactions cause the emergence of a global complex behavior. Cellular automata essentially consist in an infinite number of identical finite automata arranged on regular grid ( $\mathbb{Z}$  in this talk). Each automaton updates its state on the basis of a local rule which takes into account the state of a fixed number of neighboring automata.

One of the main features of the cellular automata model is that all updates are synchronous and according to the same local rule. Both of these last requirements can be an issue when simulating many natural phenomena. Indeed, strong homogeneity of time or space seems difficult to apply in some contexts (chemical reactions in a cell, for instance).

For these reasons, a number of variants relaxing one of these constraints has been introduced [1,2,4,6,7,8]. In this talk we focus on asynchronicity. After a short state of art, the attention is turned to  $m$ -ACA, a new computational model which has been introduced in order to try to give a common playground for a large part of the asynchronous CA models. The main idea is to have some kind of oracle that, at each time step, tells which cells should be updated. Clearly, the choice of the oracle deeply influences the final model and its behavior. Care should be taken to avoid either to fall back to the standard model or to completely destroy computational capabilities [3].

Therefore, the proposal is that each cell (or group of cells) is given a probability measure which tells if the cell (or the group) should be updated or not. Moreover, all probabilities are independently distributed and follow some *fairness* conditions to assure that a cell has a non-zero probability of being eventually updated and that it is not always updated.

In [3], several basic properties like  $(\mu)$ -surjectivity and  $(\mu)$ -injectivity are investigated proving that almost surely  $(\mu)$ -surjective CA are almost surely  $(\mu)$ -injective and vice-versa. The study of more complex dynamical properties has just started. Some properties are given for the equicontinuous behavior (i.e. the less topologically complex ) but the overall impression is that finding very

---

<sup>\*</sup> This work has been partially supported by the French National Research Agency project EMC (ANR-09-BLAN-0164) and by PRIN/MIUR project “Mathematical aspects and forthcoming applications of automata and formal languages”.

general results is rather difficult. In the spirit of [5], we focus on some particular sub-classes to get some intuitions in the hope of being able to find general principles. For the special case of the shift CA (all states are shifted left) we are going to prove that any fair measure forces an homogenization process so that, in the limit, with high probability, we shall observe only configurations in which all automata are in the same state. A similar behavior is observed for a large class of CA that we try to characterize.

**Acknowledgements.** The author warmly thanks Alberto Dennunzio, Luca Manzoni and Giancarlo Mauri for sharing the “*m*-ACA adventure”.

## References

1. Bersini, H., Detours, V.: Asynchrony induces stability in cellular automata based models. In: Proceedings of Artificial Life IV, pp. 382–387. MIT Press, Cambridge (1994)
2. Buvel, R.L., Ingerson, T.E.: Structure in asynchronous cellular automata. *Physica D* 1, 59–68 (1984)
3. Dennunzio, A., Formenti, E., Manzoni, L., Mauri, G.: *m*-asynchronous cellular automata. In: Sirakoulis, G.C., Bandini, S. (eds.) ACRI 2012. LNCS, vol. 7495, pp. 653–662. Springer, Heidelberg (2012)
4. Fatès, N., Morvan, M.: An experimental study of robustness to asynchronism for elementary cellular automata. *Complex Systems* 16(1), 1–27 (2005)
5. Fatès, N., Thierry, E., Morvan, M., Schabanel, N.: Fully asynchronous behavior of double-quiescent elementary cellular automata. *Theor. Comput. Sci.* 362(1-3), 1–16 (2006)
6. Lumer, E.D., Nicolis, G.: Synchronous versus asynchronous dynamics in spatially distributed systems. *Physica D* 71, 440–452 (1994)
7. Regnault, D., Schabanel, N., Thierry, E.: Progresses in the analysis of stochastic 2d cellular automata: A study of asynchronous 2d minority. *Theoretical Computer Science* 410, 4844–4855 (2009)
8. Schönfisch, B., de Roos, A.: Synchronous and asynchronous updating in cellular automata. *BioSystems* 51, 123–143 (1999)

# Processes Inspired by the Functioning of Living Cells: Natural Computing Approach

## (Abstract)

Andrzej Ehrenfeucht<sup>1</sup> and Grzegorz Rozenberg<sup>1,2</sup>

<sup>1</sup> Department of Computer Science  
University of Colorado at Boulder  
Boulder, CO 80309, U.S.A.

<sup>2</sup> Leiden Institute of Advanced Computer Science (LIACS)  
Leiden University  
Niels Bohrweg 1, 2300 RA Leiden  
The Netherlands  
`rozenber@liacs.nl`

Natural Computing (see, e.g., [12,13]) is concerned with human-designed computing inspired by nature as well as with computation taking place in nature, i.e., it investigates models, computational techniques, and computational technologies inspired by nature as well as it investigates, in terms of information processing, phenomena/processes taking place in nature.

Examples of the first strand are evolutionary, neural, molecular, and quantum computation, while examples of the second strand are investigations into the computational nature of self-assembly, the computational nature of developmental processes and the computational nature of biochemical reactions. Obviously, the two research strands are not disjoint.

A computational understanding of the functioning of the living cell is one of the research topics from the second strand. A motivation for this research is nicely formulated by Richard Dawkins, a world leading expert in evolutionary biology: "If you want to understand life, don't think about vibrant throbbing gels and oozes, think about information technology", see [4].

We view this functioning in terms of formal processes resulting from interactions between individual reactions, where these interactions are driven by two mechanisms, facilitation and inhibition: reactions may (through their products) facilitate or inhibit each other.

We present a formal framework for the investigation of processes resulting from these interactions. We provide the motivation by explicitly stating a number of assumptions that hold for these interactive processes, and we point out that these assumptions are very different from assumptions underlying traditional models of computation.

The central formal model of our framework, reaction systems (see [1,5,10]), follows the philosophy of processes outlined above, and moreover:

- (1) it takes into account the basic bioenergetics (flow of energy) of the living cell,



- (2) it abstracts from various technicalities of biochemical reactions to the extent that it becomes a qualitative rather than a quantitative model, and
- (3) it takes into account the fact that the living cell is an open system and so its behavior (expressed by formal processes) is influenced by its environment.

Our full formal framework (see [1,5]) contains also models that are extensions of the basic model of reaction systems. The research themes investigated within this framework are motivated either by biological considerations or by the need to understand the underlying computations. Some examples of these themes are:

- the notion of time in reaction systems, see [11],
- formation of modules in biological systems, see [9,16],
- understanding decay and its influence on interactive processes, see [3],
- how to include in our framework quantitative aspects of processes in living cells, see [1,5,9,11],
- static and dynamic causalities, see [2],
- the nature of state transitions in reaction systems, see [6,14,8,15].

We (hope to) demonstrate that the framework of reaction systems is:

- (i) well motivated by and relevant for biological considerations, and
- (ii) novel and attractive from the theory of computation point of view.

## References

1. Brijder, R., Ehrenfeucht, A., Main, M.G., Rozenberg, G.: A tour of reaction systems. *International Journal of Foundations of Computer Science* 22(7), 1499–1517 (2011)
2. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: A note on Causalities in Reaction Systems. *Electronic Communications of ECASST* 30 (2010)
3. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: Reaction Systems with Duration. In: Kelemen, J., Kelemenová, A. (eds.) *Computation, Cooperation, and Life*. LNCS, vol. 6610, pp. 191–202. Springer, Heidelberg (2011)
4. Dawkins, R.: *The Blind Watchmaker*. Penguin, Harmondsworth (1986)
5. Ehrenfeucht, A., Kleijn, J., Koutny, M., Rozenberg, G.: Qualitative and Quantitative Aspects of a Model for Processes Inspired by the Functioning of the Living Cell. In: Katz, E. (ed.) *Biomolecular Information Processing. From Logic Systems to Smart Sensors and Actuators*, pp. 303–322. Wiley-VCH Verlag, Weinheim (2012)
6. Ehrenfeucht, A., Kleijn, J., Koutny, M., Rozenberg, G.: Minimal Reaction Systems. In: Priami, C., Petre, I., de Vink, E. (eds.) *Transactions on Computational Systems Biology XIV*. LNCS (LNBI), vol. 7625, pp. 102–122. Springer, Heidelberg (2012)
7. Ehrenfeucht, A., Main, M.G., Rozenberg, G.: Combinatorics of life and death for reaction systems. *International Journal of Foundations of Computer Science* 21(3), 345–356 (2010)
8. Ehrenfeucht, A., Main, M.G., Rozenberg, G.: Functions defined by reaction systems. *International Journal of Foundations of Computer Science* 21(1), 167–178 (2011)
9. Ehrenfeucht, A., Rozenberg, G.: Events and Modules in Reaction Systems. *Theoretical Computer Science* 376(1-2), 3–16 (2007)

10. Ehrenfeucht, A., Rozenberg, G.: Reaction Systems. *Fundamenta Informaticae* 75(1-4), 263–280 (2007)
11. Ehrenfeucht, A., Rozenberg, G.: Introducing Time in Reaction Systems. *Theoretical Computer Science* 410(4-5), 310–322 (2009)
12. Kari, L., Rozenberg, G.: The many facets of natural computing. *Communications of the ACM* 51(10), 72–83 (2008)
13. Rozenberg, G., Bäck, T., Kok, J. (eds.): *Handbook of Natural Computing*. Springer (2012)
14. Salomaa, A.: On state sequences defined by reaction systems. In: Constable, R.L., Silva, A. (eds.) *Logic and Program Semantics*. LNCS, vol. 7230, pp. 271–282. Springer, Heidelberg (2012)
15. Salomaa, A.: Functions and sequences generated by reaction systems. *Theoretical Computer Science* 466, 87–96 (2012)
16. Schlosser, G., Wagner, G.P. (eds.): *Modularity in Development and Evolution*. The University of Chicago Press, Chicago (2004)

# On the Power of Threshold Measurements as Oracles

Edwin Beggs<sup>1</sup>, José Félix Costa<sup>2,3</sup>, Diogo Poças<sup>2,3,\*</sup>, and John V. Tucker<sup>1</sup>

<sup>1</sup> College of Science, Swansea University, Singleton Park, Swansea SA2 8PP,  
Wales, United Kingdom

<sup>2</sup> Department of Mathematics, Instituto Superior Técnico, UTL, Lisbon

<sup>3</sup> Centro de Matemática e Aplicações Fundamentais, University of Lisbon  
diogopocas1991@gmail.com

**Abstract.** We consider the measurement of physical quantities that are thresholds. We use hybrid computing systems modelled by Turing machines having as an oracle physical equipment that measures thresholds. The Turing machines compute with the help of qualitative information provided by the oracle. The queries are governed by timing protocols and provide the equipment with numerical data with (a) infinite precision, (b) unbounded precision, or (c) finite precision. We classify the computational power in polynomial time of a canonical example of a threshold oracle using non-uniform complexity classes.

## 1 Introduction

Computation and measurement are intimately connected in all sorts of ways. Measurement is a scientific activity supported by a comprehensive axiomatic theory developed in the 20th Century using the methods of mathematical logic (see [9,11]). We are developing a new theory of measurement processes from an algorithmic perspective, in a series of papers (see [2,3,4,5,6,7,8]). At the heart of our theory is the idea that an experimenter measures a physical quantity by applying an algorithmic procedure to equipment, and that we can model *the experimenter as a Turing machine and the equipment as a device connected to the Turing machine as a physical oracle*. The Turing machine abstracts the experimental procedure, encoding the experimental actions as a program. The physical oracle model is rather versatile: for example, it accommodates using the measurements in subsequent computations and, indeed, arbitrary interactions with equipment. Some implications for the axiomatic theory have been considered in [5].<sup>1</sup> Case studies shape the development of the theory. The standard oracle to a Turing machine is a set that contains information to boost the power and efficiency of computation: a query is a set membership question that is answered in one time step. Experiments require queries based upon rational numbers (dyadic rationals denoted by finite binary strings).

---

\* Corresponding author.

<sup>1</sup> Scientific activity seen as a Turing machine can be found in computational learning theory (see [10]).

The measurement of distance taught us that oracles involve information with possible error (see [2,4,8]). The measurement of a mass taught us that oracles may take considerable time to consult (such an experiment is fully analysed in [6,8]). An important difference is the need of a cost function  $T$  of the size of the query (e.g., a timer) as part of the Turing machine. To measure the value of a physical quantity, i.e., a real number  $\mu$ , the experimenter (= the Turing machine) proceeds to construct approximations (which are generated by *oracle consultations*). Whenever possible, a measurement procedure should approximate the unknown quantity from above and from below, in a series of experimental values that converges. We call such experiments *two-sided*. Two-sided measurement is the focus of our previous work (see [8]), and that of axiomatic measurement theory (see [9]). However, not all measurements can be made this way: some quantities by their nature, or by the nature of the equipment, are *thresholds that can only be approximated either just from below or just from above*. Examples are experiments on activation thresholds for the neurone and Rutherford scattering. We study threshold experiments in this paper, which are complex and are not yet addressed in the literature. We prove the following new theorems that indicate the computational power of threshold oracles and reveal differences with the less complex two-sided case:

**Theorem 1.1.** (1) *If a set  $A$  is decided in polynomial time by an oracle Turing machine coupled with a threshold oracle of infinite precision, then  $A \in P/\log^2\star$ . If a set  $A$  is in  $P/\log\star$ , then  $A$  is decided by a oracle Turing machine coupled with a threshold oracle of infinite precision.*<sup>2</sup> (2) *If a set  $A$  is decided by an oracle Turing machine coupled with a threshold oracle of unbounded or fixed precision, then  $A \in BPP//\log^2\star$ . If a set  $A$  is in  $BPP//\log\star$ , then  $A$  is decided in polynomial time by a oracle Turing machine coupled with a threshold oracle.*<sup>3,4</sup>

Threshold oracles have not yet been considered in the literature (e.g., in [4]) and the results about two-sided oracles do not apply to these systems. Moreover, the upper bound known so far for the two-sided oracles with non-infinite precision is  $P/\text{poly}$  (except for particular types of two-sided oracles considered in [4] and [7] for which the upper bounds are  $P/\text{poly}$  and  $BPP//\log\star$ , respectively).

<sup>2</sup> Let  $\mathcal{B}$  be a class of sets and  $\mathcal{F}$  a class of functions. The advice class  $\mathcal{B}/\mathcal{F}$  is the class of sets  $A$  for which there exists  $B \in \mathcal{B}$  and some  $f \in \mathcal{F}$  such that, for every word  $w$ ,  $w \in A$  if and only if  $\langle w, f(|w|) \rangle \in B$ . For the prefix advice class  $\mathcal{B}/\mathcal{F}\star$  some (prefix) function  $f \in \mathcal{F}$  must exist such that, for all words  $w$  of length less than or equal to  $n$ ,  $w \in A$  if and only if  $\langle w, f(n) \rangle \in B$ . The role of advices in computation theory is fully discussed e.g. in [1], Chapter 5. We use  $\log^2$  to denote the class of advice functions such that  $|f(n)| \in \mathcal{O}((\log(n))^2)$ .

<sup>3</sup>  $BPP//\mathcal{F}\star$  is the class of sets  $A$  for which a probabilistic Turing machine  $\mathcal{M}$ , a prefix function  $f \in \mathcal{F}\star$ , and a constant  $\gamma < \frac{1}{2}$  exist such that, for every length  $n$  and input  $w$  with  $|w| \leq n$ ,  $\mathcal{M}$  rejects  $\langle w, f(n) \rangle$  with probability at most  $\gamma$  if  $w \in A$  and accepts  $\langle w, f(n) \rangle$  with probability at most  $\gamma$  if  $w \notin A$ .

<sup>4</sup> Note that in experiments where the lower/upper bounds are  $P/\text{poly}$  for the infinite precision case, the unbounded comes together because  $BPP//\text{poly} = P/\text{poly}$ . In the threshold experiments, however, the unbounded and finite precision cases display identical power.

## 2 Threshold Experiments

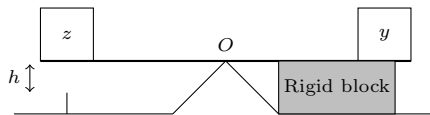
We will begin by listing some examples of threshold experiments and then we will focus on one particular experiment, the *broken balance experiment*, which is canonical.

### 2.1 The Squid Giant Motor Neurone

This first threshold experiment is inspired in the spiking neurone, such as the squid giant motor neurone, and it is designed to measure the threshold of activation: an electric current  $\iota$  is injected into the cell and the action potential, once generated, can be detected along the axon. Suppose that the rest (membrane) potential is  $\nu_0$  ( $\nu_0 \sim -65mV$ ) and that the threshold electric current is  $\iota_0$  ( $\iota_0 \sim 2nA$ ). The goal is to measure the threshold  $\iota_0$ , for some concentration of the ions: (a) if  $\iota < \iota_0$ , then no signal is sent along the axon and (b) if  $\iota \geq \iota_0$ , a series of action potentials is propagated along the axon. Once the current is switched off, the rest potential is reset.

### 2.2 The Photoelectric Effect Experiment

The equipment consists of a metallic surface, a source of monochromatic light and an electron detector. Each photon of the light beam has energy  $E = hf$ , where  $h$  is the Planck constant and  $f$  is the frequency of the light. On the other hand, the metallic surface is characterised by a value  $\phi = hf_0$  of energy, where  $f_0$  is the minimum (threshold) frequency required for photoelectric emission. The goal is to measure  $f_0$ , and to that end we can send a light beam with frequency  $f$ : (a) if  $f \leq f_0$ , then no electron escapes the surface and (b) if  $f > f_0$ , then the electrons are ejected with kinetic energy  $E = h(f - f_0)$ . In this way, the photoelectric experiment is a threshold experiment, since we only get a response whenever the light beam frequency exceeds the threshold frequency.



**Fig. 1.** Schematic representation of the *broken balance experiment*

### 2.3 The BBE: Broken Balance Experiment

The experiment consists of a balance scale with two pans (see Figure 1). In the right pan we have some body with an unknown mass  $y$ . To measure  $y$  we place test masses  $z$  on the left pan of the balance: (a) if  $z < y$ , then the scale will not move since the rigid block prevents the right pan from moving down, (b) if  $z > y$ , then the left pan of the scale will move down, which will be detected in some

way and (c) if  $z = y$ , then we assume that the scale will not move since it is in equilibrium. We assume the following characteristics of the apparatus inter alia: (a)  $y$  is a real number in  $[0, 1]$ , (b) the mass  $z$  can be set to any dyadic rational in the interval  $[0, 1]$ , and (c) a pressure-sensitive stick is placed below the left side of the balance, such that, when the left pan touches the pressure-sensitive stick, it reacts producing a signal. In the context of classical, pure Newtonian mechanics of the rigid body, in the perfect Platonic world, once we assume that the test mass weighs  $z$  and the unknown mass weighs  $y$ , the cost of the experiment,  $T_{exp}(z, y)$ , which is the time taken for the left pan of the balance to touch the pressure stick, is given by:

$$[T_{exp}(z, y) = \tau \times \sqrt{\frac{z + y}{\max(0, z - y)}} \text{ for all } y, z \in \mathbb{R}.^5] \tag{1}$$

### 3 The BBE Machine as a Means to Measure Real Numbers

In what follows the suffix operation  $\downarrow_n$  on a word  $w$ ,  $w\downarrow_n$ , denotes the prefix sized  $n$  of the  $\omega$ -word  $w\omega$ , no matter the size of  $w$ . Letters such as  $a, b, c, \dots$ , denote constants. To the oracle Turing machine model  $\mathcal{M}$  we associate a schedule  $T : \mathbb{N} \rightarrow \mathbb{N}$ , a time constructible function such that  $T(\ell)$  steps of busy waiting of  $\mathcal{M}$  are needed for the oracle to provide an answer ‘YES’ or ‘TIMEOUT’, resulting in a transition of  $\mathcal{M}$  to the state  $q_{YES}$  or  $q_{TIMEOUT}$ , respectively. Everything about setting an oracle Turing machine coupled with a physical experiment as oracle can be found in [2,6].

A larger variety of experiments could have been mentioned (such as Rutherford’s scattering experiment). However, since the BBE is fairly simple to analyse and understand, and as it displays the properties of threshold experiments, we will focus on it. Just as in previous investigations (see, e.g., [2,6,7]), we will consider different types of precision, i.e., different communication protocols between the experimenter/Turing machine and the oracle/analogue device: (a) *infinite precision*: when the dyadic  $z$  is read in the query tape, a test mass  $z$  is simultaneously placed in the left pan, (b) *unbounded precision*: when the dyadic  $z$  is read in the query tape, a test mass  $z'$  is simultaneously placed in the left pan such that  $z - 2^{-|z|} \leq z' \leq z + 2^{-|z|}$  and (c) *fixed precision*  $\epsilon > 0$ : when the dyadic  $z$  is read in the query tape, a test mass  $z'$  is simultaneously placed in the left pan such that  $z - \epsilon \leq z' \leq z + \epsilon$ . In the last two cases,  $z'$  is assumed to be an independent random variable, with a uniform distribution on the error interval. In what follows,  $\text{Mass}(m\downarrow_\ell)$  denotes the action that triggers the BBE experiment with mass  $m\downarrow_\ell$ . Depending on the context, the experiment is performed either with infinite, unbounded or finite precision, as explained above.

---

<sup>5</sup> This expression for the time, specifically exhibiting an exponential growth on the precision of  $z$  with respect to the unknown  $y$ , is typical in physical experiments, regardless of the concept being measured. The constant  $\tau$  depends on the geometry of the lever, the value of  $h$  and the acceleration of gravity  $g$ .

**Definition 3.1.** We say that a set  $A$  is decidable by an infinite precision BBE machine in polynomial time if there is an oracle Turing machine  $\mathcal{M}$ , an unknown mass  $y$  and a time schedule  $T$  such that  $\mathcal{M}$  decides  $A$  and runs in polynomial time. We say that a set  $A$  is decidable by an unbounded (or fixed) precision BBE machine in polynomial time if there is an oracle Turing machine  $\mathcal{M}$  running in polynomial time, an unknown mass  $y$ , a time schedule  $T$ , and some  $0 < \gamma < 1/2$  such that, for any input word  $w$ , (a) if  $w \in A$ , then  $\mathcal{M}$  accepts  $w$  with probability at least  $1 - \gamma$  and (b) if  $w \notin A$ , then  $\mathcal{M}$  rejects  $w$  with probability at least  $1 - \gamma$ .

<p><b>ALGORITHM “BINARY SEARCH” :</b></p> <p><b>Input</b> number <math>\ell \in \mathbb{N}</math>; % number of places to the right of the left leading 0  <math>x_0 := 0</math>; <math>m := 0</math>, <math>x_1 := 1</math>;  <b>While</b> <math>x_1 - x_0 &gt; 2^{-\ell}</math> <b>Do Begin</b>  <math>m := (x_0 + x_1)/2</math>;  <math>s := \text{Mass}(m _{\ell})</math>; % Proc. Mass is either deter. or stoch. and takes time <math>T(\ell)</math>  <b>If</b> <math>s = \text{‘YES’}</math> <b>Then</b> <math>x_1 := m</math> <b>Else</b> <math>x_0 := m</math>;  <b>End While</b>;  <b>Output</b> <math>x_0</math>.</p>
---

**Fig. 2.** The experimental procedure Mass takes the scheduled time  $T(\ell)$ , where  $\ell$  is the size of the query and  $T$  an arbitrary time constructible function

<p><b>ALGORITHM “SEARCH(<math>\epsilon, h</math>)” :</b></p> <p><b>Input</b> number <math>\ell \in \mathbb{N}</math>; % number of places to the right of the left leading 0  <math>c := 0</math>; <math>\zeta := 2^{2\ell+h}</math>; % <math>h</math> is used to bound the probability of error  <b>Repeat</b> <math>\zeta</math> times  <math>s := \text{Mass}(1 _{\ell})</math>; % Recall that this step takes <math>T(\ell)</math> units of time  <b>If</b> <math>s = \text{‘YES’}</math> <b>Then</b> <math>c := c + 1</math>;  <b>End Repeat</b>;  <b>Output</b> <math>c/\zeta</math>.</p>
--

**Fig. 3.** The experimental procedure Mass is stochastic for the fixed precision case

**Proposition 3.1.** (1) Let  $s$  be the result of  $\text{Mass}(m)$  in Figures 2 and 3, for an unknown mass  $y$  and time schedule  $T$ . In the infinite precision scenario, (a) if  $s = \text{‘YES’}$ , then  $y < m$  and (b) if  $s = \text{‘TIMEOUT’}$ , then  $y > m - (\tau/T(|m|))^2$ . In the unbounded precision scenario, (a) if the oracle  $\text{Mass}(m)$  answers ‘YES’, then  $y < m + 2^{-|m|}$  and (b) if the oracle  $\text{Mass}(m)$  answers ‘TIMEOUT’, then  $y > m - 2^{-|m|} - (\tau/T(|m|))^2$ . (2) For any unknown mass  $y$  and any time schedule  $T$ , (a) the time complexity of algorithm of Figure 2, both in the infinite and unbounded precision scenarios, for input  $\ell$ , is  $\mathcal{O}(\ell T(\ell))$ , (b1) in the infinite precision case, for all  $k \in \mathbb{N}$ , there exists  $\ell \in \mathbb{N}$  such that  $T(\ell) \geq \tau 2^{k/2}$  and the

output is a dyadic rational  $m$  such that  $|y - m| < 2^{-k}$ , (b2) in the unbounded precision case, for all  $k \in \mathbb{N}$ , there exists  $\ell \in \mathbb{N}$  such that  $\ell \geq k + 1$  and  $T(\ell) \geq \tau 2^{(k+1)/2}$  and the output is a dyadic rational  $m$  such that  $|y - m| < 2^{-k}$ , and (c) moreover, both in the infinite and unbounded precision scenarios,  $\ell$  is at most exponential in  $k$  and, if  $T(k)$  is exponential in  $k$ , then the value of  $\ell$  witnessing (b) can be taken to be linear in  $k$ . **(3)** For all  $s \in (0, 1)$ ,  $\epsilon \in (0, 1/2)$ ,  $h \in \mathbb{N}$ , and time schedule  $T$ , (a) the time complexity of algorithm of Figure 3 for input  $\ell$  is  $\mathcal{O}(2^{2\ell}T(\ell))$ , (b) for all  $k \in \mathbb{N}$  there exists  $\ell \in \mathbb{N}$  such that  $T(\ell) > \tau 2^{(k+1)/2} / \sqrt{2\epsilon}$  and thus, with probability of error  $2^{-h}$ , the output of the algorithm is a dyadic rational  $m$  such that  $|s - m| < 2^{-\ell}$ , and (c) if  $T(k)$  is exponential in  $k$ , then the value of  $\ell$  witnessing the above proposition is linear in  $k$ .

In both cases of *unbounded* and *finite precision*, the experiment becomes probabilistic and we can use it to simulate independent coin tosses and to produce random strings. We can state (see [13,2]):

**Proposition 3.2.** **(1)** For all unknown mass  $y$  and all time schedule  $T$  there is a dyadic rational  $z$  and a real number  $\delta \in (0, 1)$  such that the result of  $\text{Mass}(z)$  is a random variable that produces ‘YES’ with probability  $\delta$  and ‘TIMEOUT’ with probability  $1 - \delta$ . **(2)** Take a biased coin with probability of heads  $\delta \in (0, 1)$  and let  $\gamma \in (0, 1/2)$ . Then there is an integer  $N$  such that, with probability of failure at most  $\gamma$ , we can use a sequence of independent biased coin tosses of length  $Nn$  to produce a sequence of length  $n$  of independent fair coin tosses.

## 4 Lower Bounds on the BBE Machine

We encode an advice function  $(f : \mathbb{N} \rightarrow \{0, 1\}^*) \in \text{log}\star$  into a real number  $\mu(f) \in (0, 1)$  by replacing every 0 by 100, every 1 by 010 and adding 001 at the end of the codes for  $f(2^k)$ , with  $k \in \mathbb{N}$  (see Section 6(c) of [2]). These numbers belong to the Cantor set  $\mathcal{C}_3$ .

**Theorem 4.1.** If  $A \in P/\text{log}\star$ , then  $A$  is decidable by a BBE machine with infinite precision in polynomial time.

*Proof:* Let  $f$  be a prefix function in  $\text{log}\star$  and  $\mathcal{M}'$  be a Turing machine running on polynomial time such that, for any natural number  $n$  and any word  $w$  such that  $|w| \leq n$ ,  $w \in A$  iff  $\mathcal{M}'$  accepts  $\langle w, f(n) \rangle$ . Let  $y = \mu(f)$  and  $T$  any exponential time schedule. Since  $f \in \text{log}$ , there are constants  $a, b \in \mathbb{N}$  such that, for all  $n$ ,  $|f(n)| \leq a \lceil \log(n) \rceil + b$ . For each  $n \in \mathbb{N}$ , let  $k_n = 3(a + 1) \lceil \log(n) \rceil + 3b + 8$ . Resorting to Proposition 3.1 (2) (b1) and (c), there is a value of  $\ell$ , linear in  $k_n$  (and thus linear in  $\lceil \log(n) \rceil$ ), such that  $T(\ell) > \tau 2^{k_n/2}$  and so the result of running the algorithm of Figure 2 for input  $\ell$  is a dyadic rational  $m$  such that  $|y - m| < 2^{-k_n}$ . Then, by Cantor  $\mathcal{C}_3$  properties,<sup>6</sup>  $m$  and  $y$  coincide in the first

<sup>6</sup> For every  $x \in \mathcal{C}_3$  and for every dyadic rational  $z \in (0, 1)$  with size  $|z| = m$ , (a) if  $|x - z| \leq 1/2^{m+5}$ , then the binary expansions of  $x$  and  $z$  coincide in the first  $i$  bits and (b)  $|x - z| > 1/2^{m+10}$ .



$k_n - 5 = 3(a + 1)\lceil \log(n) \rceil + 3(b + 1)$  bits, which means that  $m$  can be used to decode  $f(2^{\lceil \log(n) \rceil})$ . The oracle machine  $\mathcal{M}$  that reads the dyadic  $m$  and then simulates  $\mathcal{M}'$  for the input word  $\langle w, f(2^{\lceil \log(n) \rceil}) \rangle$  decides  $A$ . Furthermore, from Proposition 3.1 (2) (a) and the fact that  $A \in P/\log\star$ , the time complexity of these activities is polynomial in  $n$ .  $\square$

**Theorem 4.2.** *If  $A \in BPP//\log\star$ , then (a)  $A$  is decidable by a BBE machine with unbounded precision in polynomial time and (b)  $A$  is decidable by a BBE machine with fixed precision  $\epsilon \in (0, 1/2)$  in polynomial time.*

*Proof:* Herein we prove (a) and leave (b) to the full paper. Let  $\mathcal{M}'$  be the advice Turing machine working in polynomial time  $p_3$ ,  $f \in \log\star$  the prefix function and  $\gamma_3 \in (0, 1/2)$  the constant witnessing that  $A \in BPP//\log\star$ . Let  $a, b \in \mathbb{N}$  be such that, for all  $n$ ,  $|f(n)| \leq a\lceil \log(n) \rceil + b$ . Let  $\gamma_2$  be such that  $\gamma_3 + \gamma_2 < 1/2$ . Let  $y = \mu(f)$  and consider any exponential time schedule  $T$ . By Proposition 3.2 (1), there is a dyadic rational  $z$  that can be used to produce independent coin tosses with probability of heads  $\delta \in (0, 1)$ . This rational depends only on  $y$  and  $T$  and can be hard-wired into the machine. By Proposition 3.2 (2), we can take an integer  $N$  (depending on  $\delta$  and  $\gamma_2$ ) such that we can use  $Nn$  biased coin tosses to simulate  $n$  fair coin tosses, with probability of failure at most  $\gamma_2$ . For each  $n \in \mathbb{N}$ , let  $k_n = 3(a + 1)\lceil \log(n) \rceil + 3b + 8$ . By Proposition 3.1 (2) (b2) and (c), there is  $\ell$ , linear in  $k_n$ , such that the result of the algorithm of Figure 2 in the case of unbounded precision, for input  $\ell$ , is a dyadic rational  $m$  such that  $|y - m| < 2^{-k_n}$ , so that, by the Cantor set properties,  $m$  can be used to decode  $f(2^{\lceil \log(n) \rceil})$ . We design a oracle machine  $\mathcal{M}$  that, on input  $w$  of size  $n$ , starts by running Binary Search for input  $\ell$  and then uses the result to decode the advice  $\mu(f)$ . In the next step, the machine uses the dyadic rational  $z$  to produce a sequence of  $Np_3(n)$  independent biased coin tosses and extract from it a new sequence of  $p_3(n)$  independent fair coin tosses. If it fails (which may happen with probability at most  $\gamma_2$ ), then the machine rejects  $w$ . Otherwise the machine simulates  $\mathcal{M}'$  on input  $\langle w, f(2^{\lceil \log(n) \rceil}) \rangle$  using the sequence of  $p_3(n)$  fair coin tosses to decide the path of the computation of  $\mathcal{M}'$ . The machine  $\mathcal{M}$  decides  $A$  in polynomial time. If  $w \in A$ , then  $\mathcal{M}$  rejects  $w$  if it failed to produce the sequence of fair coin tosses or if  $\mathcal{M}'$  rejected  $w$ . The probability of rejecting  $w$  is bounded by  $\gamma_2 + \gamma_3$ . On the other hand, if  $w \notin A$ , then  $\mathcal{M}$  accepts  $w$  if it produced a sequence of fair coin tosses and if  $\mathcal{M}'$  accepted  $w$ , and this happens with probability at most  $\gamma_3$ . This means that the error probability of  $\mathcal{M}$  is bounded by constant  $\gamma_2 + \gamma_3$  which is less than  $1/2$ . By Proposition 3.1 (2) (a), the time complexity of the first step is  $\mathcal{O}(\ell T(\ell))$ . Since  $\ell$  is logarithmic in  $n$  and  $T$  is exponential in  $\ell$ , the result is bounded by some polynomial in  $n$ ,  $p_1(n)$ . The time complexity of the second step is also bounded by some polynomial  $p_2$  in  $n$ , since we require only a polynomial amount of  $Np_3(n)$  biased coin tosses. Finally, since  $\mathcal{M}'$  runs in polynomial time  $p_3$ , we conclude that  $\mathcal{M}$  runs in polynomial time  $\mathcal{O}(p_1 + p_2 + p_3)$ .  $\square$

## 5 Upper Bounds on the BBE Machine

### 5.1 $P/\log^\star$ is an Upper Bound for the Infinite Precision Case

We introduce a sequence of real numbers called *boundary numbers*. These are defined in terms of the time  $T_{exp}(z, y)$  taken by the experiment for test value  $z$  and unknown value  $y$  (see the timing Equation (1) in 2.3 for an example of this), and the time schedule  $T : \mathbb{N} \rightarrow \mathbb{N}$  which used to determine the output ‘TIMEOUT’.

**Definition 5.1.** *Let  $y \in (0, 1)$  be the unknown mass and  $T$  a time schedule. Then, for all  $k \in \mathbb{N}$ , we define  $w_k \in (0, 1)$  as the number such that  $T_{exp}(w_k, y) = T(k)$ . We also define  $z_k$  as  $z_k = w_k \downarrow_k$ .*

For any oracle query  $z$  of size  $k$ , (a) if  $z \leq z_k$ ,<sup>7</sup> then the result of the experiment is ‘TIMEOUT’ and (b) if  $z > z_k$ , then the result of the experiment is ‘YES’. Notice that  $z_k$  is precisely the result of the algorithm for input  $k$  and as such, by knowing  $z_k$ , we can obtain the result of any experiment of size  $k$  (in the infinite precision case) without having to perform it. This is the core idea of the two following proofs.

**Theorem 5.1.** *If  $A$  is a set decidable by a BBE machine with infinite precision in polynomial time and the chosen time schedule is exponential, then  $A \in P/\log^2 \star$ .*

*Proof:* Suppose that  $A$  is decided by a BBE machine  $\mathcal{M}$  in polynomial time, with exponential time schedule  $T$ . Since  $T$  is exponential and the running time is polynomial, we conclude that the size of the oracle query grows at most logarithmically in the size of the input word, i.e., there are constants  $a, b \in \mathbb{N}$  such that, for any input word of size  $n$ , the computation of  $\mathcal{M}$  only queries words with size less than or equal to  $a \lceil \log(n) \rceil + b$ . Consider the advice function  $f$  such that  $f(n)$  encodes the word  $z_1 \# z_2 \# \dots \# z_t$ , where  $t = a \lceil \log(n) \rceil + b$ . We observe that  $f$  is a prefix function and  $|f(n)| \in \mathcal{O}(t - 1 + \sum_{i=1}^t i) = \mathcal{O}(t^2) = \mathcal{O}(\log^2(n))$ . Furthermore, we can use  $f(n)$  to determine the answer to any possible oracle query of size less than  $a \lceil \log(n) \rceil + b$ . To decide the set  $A$  in polynomial time with advice  $f$ , simply simulate the original machine  $\mathcal{M}$  on the input word and, whenever  $\mathcal{M}$  is in the query state, simulate the experiment by comparing the query word with the appropriate  $z_i$  in the advice function. As this comparison can be done in polynomial time and  $\mathcal{M}$  runs in polynomial time too, we conclude that  $A$  can be decided in polynomial time with the given advice.  $\square$

Observe that  $w_k \searrow y$ , where  $y$  is the unknown mass. As we are going to see, under some extra assumptions on the time schedule, the value of  $z_{k+1}$  can be obtained by adding to the word  $z_k$  a very few bits of information, shortening the encoding to  $\mathcal{O}(\log(n))$  bits.

---

<sup>7</sup> This comparison can be seen either as a comparison between reals — the mass values —, or as a comparison between binary strings in the lexicographical order — the corresponding dyadic rationals.

**Theorem 5.2.** *If  $A$  is a set decidable by a BBE machine with infinite precision in polynomial time and the chosen time schedule is  $T(k) \in \Omega(2^{k/2})$ ,<sup>8</sup> then  $A \in P/\log\star$ .*

*Proof:* Since  $T(k) \in \Omega(2^{k/2})$ , it follows that there exist constants  $\sigma, k_0 \in \mathbb{N}$  such that  $T(k) \geq \sigma 2^{k/2}$ , for  $k \geq k_0$ . By Proposition 3.1 (2) (b1) and (c), we can ensure that the value of the boundary number  $w_k$  is such that  $y < w_k < y + 2^{-k+c}$ , for some constant  $c \in \mathbb{N}$  and for  $k > k_0$ . This means that, when we increase the size of  $k$  by one bit, we also increase the precision on  $y$  by one bit. Let us write the dyadic rational  $z_k$  as the concatenation of two strings,  $z_k = x_k \cdot y_k$ , where  $y_k$  has size  $c$  and  $x_k$  has size  $k - c$ . Note that  $w_k - 2^{-k+c} < x_k < w_k$ , i.e.  $|x_k - y| < 2^{-k+c}$ . The bits of  $x_k$  provide information about the possibilities for the binary expansion of  $y$ . We show that we can obtain  $x_{k+1}$  from  $x_k$  with just two more bits of information. Suppose that  $x_k$  ends with the sequence  $x_k = \dots 10^\ell$ . The only two possibilities for the first  $k - c$  bits of  $y$  are  $\dots 10^\ell$  or  $\dots 01^\ell$ . Thus,  $x_{k+1}$  must end in one of the following:  $x_{k+1} = \dots 10^\ell 1$  or  $x_{k+1} = \dots 10^\ell 0$  or  $x_{k+1} = \dots 01^{\ell+1}$  or  $x_{k+1} = \dots 01^\ell 0$ . That is, *even though  $x_k$  is not necessarily a prefix of  $x_{k+1}$* , it still can be obtained from  $x_k$  by appending some information that determines which of the four possibilities occur. We define the function  $f(n)$  as follows: (a) if  $n < k_0$ , then  $f(n) = z_1 \# z_2 \# \dots \# z_n$ , (b)  $f(k_0) = f(k_0 - 1) \# x_{k_0} \# \# y_{k_0}$ , and (c) if  $n > k_0$ , then  $f(n) = f(n - 1) \# \# b_1 b_2 y_n$ , where the bits  $b_1 b_2$  are used to determine one of the four possibilities for  $x_n$  with respect to  $x_{n-1}$ . Observe also that from  $f(n)$  one can recover the values of  $z_k$ , for all  $k \leq n$ . Moreover,  $|f(n)|$  is linear in  $n$ , since all  $y_k$  have size  $d$ . Since  $A$  is decided by a BBE machine  $\mathcal{M}$  in polynomial time and  $T$  is exponential, the size of the oracle query grows at most logarithmically in the size of the input word. There are constants  $d, e \in \mathbb{N}$  such that, for any input word of size  $n$ , the computation of  $\mathcal{M}$  only queries for words with size less than or equal to  $d \lceil \log(n) \rceil + e$ . We define the advice function  $g : \mathbb{N} \rightarrow \{0, 1\}^*$  such that  $g(n) = f(d \lceil \log(n) \rceil + e)$ . Note that  $|g(n)| = \mathcal{O}(\log(n))$  and  $g(n)$  can be used to determine the result of any oracle query for any computation for any input word of size less than or equal to  $n$ . Then, as in the proof of Theorem 5.1, we can devise a Turing machine that decides  $A$  in polynomial time using  $g$  as advice, witnessing that  $A \in P/\log\star$ .  $\square$

## 5.2 $BPP//\log^2\star$ Is an Upper Bound for the Unbounded Precision Case

Our next step is to prove that any set decidable using a BBE machine with unbounded precision in polynomial time can also be decided in polynomial time using an advice of a particular size. Given a BBE machine  $\mathcal{M}$ , we construct an advice function  $f$  with the following properties: (a) for any  $n$ ,  $f(n)$  contains enough information to answer all queries occurring during the computation of  $\mathcal{M}$  on a word of size  $n$  and (b) the size of  $f(n)$  grows as slowly as we can accomplish.

<sup>8</sup> We define  $\Omega(g)$  as the class of functions  $f$  such that there exist  $p \in \mathbb{N}$  and  $r \in \mathbb{R}^+$  such that, for all  $n \geq p$ ,  $f(n) \geq rg(n)$ .

In the previous section, we made the observation that a dyadic rational  $z_n$  of size  $n$  could be used to answer all oracle queries of size up to  $n$ . Thus, using an exponential time schedule, we could simulate any polynomial time computation having the oracle replaced by an advice containing a logarithmic number of  $z_n$ 's. Given a threshold oracle (that is, an oracle with two possible random answers), we can depict the sequence of the answers in a binary tree, where each path is labelled with its probability. The leaves of these trees are marked with an accept or reject. Then, to get the probability of acceptance of a particular word, we simply add the probabilities for each path that ends in acceptance. The next basic idea is to think of what would happen if we change the probabilities in the tree. This means that we are using the same procedure of the Turing machine, but now with a different probabilistic oracle. Suppose that the tree has depth  $t$  and there is a real number  $\rho$  that bounds the difference in the probabilities labelling all pairs of corresponding edges in the two trees. Proposition 2.1 of [4] states that the difference in the probabilities of acceptance of the two trees is at most  $2t\rho$ . (Motivation from the automata theory comes from von Neumann's article [13].)

Recall the sequence of real numbers  $w_k$  such that  $w_k$  is the solution to the equation  $T_{exp}(w_k, y) = T(k)$ . This means that  $w_k$  is the mass in which the time taken for the experiment to return a value equals the time scheduled for an experiment with a query of size  $k$ . The numbers  $w_k$  verify two important properties. First, if we round down  $w_k$  to the first  $k$  bits, we get  $z_k$ . That is,  $z_k \leq w_k < z_k + 2^{-k}$ . Remember that  $z_k$  is the result of the algorithm of Figure 2 for input  $k$ , or alternatively, is the biggest dyadic rational of size  $k$  for which the result of the experiment is 'TIMEOUT' (see Proposition 3.1 (1)). The second property is that, when performing the experiment  $\text{Mass}(z_k)$  in Figure 2, since the mass  $z'$  is uniformly sampled from the interval  $(z_k - 2^{-k}, z_k + 2^{-k})$ , the probability of obtaining result 'YES' is precisely  $(z_k + 2^{-k} - w_k)/(2 \times 2^{-k}) = 1/2 - (w_k - z_k)/(2 \times 2^{-k})$ . From these facts we can conclude that, if we know the first  $k + d$  bits of  $w_k$ , then we can obtain an approximation of the probability of answer 'YES' when performing experiment  $\text{Mass}(z_k)$  with an error of at most  $2^{-d}$ . The same reasoning can be made for the experiment  $\text{Mass}(z_k + 2^{-k})$ , which is the other dyadic rational of size  $k$  for which the experiment is not deterministic. In this case, the probability of answer 'YES' is  $1 - (w_k - z_k)/(2 \times 2^{-k})$ , and this value can also be approximated by knowing the first  $k + d$  bits of  $w_k$ , with an error of at most  $2^{-d}$ . We state without proof the theorem of this section:<sup>9</sup>

**Theorem 5.3.** *If  $A$  is a set decided by a BBE machine in polynomial time with unbounded precision and exponential time schedule  $T$ , then  $A \in BPP // \log^2 \star$ .*

### 5.3 $BPP // \log^2 \star$ Is an Upper Bound for the Finite Precision Case

We now establish an upper bound for the class of sets decided by BBE machines with finite precision in polynomial time. Theorem 5.4 has a proof that follows

---

<sup>9</sup> The proof of this theorem and of Theorem 5.4 can be found in the full paper on threshold oracles, available on demand.

the same lines of the proof of Theorem 5.3. We discuss now how the bits of the probability distribution can be computed. The numbers  $w_k$  are defined as in the beginning of Section 5.2. The following proposition is straightforward:

**Proposition 5.1.** *For any dyadic rational  $z$  of size  $k$  let  $P(z)$  be the probability of obtaining answer ‘YES’ when performing the experiment with test mass  $z$ , unknown mass  $y$ , finite precision  $\epsilon$ , and time schedule  $T$ .*

$$P(z) = \begin{cases} 0 & \text{if } z < w_k - \epsilon \\ \frac{1}{2} + \frac{z-w_k}{2\epsilon} & \text{if } w_k - \epsilon \leq z < w_k + \epsilon \\ 1 & \text{if } w_k + \epsilon \leq z \end{cases}$$

Our advice function will contain dyadic rational approximations of  $w_k$  and  $\epsilon$  that will be used to compute approximations to  $P(z)$  up to  $2^{-e}$ , for some  $e \in \mathbb{N}$  and for any dyadic rational  $z$  of size  $k$ . Let  $d$  be an integer such that  $2^{-d} \leq \epsilon$ , and let  $w'_k$  and  $\epsilon'$  be  $w_k$  and  $\epsilon$  rounded up to the first  $d + e$  and  $d + e + 1$  bits, respectively. We can then compute  $(z - w_k)/2$  with precision  $2^{-d-e-1}$  and  $P(z)$  with error less than  $2^{-e}$ . The number of digits required grows linearly with the precision desired on  $P(z)$  that in its turn increases logarithmically with the size of the input word. We conclude that, for queries of size less than or equal to that of  $z$ , only a logarithmic amount of bits of  $P(z)$  is required. Again, we state without proof:

**Theorem 5.4.** *If  $A$  is a set decided by a BBE machine in polynomial time with fixed precision and an exponential time schedule, then  $A \in BPP//\log^2 \star$ .*

## 6 Conclusions

We have introduced methods to study the computational power of threshold systems, such as the neurone or photoelectric cells, for which quantities can only be measured either from below or from above. We showed that Turing machines equipped with threshold oracles in polynomial time have a computational power between  $P/\log \star$  and  $BPP//\log^2 \star$ , no matter whether the precision is infinite, unbounded or fixed. We expect that hybrid systems in general cannot transcend such computational power and that this computational power stands to hybrid systems as the Church-Turing thesis is to Turing machines. Our result weakens the claims of other classes associated with models of physical systems (see, e.g.,  $P/\text{poly}$  in [12]). In studying two-sided experiments (as in [6]), we saw that an oracle answer such as ‘LEFT’ would imply that  $z < y$  and an oracle answer such as ‘RIGHT’ would imply that  $z > y$ , where  $y$  is the unknown mass and  $z$  the test mass. In a threshold experiment, we saw that the oracle answer ‘YES’ would imply that  $z > y$ . However, there exists yet another type of physical experiment, the *vanishing experiment*, in which the answer ‘YES’ implies only that  $z \neq y$ . An example is the determination of Brewster’s angle in Optics: in the lab measurement of the critical angle of incidence of a monochromatic light ray into the surface of separation of two media such that there is a transmitted ray but no reflected ray. Vanishing experiments are a new type of measurement to

investigate. We think that our model captures (i) the complexity of measurement and the limits to computational power of hybrid systems and (ii) the limits of what can be measured (such as in [6]). Reactions towards a gedankenexperiment, such as measuring mass as in Section 2, as an oracle can express dissatisfaction as such idealized devices cannot be built. Unfortunately, there seems to be a diffuse philosophy that considers the Turing machine an object of a different kind. Clearly, both the abstract physical experiment and the Turing machine are gedankenexperiments and non-realizable. To implement a Turing machine the engineer would need either unbounded space and an unlimited physical support structure, or unbounded precision in some finite interval to code for the contents of the tape. However, the experiment can be set up to some degree of precision in the same way that the Turing machine can be implemented up to some degree accuracy. Knowing that both objects, the Turing machine and the measurement device, are of the same ideal nature, we argue that the models allow us to study the power of adding real numbers to computing devices, characteristic of hybrid machines, and the limits of what can be measured.

**Acknowledgements.** The research of José Félix Costa and Diogo Poças is supported by FCT PEst – OE/MAT/UI0209/2011.

## References

1. Balcázar, J.L., Días, J., Gabarró, J.: Structural Complexity I, 2nd edn. Springer (1988, 1995)
2. Beggs, E., Costa, J.F., Loff, B., Tucker, J.V.: Computational complexity with experiments as oracles. *Proceedings of the Royal Society, Series A (Mathematical, Physical and Engineering Sciences)* 464(2098), 2777–2801 (2008)
3. Beggs, E., Costa, J.F., Loff, B., Tucker, J.V.: On the complexity of measurement in classical physics. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 20–30. Springer, Heidelberg (2008)
4. Beggs, E., Costa, J.F., Loff, B., Tucker, J.V.: Computational complexity with experiments as oracles II. Upper bounds. *Proceedings of the Royal Society, Series A (Mathematical, Physical and Engineering Sciences)* 465(2105), 1453–1465 (2009)
5. Beggs, E., Costa, J.F., Tucker, J.V.: Computational Models of Measurement and Hempel’s Axiomatization. In: Carsetti, A. (ed.) *Causality, Meaningful Complexity and Knowledge Construction. Theory and Decision Library A*, vol. 46, pp. 155–184. Springer (2010)
6. Beggs, E., Costa, J.F., Tucker, J.V.: Limits to measurement in experiments governed by algorithms. *Mathematical Structures in Computer Science* 20(06), 1019–1050 (2010) Special issue on Quantum Algorithms, Venegas-Andraca, S.E. (ed.)
7. Beggs, E., Costa, J.F., Tucker, J.V.: Axiomatizing physical experiments as oracles to algorithms. *Philosophical Transactions of the Royal Society, Series A (Mathematical, Physical and Engineering Sciences)* 370(12), 3359–3384 (2012)
8. Beggs, E., Costa, J.F., Tucker, J.V.: The impact of models of a physical oracle on computational power. *Mathematical Structures in Computer Science* 22(5), 853–879 (2012), Special issue on Computability of the Physical, Calude, C.S., Barry Cooper, S. (eds.)

9. Hempel, C.G.: Fundamentals of concept formation in empirical science. *International Encyclopedia of Unified Science* 2(7) (1952)
10. Jain, S., Osherson, D.N., Royer, J.S., Sharma, A.: *Systems That Learn. An Introduction to Learning Theory*, 2nd edn. The MIT Press (1999)
11. Krantz, D.H., Suppes, P., Duncan Luce, R., Tversky, A.: *Foundations of Measurement*. Dover (2009)
12. Siegelmann, H.T.: *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser (1999)
13. von Neumann, J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: *Automata Studies*, pp. 43–98. Princeton University Press (1956)

# Size Lower Bounds for Quantum Automata<sup>\*</sup>

Maria Paola Bianchi, Carlo Mereghetti, and Beatrice Palano

Dip. Informatica, Univ. degli Studi di Milano, v. Comelico 39, 20135 Milano, Italy  
{bianchi,mereghetti,palano}@di.unimi.it

**Abstract.** We compare the descriptonal power of quantum finite automata with control language (QFCs) and deterministic finite automata (DFAs). By suitably adapting Rabin’s technique, we show how to convert any given QFC to an equivalent DFA, incurring in an at most exponential size increase. This enables us to state a lower bound on the size of QFCs, which is logarithmic in the size of equivalent minimal DFAs. In turn, this result yields analogous size lower bounds for several models of quantum finite automata in the literature.

**Keywords:** quantum finite automata, descriptonal complexity.

## 1 Introduction

While we can hardly expect to see a full-featured quantum computer in the near future, it is reasonable to envision classical computing devices incorporating small quantum components. Since the physical realization of quantum systems has proved to be a complex task, it is reasonable to keep quantum components as “small” as possible. Thus, it is well worth investigating, from a theoretical point of view, *lower limits* to the size of quantum devices when performing certain tasks, also emphasizing *trade-offs* with the size of equivalent classical devices.

Small size quantum devices are modeled by *quantum finite automata* (QFAs), a theoretical model for quantum machines with finite memory. Originally, two models of QFAs are proposed: *measure-once* QFAs (MO-QFAs) [8,14], where the probability of accepting words is evaluated by “observing” just once, at the end of input processing, and *measure-many* QFAs (MM-QFAs) [2,13], having such an observation performed after each move. Results in the literature (see, e.g., [4] for a survey) show that MO-QFAs are strictly less powerful than MM-QFAs which, in turn, are strictly less powerful than classical (deterministic or probabilistic) automata. Several modifications to these two original models of QFAs are then proposed, in order to tune computational power and motivated by different possible physical realizations. Thus, e.g., enhanced [16], reversible [9], Latvian [1] QFAs, and QFAs with quantum and classical states [21] are introduced.

Along this line of research, the model of *quantum finite automata with control language* (QFCs) is proposed in [4], as a hybrid system featuring both a quantum

---

<sup>\*</sup> Partially supported by MIUR under the project “PRIN: Automi e Linguaggi Formali: Aspetti Matematici e Applicativi.”



and a classical component. In [4,15], it is proved that the class of languages accepted with isolated cut point by QFCs coincides with regular languages, and that QFCs can be exponentially smaller than equivalent classical automata.

A relevant feature of QFCs, of interest in this paper, is that they can naturally and directly simulate several models of QFAs by preserving the size. This property makes QFCs a general unifying framework within which to investigate size results for different quantum paradigms: size lower bounds or size trade-offs proved for QFCs may directly apply to simulated types of QFAs as well. In fact, the need for a general quantum framework is witnessed by several results in the literature (see, e.g., [2,3,5,7]), showing that QFAs can be exponentially more succinct than equivalent classical automata, by means of techniques which are typically targeted on the particular type of QFA and not easily adaptable to other paradigms. So, to cope with this specialization problem, here we study size lower bounds and trade-offs for QFCs.

After introducing some basic notions, we show in Section 3 how to build from a given QFC an equivalent DFA. To this aim, we must suitably modify classical Rabin's technique [17], since the equivalence relation we choose to define the state set of the DFA is not a congruence. On the other hand, this relation – based on the classical Euclidean norm – allows us to directly estimate the cost of the conversion  $\text{QFC} \rightarrow \text{DFA}$  by a geometrical argument on compact spaces. We obtain that the size of the resulting DFA is at most exponentially larger than the size of the QFC. Stated in other terms in Section 4, this latter result directly implies that QFCs are at most exponentially more succinct than classical equivalent devices. Indeed, due to QFCs generality, this succinctness result carries over other models of QFAs, such as MO-QFAs, MM-QFAs, and reversible QFAs.

## 2 Preliminaries

We quickly recall some notions of linear algebra, useful to describe the quantum world. For more details, we refer the reader to, e.g., [12,19]. The fields of real and complex numbers are denoted by  $\mathbb{R}$  and  $\mathbb{C}$ , respectively. Given a complex number  $z = a + ib$ , we denote its *real part*, *conjugate*, and *modulus* by  $z_R = a$ ,  $z^* = a - ib$ , and  $|z| = \sqrt{zz^*}$ , respectively. We let  $\mathbb{C}^{n \times m}$  denote the set of  $n \times m$  matrices with entries in  $\mathbb{C}$ . Given a matrix  $M \in \mathbb{C}^{n \times m}$ , for  $1 \leq i \leq n$  and  $1 \leq j \leq m$ , we let  $M_{ij}$  denote its  $(i, j)$ th entry. The *transpose* of  $M$  is the matrix  $M^T \in \mathbb{C}^{m \times n}$  satisfying  $M^T_{ij} = M_{ji}$ , while we let  $M^*$  be the matrix satisfying  $M^*_{ij} = (M_{ij})^*$ . The *adjoint* of  $M$  is the matrix  $M^\dagger = (M^T)^*$ .

For matrices  $A, B \in \mathbb{C}^{n \times m}$ , their *sum* is the  $n \times m$  matrix  $(A+B)_{ij} = A_{ij} + B_{ij}$ . For matrices  $C \in \mathbb{C}^{n \times m}$  and  $D \in \mathbb{C}^{m \times r}$ , their *product* is the  $n \times r$  matrix  $(CD)_{ij} = \sum_{k=1}^m C_{ik}D_{kj}$ . For matrices  $A \in \mathbb{C}^{n \times m}$  and  $B \in \mathbb{C}^{p \times q}$ , their *Kronecker (or tensor) product* is the  $np \times mq$  matrix defined as

$$A \otimes B = \begin{pmatrix} A_{11}B & \cdots & A_{1m}B \\ \vdots & \ddots & \vdots \\ A_{n1}B & \cdots & A_{nm}B \end{pmatrix}.$$

When operations can be performed, we have that  $(A \otimes B) \cdot (C \otimes D) = AC \otimes BD$ . A *Hilbert space* of dimension  $n$  is the linear space  $\mathbb{C}^{1 \times n}$  of  $n$ -dimensional complex row vectors equipped with sum and product by elements in  $\mathbb{C}$ , in which the *inner product*  $\langle \varphi, \psi \rangle = \varphi \psi^\dagger$  is defined. From now on, for the sake of simplicity, we will write  $\mathbb{C}^n$  instead of  $\mathbb{C}^{1 \times n}$ . The *norm* of a vector  $\varphi \in \mathbb{C}^n$  is given by  $\|\varphi\| = \sqrt{\langle \varphi, \varphi \rangle}$ . We recall the following properties, for  $\varphi, \psi, \xi, \zeta \in \mathbb{C}^n$  and  $r \in \mathbb{R}$ :

$$\begin{aligned} \langle \varphi, \psi \rangle &= \langle \psi, \varphi \rangle^* = \langle \psi^*, \varphi^* \rangle, & \langle \varphi + \psi, \xi \rangle &= \langle \varphi, \xi \rangle + \langle \psi, \xi \rangle, \\ \langle r\varphi, \psi \rangle &= r\langle \varphi, \psi \rangle = \langle \varphi, r\psi \rangle, & \langle \varphi \otimes \psi, \xi \otimes \zeta \rangle &= \langle \varphi, \xi \rangle \langle \psi, \zeta \rangle, \\ \|\varphi - \psi\|^2 &= \|\varphi\|^2 + \|\psi\|^2 - 2\langle \varphi, \psi \rangle_{\mathbb{R}}, & \|\varphi \otimes \psi\| &= \|\varphi\| \|\psi\|. \end{aligned}$$

The *angle between complex vectors*  $\varphi$  and  $\psi$  is defined as (see, e.g., [18]):

$$\text{ang}(\varphi, \psi) = \arccos \frac{\langle \varphi, \psi \rangle_{\mathbb{R}}}{\|\varphi\| \|\psi\|}.$$

If  $\langle \varphi, \psi \rangle = 0$ , we say that  $\varphi$  is *orthogonal* to  $\psi$ . Two subspaces  $X, Y \subseteq \mathbb{C}^n$  are orthogonal if any vector in  $X$  is orthogonal to any vector in  $Y$ . In this case, the linear space generated by  $X \cup Y$  is denoted by  $X \oplus Y$ . A matrix  $M \in \mathbb{C}^{n \times n}$  is said to be *unitary* whenever  $MM^\dagger = I = M^\dagger M$ , where  $I \in \mathbb{C}^{n \times n}$  is the identity matrix. Equivalently,  $M$  is unitary if and only if it preserves the norm, i.e.,  $\|\varphi M\| = \|\varphi\|$  for any  $\varphi \in \mathbb{C}^n$ .  $M$  is said to be *Hermitian* whenever  $M = M^\dagger$ . For a Hermitian matrix  $\mathcal{O} \in \mathbb{C}^{n \times n}$ , let  $c_1, \dots, c_s$  be its eigenvalues and  $E_1, \dots, E_s$  the corresponding eigenspaces. It is well known that each eigenvalue  $c_k$  is real, that  $E_i$  is orthogonal to  $E_j$ , for every  $1 \leq i \neq j \leq s$ , and that  $E_1 \oplus \dots \oplus E_s = \mathbb{C}^n$ . So, every vector  $\varphi \in \mathbb{C}^n$  can be uniquely decomposed as  $\varphi = \varphi_1 + \dots + \varphi_s$ , for unique  $\varphi_j \in E_j$ . The linear transformation  $\varphi \mapsto \varphi_j$  is the *projector*  $P_j$  onto the subspace  $E_j$ . Actually, the Hermitian matrix  $\mathcal{O}$  is biunivocally determined by its eigenvalues and projectors as  $\mathcal{O} = \sum_{i=1}^s c_i P_i$ . We recall that a matrix  $P \in \mathbb{C}^{n \times n}$  is a projector if and only if  $P$  is Hermitian and idempotent (i.e.,  $P^2 = P$ ). As we will see, unitary matrices describe evolution in quantum systems, while Hermitian matrices represent observables to be measured.

We recall that  $S \subseteq \mathbb{C}^n$  is a *compact set* if and only if every infinite sequence of elements in  $S$  contains a convergent subsequence, whose limit lies in  $S$ . For a given vector  $\varphi \in \mathbb{C}^n$  and a real positive value  $r$ , we define the set  $\mathcal{B}_r(\varphi) = \{v \in \mathbb{C}^n \mid \|v - \varphi\| \leq r\}$  as the *ball of radius  $r$  centered in  $\varphi$* . The balls  $\mathcal{B}_r(\varphi)$  are examples of compact sets in  $\mathbb{C}^n$ .

We assume the reader is familiar with basic notions on formal language theory (see, e.g., [11]). The set of all words (including the empty word  $\varepsilon$ ) over a finite alphabet  $\Sigma$  is denoted by  $\Sigma^*$ , and with  $\Sigma^n$  we denote the set of words of length  $n$ .

A *deterministic finite state automaton* (DFA) is a 5-tuple  $D = \langle Q, \Sigma, \tau, q_1, F \rangle$ , where  $Q$  is the finite set of states,  $\Sigma$  the finite input alphabet,  $q_1 \in Q$  the initial state,  $F \subseteq Q$  the set of final (accepting) states, and  $\tau : Q \times \Sigma \rightarrow Q$  is the transition function. An input word is *accepted*, if the induced computation starting from the state  $q_1$  ends in some final state  $q \in F$  after consuming the whole input. The set of all words accepted by  $D$  is denoted by  $L_D$  and called the accepted language. An alternative equivalent representation for  $D$  is by the

3-tuple  $D = \langle \alpha, \{M(\sigma)\}_{\sigma \in \Sigma}, \beta \rangle$ , where  $\alpha \in \{0, 1\}^{|\mathcal{Q}|}$  is the characteristic row vector of the initial state,  $M(\sigma) \in \{0, 1\}^{|\mathcal{Q}| \times |\mathcal{Q}|}$  is the boolean matrix satisfying  $(M(\sigma))_{ij} = 1$  if and only if  $\tau(q_i, \sigma) = q_j$ , and  $\beta \in \{0, 1\}^{|\mathcal{Q}| \times 1}$  is the characteristic column vector of the final states. The accepted language can now be defined as  $L_D = \{\sigma_1 \cdots \sigma_n \in \Sigma^* \mid \alpha M(\sigma_1) \cdots M(\sigma_n) \beta = 1\}$ .

Let us now introduce the model of quantum finite automata with control language [4,15].

**Definition 1.** *Given an input alphabet  $\Sigma$  and an endmarker symbol  $\# \notin \Sigma$ , a  $q$ -state quantum finite automaton with control language (QFC) is a system  $\mathcal{A} = \langle \phi, \{U(\gamma)\}_{\gamma \in \Gamma}, \mathcal{O}, \mathcal{L} \rangle$ , for  $\Gamma = \Sigma \cup \{\#\}$ , where*

- $\phi \in \mathbb{C}^q$  is the initial amplitude vector satisfying  $\|\phi\| = 1$ ,
- $U(\gamma) \in \mathbb{C}^{q \times q}$  is a unitary matrix, for any  $\gamma \in \Gamma$ ,
- $\mathcal{O} = \sum_{c \in C} cP(c)$  is a Hermitian matrix representing an observable where  $C$ , the set of eigenvalues of  $\mathcal{O}$ , is the set of all possible outcomes of measuring  $\mathcal{O}$ , and  $P(c)$  denotes the projector onto the eigenspace corresponding to  $c \in C$ ,
- $\mathcal{L} \subseteq C^*$  is a regular language, called the control language.

An input for  $\mathcal{A}$  is any word from  $\Sigma^*$  closed by the symbol  $\#$ . The behavior of  $\mathcal{A}$  on  $x_1 \cdots x_n \# \in \Sigma^* \#$  is as follows. At any time, the state of  $\mathcal{A}$  is a vector  $\xi \in \mathbb{C}^q$  with  $\|\xi\| = 1$ . The computation starts in the state  $\phi$ , then transformations associated with the symbols in  $x_1 \cdots x_n \#$  are applied in succession. Precisely, the transformation corresponding to a symbol  $\gamma \in \Gamma$  consists of two steps:

- (i) EVOLUTION: the unitary operator  $U(\gamma)$  is applied to the current state  $\xi$  of the automaton, leading to the new state  $\xi'$ .
- (ii) MEASURING: the observable  $\mathcal{O}$  is measured on  $\xi'$ . According to quantum mechanics principles, the result of measurement is  $c_k$  with probability  $\|\xi' P(c_k)\|^2$ , and the state of the automaton “collapses” to  $\frac{\xi' P(c_k)}{\|\xi' P(c_k)\|}$ .

So, the computation on  $x_1 \cdots x_n \#$  yields a given sequence  $y_1 \cdots y_n y_\#$  of results of the measurements of  $\mathcal{O}$  with probability  $p_{\mathcal{A}}(y_1 \cdots y_n y_\#; x_1 \cdots x_n \#)$  defined as

$$p_{\mathcal{A}}(y_1 \cdots y_n y_\#; x_1 \cdots x_n \#) = \left\| \phi \left( \prod_{i=1}^n U(x_i) P(y_i) \right) U(\#) P(y_\#) \right\|^2.$$

A computation yielding the word  $y_1 \cdots y_n y_\#$  of measure outcomes is *accepting* whenever  $y_1 \cdots y_n y_\# \in \mathcal{L}$ , otherwise it is rejecting. Hence, the probability that the QFC  $\mathcal{A}$  exhibits an accepting computation on input  $x_1 \cdots x_n \#$  is

$$\mathcal{E}_{\mathcal{A}}(x_1 \cdots x_n) = \sum_{y_1 \cdots y_n y_\# \in \mathcal{L}} p_{\mathcal{A}}(y_1 \cdots y_n y_\#; x_1 \cdots x_n \#).$$

The function  $\mathcal{E}_{\mathcal{A}} : \Sigma^* \rightarrow [0, 1]$  is the *stochastic event induced by  $\mathcal{A}$* .

The *language accepted by  $\mathcal{A}$  with cut point  $\lambda \in [0, 1]$*  is the set of words  $L_{\mathcal{A}, \lambda} = \{x \in \Sigma^* \mid \mathcal{E}_{\mathcal{A}}(x) > \lambda\}$ . The cut point is said to be *isolated* whenever there exists  $\delta \in (0, \frac{1}{2}]$  such that  $|\mathcal{E}_{\mathcal{A}}(x) - \lambda| \geq \delta$ , for any  $x \in \Sigma^*$ .

When referring to the size of a QFC, we must account for both the quantum and the classical component. Hence, in what follows, we say that  $\mathcal{A}$  has  $q$  quantum states and  $k$  classical states whenever it is a  $q$ -state QFC and the control language  $\mathcal{L}$  is recognized by a  $k$ -state DFA.

Throughout the paper, we say that two automata are *equivalent* whenever they accept the same language.

### 3 Converting QFCs to DFAs

We start by defining a matrix representation for QFCs. Then, for any given QFC, we construct an equivalent DFA by suitably generalizing Rabin's technique. Finally, we analyze the state complexity of the resulting DFA with respect to the size of the original QFC.

#### 3.1 Linear Representation of QFCs

A convenient way to work with QFCs is by using their linear representation [4]. Let  $\mathcal{A} = \langle \phi, \{U(\sigma)\}_{\sigma \in \Gamma}, \mathcal{O} = \sum_{c \in C} cP(c), \mathcal{L} \rangle$  be a QFC with  $\delta$ -isolated cut point  $\lambda$ , and let  $D = \langle \alpha, \{M(c)\}_{c \in C}, \beta \rangle$  be the minimal DFA recognizing  $\mathcal{L}$ . Denote by  $q$  and  $k$  the number of quantum and classical states of  $\mathcal{A}$ . We define the *linear representation* of  $\mathcal{A}$  as the 3-tuple  $\text{Li}(\mathcal{A}) = \langle \varphi_0, \{V(\sigma)\}_{\sigma \in \Gamma}, \eta \rangle$  with

- $\varphi_0 = (\phi \otimes \phi^* \otimes \alpha)$ , a vector in  $\mathbb{C}^{q^2k}$ ,
- $V(\sigma) = (U(\sigma) \otimes U^\dagger(\sigma) \otimes I) \cdot \sum_{c \in C} P(c) \otimes P(c) \otimes M(c)$ , a matrix in  $\mathbb{C}^{q^2k \times q^2k}$ ,
- $\eta = \sum_{j=1}^q e_j \otimes e_j \otimes \beta$ , a vector in  $\mathbb{C}^{q^2k}$ ,

where  $e_j$  is the vector with 1 in its  $j$ th component and 0 elsewhere. The main point, not so hard to verify, is that  $\text{Li}(\mathcal{A})$  enables us to represent the stochastic event induced by  $\mathcal{A}$  as  $\mathcal{E}_{\mathcal{A}}(x) = \varphi_0 V(x\#) \eta$ , where we let  $V(\omega) = \prod_{i=1}^n V(\sigma_i)$  for any  $\omega = \sigma_1 \cdots \sigma_n \in \Gamma^*$ . In addition, as shown in [4], we have  $\|\varphi_0 V(\omega)\| \leq 1$  for any  $\omega \in \Gamma^*$ . Therefore, all the state vectors of  $\text{Li}(\mathcal{A})$  belong to the unitary ball  $\mathcal{B}_1(\mathbf{0}) \subset \mathbb{C}^{q^2k}$  centered in the zero-vector  $\mathbf{0}$ .

We are going to show a crucial result saying, roughly speaking, that any word  $\omega$  induces an evolution in  $\text{Li}(\mathcal{A})$  which increases the distance between two different starting vectors only by a constant factor *not depending on the length of  $\omega$* . To this aim, we need some technical lemmas, the first one shown in [4]:

**Lemma 1.** *For any  $\sigma \in \Sigma$ , let  $U(\sigma)$  be a unitary matrix, and let an observable  $\mathcal{O} = \sum_{c \in C} cP(c)$ . Then, for any complex vector  $\varphi$  and word  $\sigma_1 \cdots \sigma_n \in \Gamma^*$ , we have  $\sum_{y=y_1 \cdots y_n \in C^n} \|\varphi \prod_{j=1}^n U(\sigma_j) P(y_j)\|^2 = \|\varphi\|^2$ .*

The next lemma states a property of vectors lying within unitary balls. From now on, for the sake of brevity, we will simply write  $\mathcal{B}_1$  to denote a unitary ball centered in  $\mathbf{0}$ , regardless the dimension of the space within which such a ball is embedded.

**Lemma 2.** *For any  $v, v' \in \mathcal{B}_1$  satisfying  $\|v'\| \geq \|v\|$  and  $\cos(\text{ang}(v', v)) \geq 0$ , we have  $\cos(\text{ang}(v' - v, v)) \geq -\frac{1}{\sqrt{2}}$ .*

We are now ready to prove the crucial result on the distance between trajectories in  $\text{Li}(\mathcal{A})$ :

**Lemma 3.** *For any state vectors  $\varphi = v \otimes v^* \otimes a$  and  $\varphi' = v' \otimes v'^* \otimes a'$  of  $\text{Li}(\mathcal{A})$ , and any  $\omega \in \Gamma^*$ , we have*

$$\|\varphi'V(\omega) - \varphi V(\omega)\| \leq 4\|\varphi' - \varphi\|. \quad (1)$$

*Proof.* We consider the case in which  $a = a'$ , and quickly address the opposite case at the end of the proof. Without loss of generality, we can assume that  $\|v'\| \geq \|v\|$ . Moreover, we assume that  $\cos(\text{ang}(v', v)) \geq 0$ . Otherwise, we can consider the vector  $-v'$  instead of  $v'$ , for which it holds  $\cos(\text{ang}(-v', v)) \geq 0$ , and the proof works unchanged since  $(-v') \otimes (-v')^* \otimes a = v' \otimes v'^* \otimes a = \varphi'$ .

By letting  $\Delta = v' - v$ , we have  $\varphi' - \varphi = v \otimes \Delta^* \otimes a + \Delta \otimes v^* \otimes a + \Delta \otimes \Delta^* \otimes a$ . So, we can rewrite the left side of Inequality (1) as

$$\begin{aligned} \|(\varphi' - \varphi)V(\omega)\| &= \|(v \otimes \Delta^* \otimes a)V(\omega) + (\Delta \otimes v^* \otimes a)V(\omega) + (\Delta \otimes \Delta^* \otimes a)V(\omega)\| \\ &\leq \|(v \otimes \Delta^* \otimes a)V(\omega)\| + \|(\Delta \otimes v^* \otimes a)V(\omega)\| + \|(\Delta \otimes \Delta^* \otimes a)V(\omega)\|. \end{aligned} \quad (2)$$

To simplify Inequality (2), we analyze the generic form  $\|(v_1 \otimes v_2^* \otimes a)V(\omega)\|$ , which can be written as

$$\left\| \sum_{y=y_1 \cdots y_n \in C^n} v_1 \prod_{j=1}^n U(\sigma_j)P(y_j) \otimes v_2^* \prod_{j=1}^n U^\dagger(\sigma_j)P(y_j) \otimes aM(y) \right\|.$$

Since  $D$ , the automaton for the control language  $\mathcal{L} \subseteq C^*$  in  $\mathcal{A}$ , is a DFA, we have  $\|aM(y)\| = 1$  for every  $y \in C^*$ . So, we can write

$$\|(v_1 \otimes v_2^* \otimes a)V(\omega)\| \leq \sum_{y=y_1 \cdots y_n \in C^n} \left\| v_1 \prod_{j=1}^n U(\sigma_j)P(y_j) \right\| \cdot \left\| v_2^* \prod_{j=1}^n U^\dagger(\sigma_j)P(y_j) \right\|.$$

The right side of this inequality can be seen as the inner product between two vectors  $\hat{v}_1, \hat{v}_2$  of dimension  $|C|^n$ , with the  $y$ th component of  $\hat{v}_1$  (resp.,  $\hat{v}_2$ ) being  $\left\| v_1 \prod_{j=1}^n U(\sigma_j)P(y_j) \right\|$  (resp.,  $\left\| v_2^* \prod_{j=1}^n U^\dagger(\sigma_j)P(y_j) \right\|$ ). By Cauchy-Schwarz inequality, we have  $|\langle \hat{v}_1, \hat{v}_2 \rangle| \leq \|\hat{v}_1\| \|\hat{v}_2\|$ . So, by Lemma 1, we can write

$$\begin{aligned} \|(v_1 \otimes v_2^* \otimes a)V(\omega)\| &\leq \sqrt{\sum_{y_1 \cdots y_n} \left\| v_1 \prod_{j=1}^n U(\sigma_j)P(y_j) \right\|^2} \cdot \sqrt{\sum_{y_1 \cdots y_n} \left\| v_2^* \prod_{j=1}^n U^\dagger(\sigma_j)P(y_j) \right\|^2} \\ &= \sqrt{\|v_1\|^2 \|v_2\|^2} = \|v_1\| \|v_2\|. \end{aligned}$$

By replacing  $v_1$  and  $v_2$  with the vectors involved in Inequality (2), we obtain

$$\|\varphi'V(\omega) - \varphi V(\omega)\| \leq 2\|v\| \|\Delta\| + \|\Delta\|^2. \quad (3)$$

We now analyze the right side of Inequality (1). We first observe that

$$\begin{aligned}
\|\varphi' - \varphi\|^2 &= \|v \otimes \Delta^* + \Delta \otimes v^* + \Delta \otimes \Delta^*\|^2 && \text{(since } \|a\| = 1\text{)} \\
&= \|v\|^2 \|\Delta\|^2 + \|\Delta\|^2 \|v\|^2 + \|\Delta\|^2 \|\Delta\|^2 + 2(\langle v, \Delta \rangle \langle \Delta^*, v^* \rangle)_R + \\
&\quad + 2(\langle v, \Delta \rangle \langle \Delta^*, \Delta^* \rangle)_R + 2(\langle \Delta, \Delta \rangle \langle v^*, \Delta^* \rangle)_R \\
&= \|v\|^2 \|\Delta\|^2 + \|\Delta\|^2 \|v\|^2 + \|\Delta\|^2 \|\Delta\|^2 + \\
&\quad + 2|\langle v, \Delta \rangle|^2 + 2(\langle v, \Delta \rangle \|\Delta\|^2)_R + 2(\|\Delta\|^2 \langle v^*, \Delta^* \rangle)_R \\
&\geq 2\|v\|^2 \|\Delta\|^2 + \|\Delta\|^4 + 2(\langle v, \Delta \rangle_R)^2 + 4\|\Delta\|^2 \langle v, \Delta \rangle_R.
\end{aligned}$$

By letting  $\theta = \text{ang}(v, \Delta)$ , we have

$$\|\varphi' - \varphi\|^2 \geq 2\|v\|^2 \|\Delta\|^2 + \|\Delta\|^4 + 2\|v\|^2 \|\Delta\|^2 (\cos(\theta))^2 + 4\|v\| \|\Delta\|^3 \cos(\theta). \quad (4)$$

By joining Inequalities (3) and (4), in order to prove the desired Inequality (1) it is enough to show that

$$(2\|v\| \|\Delta\| + \|\Delta\|^2)^2 \leq 16(\|\Delta\|^4 + 4\|v\| \|\Delta\|^3 \cos(\theta) + 2\|v\|^2 \|\Delta\|^2 (1 + (\cos(\theta))^2)).$$

We can divide both sides by  $\|\Delta\|^2$ , since for  $\|\Delta\| = 0$  the inequality is trivially verified. By solving with respect to  $\|\Delta\|$ , we get that the inequality is always true if it holds  $4\|v\|^2(16 \cos(\theta) - 1)^2 - 60\|v\|^2(8 \cos(\theta)^2 + 7) \leq 0$ . If  $\|v\| = 0$ , this is clearly verified. Otherwise, dividing by  $\|v\|^2$  and routine manipulation lead us to study the equivalent inequality

$$17(\cos(\theta))^2 - 4 \cos(\theta) - 13 \leq 0. \quad (5)$$

Recall that, at the beginning of the proof, we assumed that  $\|v'\| \geq \|v\|$  and  $\cos(\text{ang}(v, v')) \geq 0$ . So, by Lemma 2, we get  $-\frac{1}{\sqrt{2}} \leq \cos(\theta) \leq 1$ . Within this interval, the left side of Inequality (5) is never positive, whence the result follows.

We conclude by quickly noticing that in the case  $a \neq a'$ , we have  $\langle a, a' \rangle = 0$ . So, one may easily obtain  $\|\varphi' - \varphi\|^2 = \|v'\|^4 + \|v\|^4$  and  $\|(\varphi' - \varphi)V(\omega)\| \leq \|v'\|^2 + \|v\|^2$ , and the claimed result again follows.  $\square$

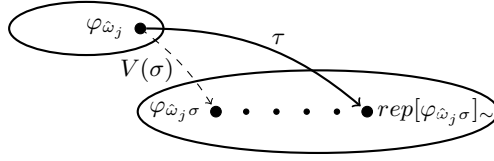
### 3.2 Conversion to DFAs

We are now ready to construct a DFA  $D_{\mathcal{A}}$  equivalent to the QFC  $\mathcal{A}$ , by using the linear representation  $\text{Li}(\mathcal{A}) = \langle \varphi_0, \{V(\sigma)\}_{\sigma \in \Gamma}, \eta \rangle$ .

For any word  $\omega \in \Sigma^*$ , let  $\varphi_\omega = \varphi_0 V(\omega)$  be the state vector reached by  $\text{Li}(\mathcal{A})$  after reading  $\omega$ . We define the relation  $\sim$  on the set  $\{\varphi_\omega \mid \omega \in \Sigma^*\} \subseteq \mathcal{B}_1$  as:

$$\varphi_\omega \sim \varphi_{\omega'} \iff \begin{array}{l} \text{there exists a sequence of words } \omega_1, \omega_2, \dots, \omega_n \in \Sigma^* \\ \text{satisfying } \omega = \omega_1, \omega' = \omega_n, \text{ and } \|\varphi_{\omega_i} - \varphi_{\omega_{i+1}}\| < \frac{\delta}{2\sqrt{qk}}. \end{array}$$

It is easy to verify that  $\sim$  is an equivalence relation, and that the distance between two vectors belonging to different equivalence classes is at least  $\frac{\delta}{2\sqrt{qk}}$ . This latter fact shows that  $\sim$  is of *finite* index, since otherwise, by taking one



**Fig. 1.** The transition  $\tau$  on a symbol  $\sigma$ . The dots represent state vectors of  $\text{Li}(\mathcal{A})$ , while the ellipses indicate equivalence classes of  $\sim$ . The smaller points between  $\varphi_{\hat{\omega}_j\sigma}$  and  $\text{rep}[\varphi_{\hat{\omega}_j\sigma}]_{\sim}$  represent the state vectors at distance smaller than  $\frac{\delta}{2\sqrt{qk}}$  witnessing the relation  $\sim$  between them. The dashed arrow indicates the original evolution on  $\text{Li}(\mathcal{A})$ , while the full arrow represents the behavior of the DFA  $D_{\mathcal{A}}$ .

vector from each class, one could construct an infinite sequence of elements in  $\mathcal{B}_1$  which cannot have any convergent subsequence, against the compactness of  $\mathcal{B}_1$ . Therefore, by letting  $s$  be the index of  $\sim$ , we choose a representative for each equivalence class, and call them  $\varphi_{\hat{\omega}_1}, \varphi_{\hat{\omega}_2}, \dots, \varphi_{\hat{\omega}_s}$ . In addition, for any word  $\omega \in \Sigma^*$ , we let  $\text{rep}[\varphi_{\omega}]_{\sim}$  denote the representative of the equivalence class the state vector  $\varphi_{\omega}$  belongs to.

We construct our DFA  $D_{\mathcal{A}}$  as follows:

- the *set of states* coincides with the set of representatives  $\{\varphi_{\hat{\omega}_1}, \varphi_{\hat{\omega}_2}, \dots, \varphi_{\hat{\omega}_s}\}$ ,
- the *input alphabet* is  $\Sigma$ ,
- the *initial state* is the vector  $\text{rep}[\varphi_{\varepsilon}]_{\sim}$ , which we assume to be  $\varphi_{\hat{\omega}_1}$ ,
- the *transition function* is defined, for any  $\sigma \in \Sigma$ , as  $\tau(\varphi_{\hat{\omega}_j}, \sigma) = \text{rep}[\varphi_{\hat{\omega}_j\sigma}]_{\sim}$ ; a step of  $\tau$  is intuitively shown in Fig. 1,
- the *final states* are the representatives  $\{\varphi_{\hat{\omega}_j} \mid \varphi_{\hat{\omega}_j} V(\#)\eta \geq \lambda + \delta\}$  associated with words accepted in the original QFC  $\mathcal{A}$ ; equivalently,  $\varphi_{\hat{\omega}_j}$  is final if and only if its equivalence class contains  $\varphi_{\omega}$  for some word  $\omega\#$  accepted by  $\mathcal{A}$ .

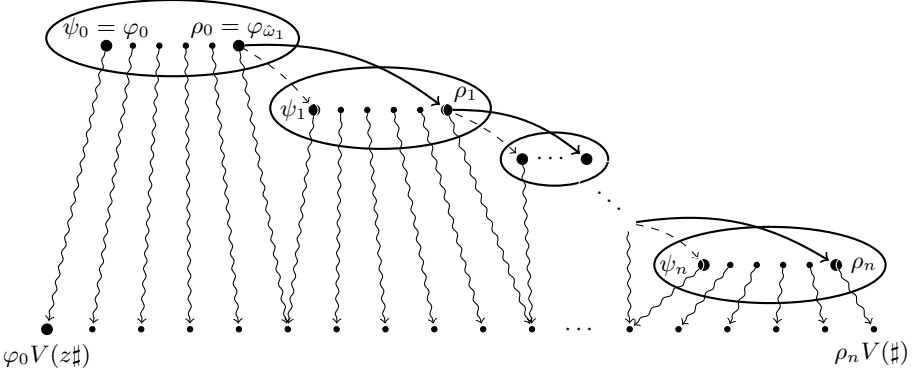
Before showing the correctness of our construction, we stress the fact that the equivalence relation  $\sim$  is *not a congruence* (in fact,  $\varphi_{\omega} \sim \varphi_{\omega'}$  does not necessarily implies  $\varphi_{\omega\sigma} \sim \varphi_{\omega'\sigma}$  for  $\sigma \in \Sigma$ , as the reader may easily verify). So, the correctness does not come straightforwardly as in Rabin's setting, but we need an explicit proof:

**Theorem 1.**  $D_{\mathcal{A}}$  is equivalent to  $\mathcal{A}$ .

*Proof.* We begin by introducing some notation:

- For a word  $z = z_1z_2 \dots z_n \in \Sigma^*$ , we let  $z_{\{j\}} = z_1z_2 \dots z_j$  be the prefix of  $z$  of length  $j$ , and  $z_{\{-j\}} = z_{j+1}z_{j+2} \dots z_n$  the remaining suffix.
- We let  $\rho_j = \tau(\varphi_{\hat{\omega}_1}, z_{\{j\}})$  be the state reached by  $D_{\mathcal{A}}$  after reading the first  $j$  symbols of  $z$ . So,  $\rho_0 = \varphi_{\hat{\omega}_1}$  is the initial state of  $D_{\mathcal{A}}$ .
- We let  $\psi_j = \rho_{j-1}V(z_j)$  be the state vector reached by  $j - 1$  steps of  $D_{\mathcal{A}}$  followed by one step of  $\text{Li}(\mathcal{A})$ . So,  $\psi_0 = \varphi_0$  is the initial state of  $\text{Li}(\mathcal{A})$ .

Note that, for each  $0 \leq j \leq n$ , we have  $\psi_j \sim \rho_j$  since  $\rho_j = \text{rep}[\psi_j]_{\sim}$ . Moreover, by definition, the vectors witnessing  $\psi_j \sim \rho_j$  are reachable in  $\text{Li}(\mathcal{A})$ . Formally: there



**Fig. 2.** Evolution scheme of the computation over the word  $z\#$ . The full arrows describe the transitions of the DFA  $D_{\mathcal{A}}$ , while the snake arrows denote the evolution in  $\text{Li}(\mathcal{A})$  from each vector  $\gamma_{j,t}$  in the equivalence class reached after  $j$  symbols, through the dynamic  $V$  over the remaining suffix  $z_{\{-j\}}\#$ , leading to the vector  $\gamma_{j,t}V(z_{\{-j\}}\#)$  in the bottom chain. In this bottom chain, the leftmost point denotes the vector reached by  $\text{Li}(\mathcal{A})$  after reading  $z\#$ , while the rightmost point is the state reached by  $D_{\mathcal{A}}$  after reading  $z$ , with a final transition of  $\text{Li}(\mathcal{A})$  on  $\#$ . Intuitively, the correctness of  $D_{\mathcal{A}}$  comes from the fact that all the vectors in the bottom chain are sufficiently close to their neighbors to represent either all accepting or all rejecting quantum states in the original QFC  $\mathcal{A}$ .

exists a sequence  $\psi_j = \gamma_{j,1}, \gamma_{j,2}, \dots, \gamma_{j,\ell_j} = \rho_j$  satisfying  $\|\gamma_{j,i} - \gamma_{j,i+1}\| < \frac{\delta}{2\sqrt{qk}}$ , and there exist  $x_{j,t} \in \Sigma^*$  such that  $\varphi_0 V(x_{j,t}) = \gamma_{j,t}$  for  $1 \leq t \leq \ell_j$ . As a consequence of Lemma 3, for every  $0 \leq j \leq n$  and  $1 \leq t \leq \ell_j$ , we have

$$\|\gamma_{j,t}V(z_{\{-j\}}\#) - \gamma_{j,(t+1)}V(z_{\{-j\}}\#)\| < 4 \cdot \frac{\delta}{2\sqrt{qk}} = \frac{2\delta}{\sqrt{qk}}. \quad (6)$$

In addition, since

$$\rho_j V(z_{\{-j\}}\#) = \psi_{j+1} V(z_{\{-(j+1)\}}\#),$$

for all  $j$ 's, Inequality (6) implies that the vectors  $\rho_j V(z_{\{-j\}}\#)$  form a chain of vectors from the final state vector  $\varphi_0 V(z\#)$  of  $\text{Li}(\mathcal{A})$  to the vector  $\rho_n V(\#)$ , where the distance between each pair of consecutive vectors is strictly smaller than  $\frac{2\delta}{\sqrt{qk}}$ .

This is intuitively shown in Fig. 2.

We first show that  $z \in L_{\mathcal{A},\lambda} \Rightarrow \tau(\varphi_{\omega_1}, z) \in F$ , which is equivalent to showing

$$\varphi_0 V(z\#) \eta \geq \lambda + \delta \Rightarrow \rho_n V(\#) \eta \geq \lambda + \delta. \quad (7)$$

Note that  $\varphi_0 = \gamma_{0,1}$ ,  $\rho_n = \gamma_{n,\ell_n}$ , and that, for  $0 \leq j \leq n$  and  $1 \leq t \leq \ell_j$ , all  $\gamma_{j,t}$ 's witnessing the relation  $\sim$  are reachable in  $\text{Li}(\mathcal{A})$  through some word  $x_{j,t} \in \Sigma^*$ , i.e.,  $\gamma_{j,t}V(z_{\{-j\}}\#) = \varphi_0 V(x_{j,t} \cdot z_{\{-j\}}\#)$ . Since  $\lambda$  is a  $\delta$ -isolated cut point, we have

$$\gamma_{j,t}V(z_{\{-j\}}\#) \eta \begin{cases} \geq \lambda + \delta & \text{if } x_{j,t}z_{\{-j\}} \in L_{\mathcal{A},\lambda}, \\ \leq \lambda - \delta & \text{if } x_{j,t}z_{\{-j\}} \notin L_{\mathcal{A},\lambda}. \end{cases}$$



Assume, by contradiction, that Inequality (7) does not hold. Then, there exists a position in the bottom chain of Fig. 2 where the acceptance probability associated with a state vector in the chain is above the cut point, while the acceptance probability associated to its right neighbor is below the cut point. More formally, there must exist  $\iota, \kappa$  such that:

$$\gamma_{\iota, \kappa} V(z_{\{-\iota\}} \#) \eta \geq \lambda + \delta \quad \text{and} \quad \gamma_{\iota, (\kappa+1)} V(z_{\{-\iota\}} \#) \eta \leq \lambda - \delta,$$

From these two inequalities and by observing that  $\|\eta\| \leq \sqrt{qk}$ , we get

$$\begin{aligned} 2\delta &\leq \|(\gamma_{\iota, \kappa} V(z_{\{-\iota\}} \#) - \gamma_{\iota, (\kappa+1)} V(z_{\{-\iota\}} \#)) \eta\| \\ &\leq \|\gamma_{\iota, \kappa} V(z_{\{-\iota\}} \#) - \gamma_{\iota, (\kappa+1)} V(z_{\{-\iota\}} \#)\| \|\eta\| \\ &\leq \|\gamma_{\iota, \kappa} V(z_{\{-\iota\}} \#) - \gamma_{\iota, (\kappa+1)} V(z_{\{-\iota\}} \#)\| \cdot \sqrt{qk} \\ &< \frac{2\delta}{\sqrt{qk}} \cdot \sqrt{qk} = 2\delta \quad (\text{by Inequality 6}). \end{aligned}$$

which is an *absurdum*.

Symmetrically, one can show that  $z \notin L_{\mathcal{A}} \Rightarrow \tau(\varphi_{\hat{\omega}_1}, z) \notin F$ , and this completes the proof.

### 3.3 Size Cost of the Conversion

We now analyze the cost, in terms of number of states, of the above conversion from QFCs to DFAs. This will enable us to obtain a general gap at most exponential between the succinctness of the quantum and classical paradigm.

**Theorem 2.** *For any given QFC  $\mathcal{A}$  with  $q$  quantum states,  $k$  classical states, and  $\delta$ -isolated cut point, there exists an equivalent DFA  $D_{\mathcal{A}}$  with  $s$  states satisfying*

$$s \leq \left(1 + \frac{4\sqrt{qk}}{\delta}\right)^{q^2 k}.$$

*Proof.* Let  $\text{Li}(\mathcal{A}) = \langle \varphi_0, \{V(\sigma)\}_{\sigma \in \Gamma}, \eta \rangle$  be the linear representation of  $\mathcal{A}$ . As observed in Section 3.1, its state vectors lies within  $\mathcal{B}_1(\mathbf{0}) \subset \mathbb{C}^d$ , for  $d = q^2 k$ . When constructing the equivalent DFA  $D_{\mathcal{A}}$  as described in Section 3.2, the number  $s$  of states of  $D_{\mathcal{A}}$  coincides with the number of equivalence classes of the relation  $\sim$ .

To estimate  $s$ , consider the ball  $\mathcal{B}_{\frac{\delta}{4\sqrt{qk}}}(\varphi_{\hat{\omega}_i}) \subset \mathbb{C}^d$ , for each representative  $\varphi_{\hat{\omega}_i}$ . Clearly, such a ball is disjoint from the analogous ball centered in  $\varphi_{\hat{\omega}_j}$ , for every  $1 \leq i \neq j \leq s$ . Moreover, all such balls are contained in  $\mathcal{B}_{1+\frac{\delta}{4\sqrt{qk}}}(\mathbf{0}) \subset \mathbb{C}^d$ , and their number is exactly the number  $s$  of equivalence classes of  $\sim$ . Since the volume of a  $d$ -dimensional ball of radius  $r$  is  $Kr^d$ , for a suitable constant  $K$  depending on  $d$ , there exist at most

$$\frac{K(1 + \delta/4\sqrt{qk})^d}{K(\delta/4\sqrt{qk})^d} = \left(1 + \frac{4\sqrt{qk}}{\delta}\right)^{q^2 k}$$

balls of radius  $\frac{\delta}{4\sqrt{qk}}$  in  $\mathcal{B}_{1+\frac{\delta}{4\sqrt{qk}}}(\mathbf{0})$ . So, this number is an upper bound for  $s$ .  $\square$

## 4 Size Lower Bound for Quantum Paradigms

By using the inequality of Theorem 2 “the other way around”, we are able to state lower limits to the descriptonal power of QFCs:

**Theorem 3.** *Any QFC with  $q$  quantum states,  $k$  classical states, and  $\delta$ -isolated cut point accepting a regular language whose minimal DFA has  $\mu$  states, satisfies*

$$qk \geq \left( \frac{\log(\mu)}{\log\left(\frac{5}{\delta}\right)} \right)^{\frac{4}{9}}.$$

*Proof.* From our QFC, we can obtain an equivalent DFA with a number of states bounded as in Theorem 2. Thus, for  $\delta \in (0, \frac{1}{2}]$  and  $q, k \geq 1$ , we have

$$\mu \leq \left( 1 + \frac{4\sqrt{qk}}{\delta} \right)^{q^2k} \leq \left( \frac{5\sqrt{qk}}{\delta} \right)^{q^2k} \leq \left( \frac{5}{\delta} \right)^{\sqrt[4]{qk} \cdot q^2k} \leq \left( \frac{5}{\delta} \right)^{(qk)^{\frac{9}{4}}},$$

whence the result follows.  $\square$

The lower bound in Theorem 3 is not only interesting in the world of QFCs, but it turns out to have several applications in the world of quantum automata. In fact, as recalled in the Introduction, QFCs represent a general unifying framework within which several types of quantum automata may directly and naturally be represented. In particular, in [4] it is proved that: (i) Any  $q$ -state measure-once quantum finite automaton (MO-QFA) can be simulated by a QFC with  $2q$  quantum states and 1 classical state. (ii) Any  $q$ -state measure-many quantum finite automaton (MM-QFA) can be simulated by a QFC with  $q$  quantum states and 3 classical states. (iii) Any  $q$ -state quantum reversible automaton (QRA) can be simulated by a QFC with  $q$  quantum states and 2 classical states. So, by such simulation results and Theorem 3, one immediately gets

**Theorem 4.** *To accept a regular language having a  $\mu$ -state minimal DFA by a MO-QFA, MM-QFA or QRA, at least  $\kappa (\log(\mu)/\log(\frac{5}{\delta}))^{4/9}$  states are necessary, with  $\kappa = 1/2$  for MO-QFA and QRA, and  $\kappa = 1/3$  for MM-QFA.*

A better asymptotically optimal lower bound of  $\log(\mu)/(2\log(1 + 2/\delta))$  is obtained in [6] for MO-QFAs. There, however, Rabin’s approach has a more direct application since the equivalence relation yielding the states of the equivalent DFA is in fact a congruence, so the correctness of the DFA is straightforward. In the case of QFCs, instead, the equivalence relation  $\sim$  is not a congruence, so we had to ensure that, starting from two different state vectors in the same equivalence class, after the evolution on the same word, the two resulting vectors are still either both accepting or both rejecting, even if they belong to different classes. This was possible because of the property proved in Lemma 3.

As natural open problems, it remains either to witness the optimality of our size lower bound for QFCs, or to improve it, especially for the particular cases of simulated machines such as, e.g., MM-QFAs and QRAs. Moreover, one of the

anonymous referees pointed out another general framework, namely *quantum automata with open time evolution* [10], which may be worth investigating by the same geometrical approach, since the computation of such devices on a given input can also be linearized [20].

**Acknowledgements.** The authors wish to thank Alberto Bertoni for useful discussions, and the anonymous referees for their comments.

## References

1. Ambainis, A., Beaudry, M., Golovkins, M., Kikusts, A., Mercer, M., Thérien, D.: Algebraic results on quantum automata. *Th. Comp. Sys.* 39, 165–188 (2006)
2. Ambainis, A., Freivalds, R.: 1-way quantum finite automata: strengths, weaknesses and generalizations. In: *Proc. 39th Symp. Found. Comp. Sci.*, pp. 332–342 (1998)
3. Ambainis, A., Yakaryilmaz, A.: Superiority of exact quantum automata for promise problems. *Information Processing Letters* 112, 289–291 (2012)
4. Bertoni, A., Mereghetti, C., Palano, B.: Quantum computing: 1-way quantum automata. In: Ésik, Z., Fülöp, Z. (eds.) *DLT 2003. LNCS*, vol. 2710, pp. 1–20. Springer, Heidelberg (2003)
5. Bertoni, A., Mereghetti, C., Palano, B.: Small size quantum automata recognizing some regular languages. *Theoretical Computer Science* 340, 394–407 (2005)
6. Bertoni, A., Mereghetti, C., Palano, B.: Some formal tools for analyzing quantum automata. *Theoretical Computer Science* 356, 14–25 (2006)
7. Bianchi, M.P., Palano, B.: Events and languages on unary quantum automata. *Fundamenta Informaticae* 104, 1–15 (2010)
8. Brodsky, A., Pippenger, N.: Characterizations of 1-way quantum finite automata. *SIAM J. Comput.* 5, 1456–1478 (2002)
9. Golovkins, M., Kravtsev, M.: Probabilistic reversible automata and quantum automata. In: Ibarra, O.H., Zhang, L. (eds.) *COCOON 2002. LNCS*, vol. 2387, pp. 574–583. Springer, Heidelberg (2002)
10. Hirvensalo, M.: Quantum automata with open time evolution. *Int. J. Nat. Comp. Res.* 1, 70–85 (2010)
11. Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading (2001)
12. Hughes, R.I.G.: *The Structure and Interpretation of Quantum Mechanics*. Harvard University Press, Cambridge (1992)
13. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: *Proc. 38th Annual Symposium on Foundations of Computer Science*, pp. 66–75 (1997)
14. Moore, C., Crutchfield, J.: Quantum automata and quantum grammars. *Theoretical Computer Science* 237, 275–306 (2000)
15. Mereghetti, C., Palano, B.: Quantum finite automata with control language. *Theoretical Informatics and Applications* 40, 315–332 (2006)
16. Nayak, A.: Optimal lower bounds for quantum automata and random access codes. In: *Proc. 40th Symposium on Foundations of Computer Science*, pp. 369–376 (1999)
17. Rabin, M.O.: Probabilistic automata. *Information and Control* 6, 230–245 (1963)
18. Scharnhorst, K.: Angles in complex vector spaces. *Act. Ap. Math.* 69, 95–103 (2001)
19. Shilov, G.: *Linear Algebra*. Prentice Hall (1971); Reprinted by Dover (1977)
20. Yakaryilmaz, A., Cem Say, A.C.: Unbounded-error quantum computation with small space bounds. *Information & Computation* 209, 873–892 (2011)
21. Zheng, S., Qiu, D., Li, L., Gruska, J.: One-way finite automata with quantum and classical states. In: Bordihn, H., Kutrib, M., Truthe, B. (eds.) *Languages Alive. LNCS*, vol. 7300, pp. 273–290. Springer, Heidelberg (2012)

# Population Protocols on Graphs: A Hierarchy

Olivier Bournez and Jonas Lefèvre

Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France  
{bournez,jlefevre}@lix.polytechnique.fr

**Abstract.** Population protocols have been introduced as a model in which anonymous finite-state agents stably compute a predicate of the multiset of their inputs via interactions by pairs. In this paper, we consider population protocols acting on families of graphs, that is to say on particular topologies. Stably computable predicates on strings of size  $n$  correspond exactly to languages of  $\text{NSPACE}(n)$ , that is to say to non-deterministic space of Turing machines. Stably computable predicates on cliques correspond to semi-linear predicates, namely exactly those definable in Presburger's arithmetic. Furthermore, we exhibit a strict hierarchy in-between when considering graphs between strings and cliques.

**Keywords:** population protocols, computability, hierarchy, space complexity.

## 1 Introduction

The model of population protocol has been introduced in [1] as a model of anonymous agents, with finitely many states, that interact in pairs according to some rules. Agents are assumed to be passively mobile, in the sense that there is no control over the way the interactions happen: interactions can be seen as resulting from a fair scheduler. The model has been designed to decide predicates. Given some input configuration, the agents have to decide whether this input satisfies the predicate, in which case the population of agents has to eventually stabilize to a configuration in which every agent is in an accepting state. A protocol must work for any size of population. Predicates computable by classical population protocols have been characterized as being precisely the semi-linear predicates, that is those predicates on counts of input agents definable in first-order Presburger arithmetic. Semi-linearity was shown to be sufficient in [1], and necessary in [2].

So far, most of the works on population protocols has concentrated on characterizing what predicates on the input configurations can be stably computed in different variants of the models and under various assumptions. Variants of the original model considered so far include restriction to one-way communications [3], restriction to particular interaction graphs [4], and random interactions [1]. Various kinds of fault tolerance has been considered for population protocols [8], including the search for self-stabilizing solutions [5]. We refer to [6] for a comprehensive 2007 survey.

From a computability point of view, the equivalence between predicates computable by population protocols and Presburger’s arithmetic definable sets (semi-linear sets) is rather intriguing. The upper bound obtained in [2] is non-trivial and rather different from classical arguments in computability or complexity theory when dealing with computational models. The work in this paper originates from an attempt to better understand the relations between the population protocol model and classical models like Turing machines.

In that perspective we consider population protocols over various interaction graphs in the spirit of [1,4]. This can be also seen as considering population protocols over particular spatial models.

Clearly classical population protocols corresponding to semi-linear sets correspond to the case where the interaction graph is a clique (i. e. complete graph). It has already been stated that one can simulate a Turing machine when working over a path (or more generally a bounded degree graph) in [4].

We review these constructions and we somehow extend them to more general classes of graphs/topologies.

In particular, we discuss intermediate classes, by considering interaction graphs in between bounded degree graphs and cliques, namely *separable graphs*. We prove that there is a whole hierarchy between paths (non-deterministic linear time) and cliques (semi-linear sets), that can be proved to be strict for graphs verifying some conditions.

## 1.1 Related Work

The following works, not covered by 2007 survey [6] can also be mentioned: in [9] a variant of population protocols where agents have unique identifier and can store a constant number of bits and a constant number of other agents’ identifiers is considered. The model has been proved to be able to compute any predicate computable by a non-deterministic Turing machine in space  $n \log n$ . Moreover the construction has been proved to tolerate Byzantine failures.

The idea of restricting to an underlying graph for interactions is already present in the initial definition of the model in [1]. The paper [4] is devoted to understand which properties of graphs can be computed by population protocols. It is proved for example that one can detect whether a given graph is a path or a tree using population protocols, or whether a graph has a bounded degree.

A variant of the classical model is studied in the paper [7]. This variant considers that each agent is a Turing machine with a space  $f(n)$  for a population of size  $n$ . For  $f(n) = \Omega(\log n)$ , the computational power of the model is characterized to be precisely the class of all symmetric predicates recognizable by a non-deterministic Turing machine in space  $nf(n)$ . A hierarchy for population protocols is inherited from Turing machines. Furthermore it is proved that if agents have  $o(\log \log)$  space then the model is no more powerful than the classical model. Our hierarchy is using both the restrictions of the communication graph and the computational power of the agents.

## 2 Definitions

We restate the population protocol model introduced by Angluin, Aspnes, Diamadi, Fisher and Peralta in [1] in the rest of this section.

Informally, in the basic population protocol model, we have a collection of anonymous agents. Each agent is given an input value, then agents interact pairwise in an order fixed by some scheduler that is assumed to satisfy some fairness guarantee. Each agent is assumed to have finitely many states, and the “program” for the system describes how the states of the two agents evolve when two agents interact. The agents’ output values change over time and must eventually converge to the correct output [6]. More formally:

**Definition 1 (Population protocol).** A population protocol  $(Q, \Sigma, in, out, \delta)$  is formally specified by:

- $Q$ , a finite set of states, that correspond to possible states for an agent;
- $\Sigma$ , a finite input alphabet;
- $in$ , an input function from  $\Sigma$  to  $Q$ , where  $in(\sigma)$  represents the initial state of an agent whose input is  $\sigma$ ;
- $out$ , an output function from  $Q$  to  $\{0, 1\}$ , where  $out(q)$  represents the output value of an agent in state  $q$ ;
- $\delta \subset Q^2 \times Q^2$ , a transition relation that describes how pairs of agents can interact. We will mostly write  $(s_1, s_2) \rightarrow (r_1, r_2)$  for  $((s_1, s_2), (r_1, r_2)) \in \delta$ .

A computation of such a protocol takes place over a fixed set  $A$  of  $n$  agents, where  $n \geq 2$ . A population configuration (over  $Q$ ) is a mapping  $C : A \rightarrow Q$  specifying the state of each agent. We will say that a configuration  $C$  contains a state  $q$  if  $C(a) = q$  for some agent  $a \in A$ . Agents with the same state are assumed in the model indistinguishable. Hence we will always reason modulo permutations of agents. In other words, configurations can also be considered as ordered multisets of states, and we assume that for all permutation  $\pi$  of  $A$ , configurations  $C$  and  $C'$  with  $C'(a) = C(\pi(a))$  are the same.

To a word  $\omega = \omega_1\omega_2\dots\omega_n \in \Sigma^*$  of length  $n$  over input alphabet  $\Sigma$ , corresponds any initial configuration  $C_0[\omega]$  where agents have corresponding initial state: fixing any numbering of the set  $A$  of agents, agent  $1 \leq i \leq n$  is in state  $in(\omega_i)$ . Given some configuration  $C$ , we write  $C \rightarrow C'$  if  $C'$  can be obtained from  $C$  by a single interaction of two agents: this means that  $C$  contains two states  $q_1$  and  $q_2$ , and  $C'$  is obtained from  $C$  by replacing  $q_1$  and  $q_2$  by  $q'_1$  and  $q'_2$ , where  $((q_1, q_2), (q'_1, q'_2)) \in \delta$ . We denote  $\rightarrow^*$  for the transitive and reflexive closure of relation  $\rightarrow$ .

An *execution* over word  $\omega = \omega_1\omega_2\dots\omega_n \in \Sigma^*$  is a finite or infinite sequence of configurations  $C_0C_1\dots C_iC_{i+1}\dots$  such that  $C_0 = C_0[\omega]$  and, for all  $i$ ,  $C_i \rightarrow C_{i+1}$ .

The order in which pairs of agents interact is assumed to be unpredictable, and can be seen as being chosen by an adversary. In order for only meaningful computations to take place, some restriction on the adversarial scheduler must be made (otherwise, nothing would forbid for example that all interactions happen

always between two same agents). As often in the distributed computing context, this is model by some fairness hypotheses: a *fair execution* is an execution such that if a configuration  $C$  appears infinitely often in the execution and  $C \rightarrow C'$  for some configuration  $C'$ . Therefore  $C'$  must appears infinitely often in the execution. This fairness property states that a configuration which is always reachable will eventually be reached. There is absolutely no control over any finite execution, but any possibility that can not be evade with a finite execution will be met. Observe that this is equivalent to say that if a configuration  $C'$  is reachable infinitely often, then it is infinitely reached.

At any step during an execution, each agent' state determines its output at that time: whenever an agent is in state  $q$ , its output is  $out(q)$ . A configuration  $C$  is said to output 1 (respectively 0) iff all agents outputs 1 (resp. 0): in other words  $out(C(a)) = 1$  for all agent  $a \in A$  (resp. 0). If in a configuration there are agents outputting 1 and other outputting 0, the output of the configuration is undefined. A configuration  $C$  is said to be *output-stable* with output  $b \in \{0, 1\}$  if  $\forall C', C \rightarrow^* C', \forall s \in C', out(s) = b$ . Namely a configuration is output-stable if its output is defined and any reachable configuration has the same output.

A fair execution  $C_0C_1 \dots C_iC_{i+1} \dots$  stabilizes with output  $b$  if there exists some  $m$  such that for all  $j \geq m$ , the output of configuration  $C_j$  is defined and given by  $b$ . In particular, a fair execution converges with output  $b$  iff it reaches an output-stable configuration with output  $b$ .

A protocol is well-specified if for any initial configuration  $C_0[w]$ , every fair execution starting from  $C_0[w]$  converges with an output, that is determined by that input configuration. A well specified protocol induces a subset  $L \subset \Sigma^*$  of configurations yielding to output 1: this subset will be said to be the language computed by the protocol.

We will also need to consider population protocols over graphs. Observe that the previous model is just the case where the graph is a clique.

**Definition 2 (Population protocol on a graph).** *Given a graph  $G$  with  $n$  vertices, a population protocol over  $G$  is a population protocol whose set of agents  $A$  is the set of vertices of  $G$ , and where in definition above two agents can interact by a single interaction only if they are adjacent in graph  $G$ : in definitions above, this only changes the definition of  $C \rightarrow C'$ . Formally:  $C \rightarrow C'$  if  $C$  contains two states  $q_1$  and  $q_2$  in respective vertices  $v_1$  and  $v_2$ , and  $C'$  is obtained from  $C$  by replacing  $q_1$  and  $q_2$  by  $q'_1$  and  $q'_2$ , where  $((q_1, q_2), (q'_1, q'_2)) \in \delta$ , and  $(v_1, v_2)$  is an edge of graph  $G$ .*

We insist on the fact that we do not change anything else. In particular the input is still distributed on the population without any control.

Given a family of graphs  $(G_n = (V_n, E_n))_{n \in \mathbb{N}}$ , where  $G_n$  has  $n = |V_n|$  vertices, a well-specified protocol also induces a subset  $L \subset \Sigma^*$ : this subset will also be said to be the language computed by the protocol over this family of graphs.

Equivalently a language  $L \subset \Sigma^*$  is computed by population protocol working over this family if there is a well-specified protocol verifying the following property: the  $w \in L$  correspond exactly to initial configurations  $C_0[w]$  for which

every fair execution of the protocol on  $G_n$ , where  $n$  is the length of  $w$ , eventually stabilize to output 1. From our definitions, languages corresponding to population protocols are necessarily invariant by permutations of letters, as initial configurations are assumed to be indistinguishable when agents are permuted.

Population protocols on bounded degree graphs can be related to Turing machines. The languages computed by population protocols on bounded degree graphs correspond exactly to symmetric languages of  $\text{NSPACE}(n)$ , that is to say to language recognized in non-deterministic space  $n$  by Turing machines, invariant by permutation of letters.

In the spirit of [7], we will also consider variants where the agents are no longer just finite automata but deterministic Turing machines with memory space constraints: this model will be called the *passively mobile machines model*, following the terminology of [7]: the agents are now Turing machines. Formally, they are Turing machines with four tapes: working tape, output tape, incoming message tape and outgoing message tape. They have internal transitions, corresponding to local computations, and external transitions, corresponding to interactions, where the two agents copy in their own incoming message tape the contents of the outgoing message tape of the other agent.

The authors in [7] show that if the  $n$  agents of the population have a memory of size at least  $\log(n)$  then they can be organized to model a Turing machine. Basically, the idea is that agents can use interactions to compute eventually unique identifiers, and then use these identifiers so that each agent then simulate a cell (or few cells) of the tape of a (global) Turing machine.

Here will be discussed smaller cases. Let us first come back to classical population protocols where agents are finite state.

We will often use variations of the following routine electing a leader in a uniform population: consider set of states is  $Q_{leader} = \{L, x\}$ , where state  $L$  means that this agent has a leader token and rules  $\delta_{leader} = \{(L, L) \rightarrow (L, x), (x, L) \rightarrow (L, x)\}$ . Thanks to the first rule, if two agents with leader token meet, one token is deleted. With this rule, the quantity of leader token will decrease to one, but the last token can never be deleted. The second rule allows any leader token to move on the graph. The computation starts with every agent with state  $L$ : in any fair execution, eventually there will remain exactly one  $L$  state in the population. Indeed, since leaders can move they can always meet, and hence the fairness hypothesis guarantees that leaders will eventually meet and disappear until only one remains.

### 3 Modeling a Turing Machine on Graphs

The paper [4] explains how to model a Turing machine with a population protocol working on a graphs family with degree bounded by  $d$ . We will explain the main steps of the construction.

The algorithm of [4] can be decomposed into three sub-algorithm: construct a distance-2-coloring of the graph; use this labeling to build a spanning tree on the graph; and use this spanning tree to model the tape of a Turing machine.



A distance-2-coloring of a graph  $G$  is a coloring of the vertices of  $G$  such that two vertices that are adjacent or have a common neighbor receive distinct colors.

A graph family will be said to be distance-2-colorable if there exists a population protocol such that starting from any initial configuration the system eventually evolves (restricting as usual to fair computations only) to a configuration that corresponds to a distance-2-coloring.

**Proposition 1.** *A graph family is distance-2-colorable iff the graph family has a bounded degree: there exists some constant  $d$  that bounds the degree of each graph of the family.*

*Proof.* The idea of the proof of the if part of the proposition is to use a moving leader token. After a move the token stops and interacts successively with two of its neighbors. If they have the same color, one changes, and the token moves away. Thanks to the fairness property, the population will eventually reach a configuration where no two neighbors of any agent have the same color. If two moving leader tokens meet, one (and only one) is destroyed. A more detailed proof of this can be found in [4].

Conversely, since agents are finite state, the degree is directly bounded by the number of colors (i. e. states) minus one in a distance-2-coloring.  $\square$

Notice that some more general families of graphs could also be distance-2-colorable, if we consider that agents would not be finite state, that is to say passively mobile machines instead of population protocols.

**Proposition 2 ([4]).** *Consider a distance-2-colorable graph family. One can build a population protocol that builds a spanning tree over this graph family.*

*Proof.* The idea of the proof is to observe that a distance-2-coloring of the graph allows one to emulate pointers. Indeed, each agent can store in his state a color  $c_0$  and by definition of the coloring, it can have at most one neighbor with color  $c_0$ ; in other words the color of an agent is like a unique pointer address for its neighbors. This allows one to build a forest where the parent links in the trees are the pointers and the forest can eventually be transformed into a spanning tree using leaders and the fairness of the scheduler.

The construction itself uses leader tokens. The leader token starts its construction on a vertex that will be the root. Then it explores the graph in depth-first fashion. While it finds a neighbor not yet in the tree, the token moves to this agent, and makes this agent point to the one it comes from. If no new son can be added to the tree, the leader token backtracks until it comes to an agent having neighbor not yet in the tree. If a leader encounters an other leader, one is erased and restart markers are produced to erase the trees already built. The construction then starts again with one less leader. It will eventually succeed when the number of leaders will reach one (by the fairness assumption). The coloring and the spanning tree construction being simultaneous, if an agent part of a tree is recolored, a *restart* marker is produced.

Eventually, the coloring will be stable, only one leader token will make the depth-first search, and a spanning tree will be built on the graph.  $\square$

The class  $\text{ZPSPACE}(s)$  corresponds to language accepted by (so-called) Zero-error Probabilistic Turing machines using a memory space bounded by  $s$ .

Equivalently, (and we will use this definition), it can be shown  $\text{ZPSPACE}(s)$  corresponds to the class of languages accepted by a non-deterministic Turing machine using a memory space  $s$ , with three special internal states *Yes*, *No* and *Don't Know* verifying the following property: if  $\omega \in L$  (resp.  $\omega \notin L$ ) then a possible execution of the machine on  $\omega$  can only output *Yes* or *Don't Know* (resp. *No* or *Don't Know*). And for every input, there are executions outputting a firm answer (*Yes* or *No*). The construction of this machine is described in [10]. The class  $\text{ZPSPACE}_{\text{sym}}(s)$  is the restriction of the symmetric languages, that is to say the language  $L$  such that for any permutation  $\pi$ ,  $\omega \in L$  iff  $\omega' \in L$  where  $\omega'[i] = \omega[\pi(i)]$ .

**Proposition 3.** *Let  $(G_n)$  be a graph family. If there is a population protocol on  $(G_n)$  that organizes the agents into a spanning tree, then there is a population protocol that model a Turing machine tape with a tape of size  $n$  on  $G_n$ .*

*Proof.* Once a spanning tree is built, the spanning tree can be used as the tape of the Turing machine: we can order the vertices of the graph using the spanning tree (using a breadth-first search for instance). The numeration gives the location of the agent in the tape.

Let  $L \in \text{ZPSPACE}_{\text{sym}}(n)$ . Using the tools from above, we can model the behavior of a Turing machine computing  $L$ . For any input  $x$ , the computation outputs either a firm answer, either *Don't Know*. Whenever the protocol ends a simulation, it starts a new one. The first simulations of the Turing machine may be made when the routine that organize the population into a tape is not finished. Those simulations gives answers with no warranty. The fairness makes certain though that the population will eventually be organized, after what exact simulation can occur. The looping simulation ensures that configurations corresponding to exact simulation of the Turing machine are always reachable. Then fairness guarantees every correct simulation will occur during any execution of the population protocol. By fairness, some execution giving the firm answer will occur. And from this instant the simulation will only answer either the correct and firm answer, either *Don't Know*. By outputting the last firm answers computed by a simulation, the protocol can decide “ $x \in L$ ?”.  $\square$

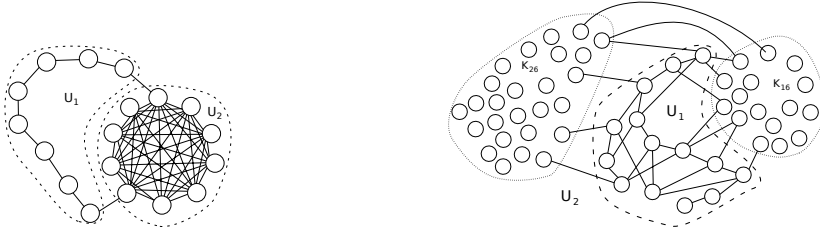
## 4 Separable Graph Family

**Definition 3.** *A graph family  $(G(n) = (V(n), E(n)))$  is  $(s, d)$ -separable if for all  $n$ , the set of vertices  $V(n)$  can be partitioned into  $U_1$  and  $U_2$  ( $U_2$  may be empty) such that:*

- $(U_1, E(n)|_{U_1})$  is a connected graph with  $s(n)$  vertices
- for all vertices  $u \in U_1$ , its degree (in  $G(n)$ ) is bounded by  $d(n)$
- for all vertices  $v \in U_2$ ,  $v$  belongs to a clique of size at least  $2^{d(n)+1}$ .

Notice that there are such separable graph families only if  $d(n) \leq \log n$  and  $s(n) \leq n$ .

For instance with  $d(n) = 2$ , a path of size  $s(n)$  connected (in one or two vertices) to a clique of size at least 8 provides an  $(s, 2)$ -separable family of graphs.



A graph of an  $(s, 2)$ -separable family    A graph of an  $(s, 4)$ -separable family

### 4.1 Recognition of the Bounded Degree Part

We will construct a simulation of Turing machine on separable graphs. The first step is to recognize the cliques of the graphs.

**Lemma 1.** *Let be  $G = (V, E)$  an  $(s, d)$ -separable graph with  $d = O(1)$ . There is a population protocol *Clique* that distinguishes  $U_1$  from  $U_2$ .*

*That means its set of states can be partitioned in  $Q_{clique}$  and  $Q_{bounded}$ , and for any fair execution  $(C_t)$  of the protocol *Clique*, there is  $t_0$  such that for all configurations  $C_t$  with  $t \geq t_0$ , an agent has a state in  $Q_{clique}$  iff it belongs to  $U_2$ .*

*Proof.* The construction of this protocol uses two routines.

*Color<sub>d+1</sub>*, the first routine, colors the agent with  $d + 1$  colors in such a way that an agent has every color in its neighborhood if it belongs to a clique of size at least  $2^{d+1}$ .

*Count<sub>d+1</sub>*, the second routine, counts (up to  $d + 1$ ) how many different colors there are in the neighborhood of any agent, and therefore decides if the agent is in the bounded degree sub-graph (where an agent can not have  $d + 1$  different colors in its neighborhood) or in the clique (where the agents have at least  $d + 1$  colors in their neighborhood). □

## 5 Modeling a Turing Machine on Separable Graph Family

### 5.1 Counting Protocol

To achieve the simulation we need a protocol counting the input letters, the objective is to have the input into a format the Turing machine can use. The idea is to use the bounded-degree sub-graph as a tape to store the entry in a more efficient form. Indeed, the protocol input is symmetric and can be describe by the number of symbols of each type without losing any information. With a tape, we can write those numbers in binary because the agents are implicitly ordered. Of course, this is possible only if this “tape” is big enough with respect to the total number of agents.

**Lemma 2.** *Let be  $G$  an  $(s, d)$ -separable graph with  $s(n) = \Omega(\log(n))$ . There is a population protocol *WriteInput* which organizes the bounded degree part as a Turing machine tape and which writes (using binary encoding) the initial input multiset on this tape.*

*Proof.* The protocol *WriteInput* uses different routines.

First, *Clique* (from Lemma 1) recognizes the bounded degree sub-graph. Then the routines described in the Propositions 1 and 2 organize the bounded degree sub-graph as a Turing machine tape.

The next routine writes the multiset corresponding to the input (in binary notation) on the tape.

This can be done only if the tape has enough cells, that is to say if there are at least  $\Omega(\log n)$  cells on the tape.  $\square$

## 5.2 Main Result of This Section

Let  $L(s)$  be the class of languages computed by population protocols on  $(s, d)$ -separable graph family with  $d = O(1)$ .

If  $s(n) = \Omega(\log n)$ , using the routine *WriteInput* to prepare the population, we can simulate a Turing machine on the bounded degree part. The Turing machine we model uses at most  $s(n)$  cells on its working tape and the result of any computation does not change if we apply any permutation on the input. In fact, we prove that we model any Turing machine that computes a language of the class  $\text{NSPACE}_{\text{sym}}(s)$  using same principles.

**Theorem 1.**  $\forall s(n) = \Omega(\log n), L(s) = \text{NSPACE}_{\text{sym}}(s)$

*Proof.* Fix some  $s(n) = \Omega(\log n)$  and  $d = O(1)$ .

First we prove  $L(s) \subseteq \text{NSPACE}_{\text{sym}}(s)$ .

A configuration of a population of size  $n$  on an  $(s, d)$ -separable graph needs  $O(s(n) + \log(n - s(n)))$  cells on a tape to be stored. Since  $s(n) = \Omega(\log n)$ , we only need  $O(s(n))$  cells. To compute the result of a computation by a population protocol we need to find a reachable configuration such that we cannot find any other reachable configuration where the output is different or undefined. Then it is computable by a Turing machine of  $\text{NSPACE}(s)$ , since  $\text{NSPACE}(s) = \text{coNSPACE}(s)$  for  $s(n) = \Omega(\log n)$  (see e. g. [10]). And the languages of  $L(s)$  are symmetric then  $L(s) \subseteq \text{NSPACE}_{\text{sym}}(s)$ .

We prove now  $\text{ZPSPACE}_{\text{sym}}(s) \subseteq L(s)$ .

Let  $M$  be a machine corresponding to a language of  $\text{ZPSPACE}_{\text{sym}}(s)$ . We can construct a population protocol of  $L(s)$  computing the same language. As direct application of the Lemma 2, we are able to model a Turing machine using a tape with  $s(n)$  cells. The proof of the Proposition 3 can be used to prove that the simulated Turing machines can compute languages of the class  $\text{ZPSPACE}_{\text{sym}}(s)$ . Therefore  $\text{ZPSPACE}_{\text{sym}}(s) \subseteq L(s)$

As  $s(n) = \Omega(\log n)$ , it is known (see e. g. [10]) that we have  $\text{NSPACE}_{\text{sym}}(s) = \text{ZPSPACE}_{\text{sym}}(s)$ .

To conclude:  $L(s) = \text{NSPACE}_{\text{sym}}(s) = \text{ZPSPACE}_{\text{sym}}(s)$   $\square$

### 5.3 Passively Mobile Machines Model

Let  $s, s'$  functions such that  $s(n) = O(n)$ . We will consider the passively mobile machines model (i. e. population protocol where the agents are Turing machines) with memory of the agents bounded by  $s'$ .

In [7], a detailed proof of the following result can be found.

**Proposition 4.** *Let  $s'(n) = \Omega(\log n)$ . The set of languages computed by the passively mobile machines model with memory of the agents bounded by  $s'$  is exactly  $\text{NSPACE}_{\text{sym}}(n \cdot s'(n))$ .*

A protocol assigning a unique identifier to every agent if they have a logarithmic memory can be devised.

In other words, if the agents have at least a logarithmic memory space, then whatever the interaction graph is, the model is equivalent to a Turing machine with a memory space equal to all the possible available space. If the agents have less memory space, considering the interaction graph may permit to gain some computational power. So we will now only consider cases where  $s'(n) = o(\log n)$ .

Let  $L_{s'}(s)$  be the class of languages computed by the passively mobile machines model with memory of the agents bounded by  $s'$  on  $(s, d)$ -separable graph family with  $d(n) = 2^{O(s'(n))}$ . We remind that  $d(n) \leq \log n$  in any case.

The results of the previous sections can be extended to the population protocol model where the agents are not finite states, but Turing machines with memory bounded by  $s'$ . Then the agent can have up to  $2^{O(s'(n))}$  different “states” (in fact they are configurations). So we can use those extra states to construct distance-2-coloring with more colors, and hence we may be able to construct a spanning tree on  $(s, d)$ -separable graphs for non constant  $d$ . More precisely we have the following result that extends Lemma 1.

**Lemma 3.** *Let be  $G = (V, E)$  an  $(s, d)$ -separable graph. There is a passively mobile machines model with memory bounded by  $\log(d(n))$  that distinguish  $U_1$  from  $U_2$ .*

It gives a way to model a Turing machine on the bounded degree sub-graph. The modeled Turing machine has a tape modeled on  $s(n)$  agents, each one having  $s'(n)$  cells; then the total space the Turing machine have is  $s(n) \cdot s'(n)$  (each of the  $s(n)$  agents contribute with  $s'(n)$  cells). And then the generalisation of the Proposition 1 is the following.

**Theorem 2.** *Let  $d, s$  and  $s'$  such that  $d(n) = 2^{O(s'(n))}$ ,  $d(n) = O(\log n)$ ,  $s'(n) = o(\log n)$  and  $s(n) \leq n$ . The class of the language computable by passively mobile machines of  $\text{DSPACE}(s')$  on  $(s, d)$ -separable graph family is  $\text{NSPACE}_{\text{sym}}(s \cdot s')$ .*

$$L_{s'}(s) = \text{NSPACE}_{\text{sym}}(s \cdot s')$$

### 5.4 A Hierarchy

With Theorems 1 and 2, we can transpose the hierarchy of non-deterministic Turing machines above the logarithmic space.

**Theorem 3.** *Let  $s_1, s_2, s'_1, s'_2$  such that  $s_i(n) = O(n)$  and  $s'_i(n) = O(\log \log n)$  for  $i = 1, 2$ . If  $s'_1 \cdot s_1(n) = o(s'_2 \cdot s_2(n))$  then  $L_{s'_1}(s_1) \subseteq L_{s'_2}(s_2)$ . And the inequality is strict if  $s'_2(n) \cdot s_2(n) = \Omega(\log n)$ .*

For instance, let be  $s'(n) = o(\log n)$  and consider the sequence of function  $\log^k n/s'(n)$ . If  $k \geq 1$ , we have  $L_{s'}(\log^k n/s'(n)) = \text{NSPACE}_{\text{sym}}(\log^k n)$ . Therefore, we have the following hierarchy with:

$$\begin{aligned} L_1(1) &\subsetneq L_{s'}(1) \\ &\subsetneq L_{s'}(\log n/s'(n)) \\ &\subsetneq L_{s'}(\log^2 n/s'(n)) \\ &\vdots \\ &\subsetneq L_{s'}(\log^k n/s'(n)) \\ &\vdots \\ &\subsetneq L_{s'}(n) = \text{NSPACE}_{\text{sym}}(n \cdot s'(n)) \end{aligned}$$

$L_1(1)$  corresponds to what is computed by the classical model with finite automaton agents and the complete interaction graph:  $L_1(1)$  is the class of the semi-linear languages.  $L_{s'}(1)$  corresponds to the passively mobile machines model where agents use  $O(s')$  memory and the complete interaction graph.  $L_{s'}(n)$  corresponds to the passively mobile machines model where agents use  $O(s')$  memory and the bounded degree interaction graph. The proof of the strict inequality  $L_1(1) \subsetneq L_{s'}(1)$  is detailed in [7].

Note that in [7], a hierarchy is built by using only the memory available on the agents. Their hierarchy can not be exactly characterized if  $s'(n) = o(\log n)$  and collapsed if  $s'(n) = o(\log \log n)$ . Using restrictions on the interaction graph, we are able to construct a more accurate hierarchy.

## 6 Conclusion

In this paper we determine the computational power of a model of population protocol working on different kind of interaction graph families and with different kind of restrictions on the computational power of the agent. The classical population protocols are the case where the interaction graph is complete and where the agents are finite automata. The case where the interaction graph is a uniformly bounded degree graph and the agents are finite automata has the same computational power as linear space non deterministic Turing machines. With intermediate interaction graphs and the agents having more memory allowed, the model is equivalent to non-deterministic Turing machine working with some bounded memory. We have constructed a Turing machine simulation with as much memory as the sum of all the memory of the agents of a “good” sub-graph. By varying the “good” sub-graph and the bound over the memory of the agents, we disserted a hierarchy. This hierarchy is inherited from the one over the non-deterministic Turing machines.

## References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, pp. 290–299. ACM (2004)
2. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. *Distributed Computing* 18(4), 235–253 (2006)
3. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. *Distributed Computing* 20(4), 279–304 (2007)
4. Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably computable properties of network graphs. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 63–74. Springer, Heidelberg (2005)
5. Angluin, D., Aspnes, J., Fischer, M.J., Jiang, H.: Self-stabilizing population protocols, vol. 3, p. 13 (November 2008)
6. Aspnes, J., Ruppert, E.: An introduction to population protocols. *Bulletin of the EATCS* 93, 106–125 (2007)
7. Chatzigiannakis, I., Michail, O., Nikolaou, S., Pavlogiannis, A., Spirakis, P.G.: Passively mobile communicating machines that use restricted space. *Theoretical Computer Science* (2011)
8. Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Ruppert, E.: When birds die: Making population protocols fault-tolerant. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 51–66. Springer, Heidelberg (2006)
9. Guerraoui, R., Ruppert, E.: Names trump malice: Tiny mobile agents can tolerate byzantine failures, pp. 484–495 (2009)
10. Saks, M.: Randomization and derandomization in space-bounded computation. In: Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity, pp. 128–149. IEEE (1996)

# Spectral Representation of Some Computationally Enumerable Sets with an Application to Quantum Provability\*

Cristian S. Calude<sup>1,\*\*</sup> and Kohtaro Tadaki<sup>2,\*\*\*</sup>

<sup>1</sup> Department of Computer Science, University of Auckland, Auckland, New Zealand  
cristian@cs.auckland.ac.nz

<sup>2</sup> Research and Development Initiative, Chuo University, Tokyo, Japan  
tadaki@kc.chuo-u.ac.jp

**Abstract.** We propose a new type of quantum computer which is used to prove a spectral representation for a class  $\mathcal{S}$  of computable sets. When  $S \in \mathcal{S}$  codes the theorems of a formal system, the quantum computer produces through measurement all theorems and proofs of the formal system. We conjecture that the spectral representation is valid for all computably enumerable sets. The conjecture implies that the theorems of a general formal system, like Peano Arithmetic or ZFC, can be produced through measurement; however, it is unlikely that the quantum computer can produce the proofs as well, as in the particular case of  $\mathcal{S}$ . The analysis suggests that showing the provability of a statement is different from writing up the proof of the statement.

## 1 Introduction

Mathematical results are accepted only if they have been proved: *the proof concludes with the proven statement, the theorem*. The proof comes first and justifies the theorem. Classically, there is no alternative scenario.

The genius mathematician Srinivasa Ramanujan discovered nearly 3900 results [2], many without proofs; nearly all his claims have been proven correct. Ramanujan first *recognised* a true statement and only later that statement was *proven*, hence accepted as a *theorem*. While we don't know how Ramanujan's mind was able to "discover" mathematical true facts, we can ask whether there is a way to understand, and possibly imitate, his approach.

In this paper a new type of quantum computer is used to prove a spectral representation for a class  $\mathcal{S}$  of computable sets is proved. For every  $S \in \mathcal{S}$  we construct a quantum system in such a way that the elements of  $S$  are exactly the eigenvalues of the Hermitian operator representing an observable of the quantum system, i.e. the spectrum of the operator. In particular,  $S$  can be represented by the energy of the associated quantum system. The operator associated to

---

\* Supported by JSPS KAKENHI Grant Number 23650001.

\*\* Work done in part during a visit to Research and Development Initiative, Chuo University, Tokyo, Japan, January 2013; partially supported also by Marie Curie FP7-PEOPLE-2010-IRSES Grant RANPHYS.

\*\*\* Corresponding author.



$S \in \mathcal{S}$  has a special numerical form which guarantees that by measurement we get both the element and the proof that the element is in  $S$ . *We conjecture that the spectral representation is valid for all computably enumerable sets.*

When  $S \in \mathcal{S}$  codes the theorems of a formal system, then the associated quantum computer produces through measurement the theorems of the formal system and their proofs. The conjecture implies that every theorem of a general (recursively axiomatisable) formal system, like Peano Arithmetic or ZFC, can be produced through measurement. However, we argue that in this general case the quantum procedure produces, like Ramanujan, only the true the statement, but not its proof. Of course, the proof can be algorithmically generated by a classical algorithm, albeit in a possibly very long time (such a computation makes sense only for statements recognised as “interesting”). For example, if the Riemann hypothesis is produced by the quantum procedure we will know that the famous hypothesis is true. However, to have a formal proof—whose existence is guaranteed by the correctness of the quantum procedure—we may need to run a very long classical procedure. The proof obtained in this way could be rather unsatisfactory, as it may not convey the “understanding”, the reason for which the Riemann hypothesis holds true (see also [4]). Although such a proof may not make us “wiser” [1], it may stimulate the search for better arguments.

The paper is structured as follows. In Section 2 we present the basic quantum mechanical facts necessary for describing our quantum systems. In Section 3 we describe a class of computable sets for which we can prove in Section 4 the representability theorem and its application to quantum provability (in Section 5). In Section 6 we discuss the generalisation of the quantum procedure to all computably enumerable sets and in Section 7 its application to quantum provability for arbitrary formal systems.

## 2 Quantum Mechanical Facts

We start with some basic facts on quantum mechanics needed for this paper. The quantum mechanical arguments are presented at the level of mathematical rigour adopted in quantum mechanics textbooks written by physicists, for example, Dirac [5] and Mahan [8].

A state of a quantum system is represented by a vector in a Hilbert space  $\mathcal{H}$ . The vector and the space are called *state vector* and *state space*, respectively. The *dynamical variables* of a system are quantities such as the coordinates and the components of momentum and angular momentum of particles, and the energy of the system. They play a crucial role not only in classical mechanics but also in quantum mechanics. Dynamical variables in quantum mechanics are represented by Hermitian operators on the state space  $\mathcal{H}$ . A dynamical variable of the system is called an *observable* if all eigenvectors of the Hermitian operator representing it form a complete system for  $\mathcal{H}$ . Normally we assume that a measurement of any observable can be performed upon a quantum system in any state (if we ignore the constructive matter, which is one of the points of this paper).

The set of possible outcomes of a measurement of an observable  $\mathcal{O}$  of a system is the eigenvalue spectrum of the Hermitian operator representing  $\mathcal{O}$ .

Let  $\{|m, \lambda\rangle\}$  be a complete orthonormal system of eigenvectors of the Hermitian operator  $A$  representing an observable  $\mathcal{O}$  such that  $A|m, \lambda\rangle = m|m, \lambda\rangle$  for all eigenvalues  $m$  of  $A$  and all  $\lambda$ , where the parameter  $\lambda$  designates the degeneracy of the eigenspace of  $A$ . Suppose that a measurement of  $\mathcal{O}$  is performed upon a quantum system in the state represented by a normalized vector  $|\Psi\rangle \in \mathcal{H}$ . Then the probability of getting the outcome  $m$  is given by  $p(m) = \sum_{\lambda} |\langle m, \lambda | \Psi \rangle|^2$ , where  $\langle m, \lambda | \Psi \rangle$  denotes the inner product of the vectors  $|m, \lambda\rangle$  and  $|\Psi\rangle$ . Moreover, given that the outcome  $m$  occurred, the state of the quantum system immediately after the measurement is represented by the normalized vector

$$\frac{1}{\sqrt{p(m)}} \sum_{\lambda} \langle m, \lambda | \Psi \rangle |m, \lambda\rangle.$$

The *commutator* between two operators  $A$  and  $B$  is defined to be  $[A, B] := AB - BA$ . Let  $\mathcal{O}_1, \dots, \mathcal{O}_k$  be observables of a quantum system and let  $A_1, \dots, A_k$  be the Hermitian operators which represent  $\mathcal{O}_1, \dots, \mathcal{O}_k$ , respectively. If the Hermitian operators commute to each other, i.e.,  $[A_j, A_{j'}] = 0$  for all  $j, j' = 1, \dots, k$ , then we can perform measurements of all  $\mathcal{O}_1, \dots, \mathcal{O}_k$  simultaneously upon the quantum system in any state. All dynamical variables which we will consider below are assumed to be observables, and we will identify any observable with the Hermitian operator which represents it.

In this paper we consider quantum systems consisting of vibrating particles. The simplest one is the quantum system of *one-dimensional harmonic oscillator*, which consists only of one particle vibrating in one-dimensional space. The dynamical variables needed to describe the system are just one coordinate  $x$  and its conjugate momentum  $p$ . The *energy* of the system is an observable, called *Hamiltonian*, and is defined in terms of  $x$  and  $p$  by

$$H = \frac{1}{2m}(p^2 + m^2\omega^2x^2),$$

where  $m$  is the mass of the oscillating particle and  $\omega$  is  $2\pi$  times the frequency. The oscillation of the particle is quantized by the *fundamental quantum condition*

$$[x, p] = i\hbar, \tag{1}$$

where  $\hbar$  is *Planck's constant*. The *annihilation operator*  $a$  of the system is defined by

$$a = \sqrt{\frac{m\omega}{2\hbar}} \left( x + \frac{ip}{m\omega} \right).$$

Its adjoint  $a^\dagger$  is called a *creation operator*. The fundamental quantum condition (1) is then equivalently rewritten as

$$[a, a^\dagger] = 1, \tag{2}$$

and the Hamiltonian can be represented in the form

$$H = \hbar\omega \left( a^\dagger a + \frac{1}{2} \right) \tag{3}$$

in terms of the creation and annihilation operators. In order to determine the

values of energy possible in the system, we must solve the eigenvalue problem of  $H$ . This problem is reduced to the eigenvalue problem of the observable  $N := a^\dagger a$ , called a *number operator*. Using the condition (2), the eigenvalue spectrum of  $N$  is shown to equal the set  $\mathbb{N}$  of all nonnegative integers. Each eigenspace of  $N$  is not degenerate, and the normalized eigenvector  $|n\rangle$  of  $N$  belonging to an arbitrary eigenvalue  $n \in \mathbb{N}$  is given by

$$|n\rangle = \frac{(a^\dagger)^n}{\sqrt{n!}}|0\rangle, \quad (4)$$

where  $|0\rangle$  is the unique normalized vector up to a phase factor such that  $a|0\rangle = 0$ . Since  $N$  is an observable, the eigenvectors  $\{|n\rangle\}$  forms a complete orthonormal system for the state space. It follows from (3) that the values of energy possible in the system are

$$E_n = \hbar\omega \left( n + \frac{1}{2} \right), \quad (n = 0, 1, 2, \dots)$$

where the eigenvector of  $H$  belonging to an energy  $E_n$  is given by (4).

Next we consider the quantum system of *k-dimensional harmonic oscillators* which consists of  $k$  one-dimensional harmonic oscillators vibrating independently without no interaction. The dynamical variables needed to describe the system are  $k$  coordinates  $x_1, \dots, x_k$  and their conjugate momenta  $p_1, \dots, p_k$ . The Hamiltonian of the system is

$$H = \sum_{j=1}^k \frac{1}{2m_j} (p_j^2 + m_j^2 \omega_j^2 x_j^2), \quad (5)$$

where  $m_j$  is the mass of the  $j$ th one-dimensional harmonic oscillator and  $\omega_j$  is  $2\pi$  times its frequency. The vibrations of  $k$  oscillators are quantized by the fundamental quantum conditions

$$[x_j, p_{j'}] = i\hbar\delta_{jj'}, \quad [x_j, x_{j'}] = [p_j, p_{j'}] = 0. \quad (6)$$

The annihilation operator  $a_j$  of the  $j$ th oscillator is defined by

$$a_j = \sqrt{\frac{m_j\omega_j}{2\hbar}} \left( x_j + \frac{ip_j}{m_j\omega_j} \right).$$

The adjoint  $a_j^\dagger$  of  $a_j$  is the creation operator of the  $j$ th oscillator. The fundamental quantum condition (6) is then equivalently rewritten as

$$[a_j, a_{j'}^\dagger] = \delta_{jj'}, \quad (7)$$

$$[a_j, a_{j'}] = [a_j^\dagger, a_{j'}^\dagger] = 0. \quad (8)$$

and the Hamiltonian can be represented in the form

$$H = \sum_{j=1}^k \hbar\omega_j \left( N_j + \frac{1}{2} \right) \quad (9)$$

where  $N_j := a_j^\dagger a_j$  is the number operator of the  $j$ th oscillator. In order to determine the values of energy possible in the system, we first solve the eigenvalue problems of the number operators  $N_1, \dots, N_k$ . We can do this simultaneously

for all  $N_j$  since the number operators commute to each other, i.e.,  $[N_j, N_{j'}] = 0$  for all  $j, j' = 1, \dots, k$ , due to (7) and (8). The eigenvalue spectrum of each  $N_j$  is shown to equal  $\mathbb{N}$  using (7). We define a vector  $|n_1, \dots, n_k\rangle$  as the tensor product  $|n_1\rangle \otimes \dots \otimes |n_k\rangle$  of  $|n_1\rangle, \dots, |n_k\rangle$ , where each  $|n_j\rangle$  is defined by (4) using  $a_j$  in place of  $a$ . For each  $j$ , the vector  $|n_1, \dots, n_k\rangle$  is a normalized eigenvector of  $N_j$  belonging to an eigenvalue  $n_j \in \mathbb{N}$ , i.e.,

$$N_j |n_1, \dots, n_k\rangle = n_j |n_1, \dots, n_k\rangle. \quad (10)$$

All the vectors  $\{|n_1, \dots, n_k\rangle\}$  form a complete orthonormal system for the state space. It follows from (9) that the values of energy possible in the system are

$$E_{n_1, \dots, n_k} = \hbar \sum_{j=1}^k \omega_j \left( n_j + \frac{1}{2} \right), \quad (n_1, \dots, n_k = 0, 1, 2, \dots)$$

The vector  $|n_1, \dots, n_k\rangle$  is an eigenvector of  $H$  belonging to an energy  $E_{n_1, \dots, n_k}$ .

The Hamiltonian (5) describes the quantum system of  $k$ -dimensional harmonic oscillators where each oscillator does not interact with any others and moves independently. In a general quantum system consisting of  $k$ -dimensional harmonic oscillators, each oscillator strongly interacts with all others. Its Hamiltonian has the general form

$$P(a_1, \dots, a_k, a_1^\dagger, \dots, a_k^\dagger), \quad (11)$$

where  $a_1, \dots, a_k$  are creation operators satisfying the quantum conditions (7) and (8), and  $P$  is a polynomial in  $2k$  variables with coefficients of complex numbers such that (11) is Hermitian.<sup>1</sup> For example, we can consider the quantum system of  $k$ -dimensional harmonic oscillators whose Hamiltonian is

$$H = \sum_j \hbar \omega_j \left( a_j^\dagger a_j + \frac{1}{2} \right) + \sum_{j \neq j'} g_{jj'} a_j^\dagger a_{j'}.$$

Here the *interaction terms*  $g_{jj'} a_j^\dagger a_{j'}$  between the  $j$ th oscillator and the  $j'$ th oscillator with a real constant  $g_{jj'}$  are added to the Hamiltonian (9). Note, however, that solving exactly the eigenvalue problem of an observable in the general form of (11) is not an easy task.

### 3 A Class of Unary Languages

In this section we introduce a class of unary languages for which the representability theorem proven in the next section holds true.

Let  $\mathbb{N}^*$  be the set of all finite sequences  $(x_1, \dots, x_m)$  with elements in  $\mathbb{N}$  ( $m \in \mathbb{N}$ ; for  $m = 0$  we get the empty sequence  $\varepsilon$ ). Let

$$L((x_1 \dots x_m), a) = \left( \prod_{i=1}^m \{1^{x_i}\}^* \right) \{1^a\}, \quad (12)$$

for all  $(x_1, \dots, x_m) \in \mathbb{N}^*$ ,  $a \in \mathbb{N}$ .

<sup>1</sup> In the monomials appearing in  $P$ , the order of the variables  $x_1, \dots, x_{2k}$  does not matter. However, since  $a_j$  and  $a_j^\dagger$  do not commute, in substituting  $a_1, \dots, a_k, a_1^\dagger, \dots, a_k^\dagger$  into the variables of  $P$  the order of these operators makes a difference. Thus, the operator (11) makes sense only by specifying this order.

**Theorem 1.** *Let  $\mathcal{L}_0$  be the minimal class of languages  $\mathcal{L}$  over  $\{1\}$  containing the languages  $\{1^n\}$  for every  $n \in \mathbb{N}$ , and which is closed under concatenation and the Kleene star operation. Then,  $\mathcal{L}_0 = \{L((x_1, \dots, x_m), a) \mid (x_1, \dots, x_m) \in \mathbb{N}^*, a \in \mathbb{N}\}$ .*

*Proof.* The class  $\mathcal{L}_0$  has the required properties because  $L(\varepsilon, a) = \{1^a\}$ , the concatenation of  $L((x_1, \dots, x_m), a)$  and  $L((y_1, \dots, y_l), b)$  is  $L((x_1, \dots, x_m), a)L((y_1, \dots, y_l), b) = L((x_1, \dots, x_m, y_1, \dots, y_l), a + b)$  and the Kleene star of  $L((x_1, \dots, x_m), a)$  is  $L((x_1, \dots, x_m), a)^* = L((x_1, \dots, x_m), a, 0)$ . In view of (12),  $\mathcal{L}_0$  is included in every class  $\mathcal{L}$  satisfying the properties in the statement of the theorem.  $\square$

**Corollary 2.** *The class  $\mathcal{L}_0$  coincides with the minimal class of languages  $\mathcal{L}$  over  $\{1\}$  which contains the languages  $\{1^n\}$  and  $\{1^n\}^*$ , for every  $n \in \mathbb{N}$  and which is closed under concatenation.*

**Comment 3.** *i) If  $L$  is a finite unary language with more than one element, then  $L \notin \mathcal{L}_0$ .*

*ii) The family  $\mathcal{L}_0$  is a proper subset of the class of regular (equivalently, context-free) languages.*

*iii) The language  $\{1^p \mid p \text{ is prime}\}$  is not in  $\mathcal{L}_0$ .*

Consider the minimal class  $\mathcal{D}_0$  of subsets of  $\mathbb{N}$  containing the sets  $\{b\}$ , for every  $b \in \mathbb{N}$ , and which is closed under the sum and the Kleene star operation. Here the sum of the sets  $S, T$  is the set  $S + T = \{a + b \mid a \in S, b \in T\}$ ; the Kleene star of the set  $S$  is the set  $S^* = \{a_1 + a_2 + \dots + a_k \mid k \geq 0, a_i \in S, 1 \leq i \leq k\}$ .

**Theorem 4.** *The following equality holds true:  $\mathcal{L}_0 = \{\{1^a \mid a \in S\} \mid S \in \mathcal{D}_0\}$ .*

Based on the above theorem, we identify  $\mathcal{L}_0$  with  $\mathcal{D}_0$  in what follows.

## 4 The Representation Theorem

Can a set  $S \in \mathcal{D}_0$  be represented as the outcomes of a quantum measurement? We answer this question in the affirmative. First we show that the sets in  $\mathcal{D}_0$  can be generated by polynomials with nonnegative integer coefficients.

**Proposition 5.** *For every set  $S \in \mathcal{D}_0$  there exists a polynomial with nonnegative integer coefficients  $F_S$  in variables  $x_1, \dots, x_k$  such that  $S$  can be represented as:*

$$S = \{F_S(n_1, \dots, n_k) \mid n_1, \dots, n_k \in \mathbb{N}\}. \quad (13)$$

*Proof.* Suppose that  $S \in \mathcal{D}_0$ . It follows from Theorem 4 and (12) that there exist  $a_1, \dots, a_k, a \in \mathbb{N}$  such that  $S = \{a_1 n_1 + \dots + a_k n_k + a \mid n_1, \dots, n_k \in \mathbb{N}\}$ . Thus, (13) holds for the polynomial  $F_S(x_1, \dots, x_k) = a_1 x_1 + \dots + a_k x_k + a$ .  $\square$

**Comment 6.** *There exist infinitely many sets not in  $\mathcal{D}_0$  which are representable in the form (13).*

Motivated by Proposition 5, we show that every set

$$S = \{F(n_1, \dots, n_k) \mid n_1, \dots, n_k \in \mathbb{N}\}, \quad (14)$$

where  $F$  is a polynomial in  $k$  variables with nonnegative integer coefficients, can be represented by the set of outcomes of a *constructive* quantum measurement. For this purpose, we focus on a quantum system consisting of  $k$ -dimensional harmonic oscillators whose Hamiltonian has the form

$$H = F(N_1, \dots, N_k), \quad (15)$$

where  $N_1, \dots, N_k$  is the number operators defined by  $N_j = a_j^\dagger a_j$  with the annihilation operator  $a_j$  of the  $j$ th oscillator. Note that the substitution of  $N_1, \dots, N_k$  into the variables of  $F$  is unambiguously defined since the number operators  $N_1, \dots, N_k$  commute to each other.

We say an observable of the form (11) is *constructive* if all coefficients of  $P$  are in the form of  $p + qi$  with  $p, q \in \mathbb{Q}$ . Thus, the Hamiltonian (15) is constructive by definition. Actually, a measurement of the Hamiltonian (15) can be performed *constructively* in an intuitive sense. The constructive measurement consists of the following two steps: First, the simultaneous measurements of the number operators  $N_1, \dots, N_k$  are performed upon the quantum system to produce the outcomes  $n_1, \dots, n_k \in \mathbb{N}$  for  $N_1, \dots, N_k$ , respectively. This is possible since the number operators commute to each other. Secondly,  $F(n_1, \dots, n_k)$  is calculated and is regarded as the outcome of the measurement of the Hamiltonian (15) itself. This is constructively possible since  $F$  is a polynomial with integer coefficients. Thus, the whole measurement process is constructive in an intuitive sense too.

**Theorem 7.** *For every set  $S$  of the form (14) there exists a constructive Hamiltonian  $H$  such that the set of all possible outcomes of a measurement of  $H$  is  $S$ .*

*Proof.* Consider the Hamiltonian  $H$  of the form (15). It is constructive, as we saw above. We show that the eigenvalue spectrum of  $H$  equals to  $S$ .

First, using (10) we get

$$F(N_1, \dots, N_k)|n_1, \dots, n_k\rangle = F(n_1, \dots, n_k)|n_1, \dots, n_k\rangle \quad (16)$$

for every  $n_1, \dots, n_k \in \mathbb{N}$ . Thus, every element of  $S$  is an eigenvalue of  $H$ . Conversely, suppose that  $E$  is an arbitrary eigenvalue of  $H$ . Then there exists a nonzero vector  $|\Psi\rangle$  such that  $H|\Psi\rangle = E|\Psi\rangle$ . Since all vectors  $\{|n_1, \dots, n_k\rangle\}$  form a complete orthonormal system for the state space, there exist complex numbers  $\{c_{n_1, \dots, n_k}\}$  such that  $|\Psi\rangle = \sum_{n_1, \dots, n_k} c_{n_1, \dots, n_k} |n_1, \dots, n_k\rangle$ . It follows from (16) that

$$\sum_{n_1, \dots, n_k} c_{n_1, \dots, n_k} F(n_1, \dots, n_k) |n_1, \dots, n_k\rangle = \sum_{n_1, \dots, n_k} c_{n_1, \dots, n_k} E |n_1, \dots, n_k\rangle.$$

Since the vectors  $\{|n_1, \dots, n_k\rangle\}$  are independent, we have

$$c_{n_1, \dots, n_k} (E - F(n_1, \dots, n_k)) = 0, \quad (17)$$

for all  $n_1, \dots, n_k \in \mathbb{N}$ . Since  $|\Psi\rangle$  is nonzero,  $c_{\bar{n}_1, \dots, \bar{n}_k}$  is also nonzero for some  $\bar{n}_1, \dots, \bar{n}_k \in \mathbb{N}$ . It follows from (17) that  $E = F(\bar{n}_1, \dots, \bar{n}_k)$ .  $\square$

## 5 An Application to Quantum Provability

Let  $S$  be a set of the form (14). In the proof of Theorem 7, we consider the measurement of the Hamiltonian of the form (15). In the case where the state  $|\Psi\rangle$  over which the measurement of the Hamiltonian is performed is chosen randomly, an element of  $S$  is generated randomly as the measurement outcome. In this manner, by infinitely many repeated measurements we get exactly the set  $S$ .

If the set  $S$  codes the “theorems” of a formal system  $\mathcal{S}$ —which is possible as  $S$  is computable—then  $F(n_1, \dots, n_k) \in S$  is a *theorem* of  $\mathcal{S}$  and the numbers  $n_1, \dots, n_k$  play the role of the *proof* which certifies it.

Suppose that a *single* measurement of the Hamiltonian of the form (15) was performed upon a quantum system in a state represented by a normalized vector  $|\Psi\rangle$  to produce an outcome  $m \in S$ , i.e., a theorem. Then, by the definition of theorems, there exists a proof  $n_1, \dots, n_k$  which makes  $m$  a theorem, i.e., which satisfies  $m = F(n_1, \dots, n_k)$ . Can we extract the proof  $n_1, \dots, n_k$  after the measurement? This can be possible in the following manner: Immediately after the measurement, the system is in the state represented by the normalized vector  $|\Phi\rangle$  given by

$$|\Phi\rangle = \frac{1}{\sqrt{C}} \sum_{m=F(n_1, \dots, n_k)} \langle n_1, \dots, n_k | \Psi \rangle |n_1, \dots, n_k\rangle,$$

where  $C$  is the probability of getting the outcome  $m$  in the measurement given:

$$C = \sum_{m=F(n_1, \dots, n_k)} |\langle n_1, \dots, n_k | \Psi \rangle|^2.$$

Since the number operators  $N_1, \dots, N_k$  commute to each other, we can perform the simultaneous measurements of  $N_1, \dots, N_k$  upon the system in the state  $|\Phi\rangle$ . Hence, by performing the measurements of  $N_1, \dots, N_k$ , we obtain any particular outcome  $n_1, \dots, n_k$  with probability  $|\langle n_1, \dots, n_k | \Phi \rangle|^2$ . Note that

$$\sum_{m=F(n_1, \dots, n_k)} |\langle n_1, \dots, n_k | \Phi \rangle|^2 = \sum_{m=F(n_1, \dots, n_k)} |\langle n_1, \dots, n_k | \Psi \rangle|^2 / C = 1.$$

Thus, with probability one we obtain some outcome  $n_1, \dots, n_k$  such that  $m = F(n_1, \dots, n_k)$ . In this manner we can immediately extract the proof  $n_1, \dots, n_k$  of the theorem  $m \in S$  obtained as a measurement outcome.

## 6 A Conjecture

In the early 1970s, Matijasevič, Robinson, Davis, and Putnam solved negatively Hilbert’s tenth problem by proving the MRDP theorem (see Matijasevič [9] for details) which states that every computably enumerable subset of  $\mathbb{N}$  is Diophantine. A subset  $S$  of  $\mathbb{N}$  is called *computably enumerable* if there exists a (classical) Turing machine that, when given  $n \in \mathbb{N}$  as an input, eventually halts if  $n \in S$  and otherwise runs forever. A subset  $S$  of  $\mathbb{N}$  is *Diophantine* if there exists a polynomial  $P(x, y_1, \dots, y_k)$  in variables  $x, y_1, \dots, y_k$  with

integer coefficients such that, for every  $n \in \mathbb{N}$ ,  $n \in S$  if and only if there exist  $m_1, \dots, m_k \in \mathbb{N}$  for which  $P(n, m_1, \dots, m_k) = 0$ .

Inspired by the MRDP theorem, we conjecture the following:

**Conjecture 8** *For every computably enumerable subset  $S$  of  $\mathbb{N}$ , there exists a constructive observable  $A$  of the form of (11) whose eigenvalue spectrum equals  $S$ .*

Conjecture 8 implies that when we perform a measurement of the observable  $A$ , a member of the computably enumerable  $S$  is stochastically obtained as a measurement outcome. As we indefinitely repeat measurements of  $A$ , members of  $S$  are being enumerated, just like a Turing machine enumerates  $S$ .

In this way a new type of quantum mechanical computer is postulated to exist. How can we construct it? Below we discuss some properties of this hypothetical quantum computer.

As in the proof of the MRDP theorem—in which a whole computation history of a Turing machine is encoded in (the base-two expansions of) the values of variables of a Diophantine equation—a whole computation history of a Turing machine is encoded in a single quantum state which does not make time-evolution (in the Schrödinger picture). Namely, a whole computation history of the Turing machine  $M$  which recognises  $S$  is encoded in an eigenstate of the observable  $A$  which is designed appropriately using the creation and annihilation operators. To be precise, let  $|\Psi\rangle = \sum_{n_1, \dots, n_k} c_{n_1, \dots, n_k} |n_1, \dots, n_k\rangle$  be an eigenvector of  $A$  belonging to an eigenvalue  $n \in S$  such that each coefficient  $c_{n_1, \dots, n_k}$  is drawn from a certain finite set  $\mathcal{C}$  of complex numbers containing 0 and the set  $\{(n_1, \dots, n_k) \mid c_{n_1, \dots, n_k} \neq 0\}$  is finite. The whole computation history of  $M$  with the input  $n$  is encoded in the coefficients  $\{c_{n_1, \dots, n_k}\}$  of  $|\Psi\rangle$  such that each finite subset obtained by dividing appropriately  $\{c_{n_1, \dots, n_k}\}$  represents the configuration (i.e., the triple of the state, the tape contents, and the head location) of the Turing machine  $M$  at the corresponding time step. The observable  $A$  is constructed such that its eigenvector encodes the whole computation history of  $M$ , using the properties of the creation and annihilation operators such as

$$a_j^\dagger |n_1, \dots, n_{j-1}, n_j, n_{j+1}, \dots, n_k\rangle = \sqrt{n_j + 1} |n_1, \dots, n_{j-1}, n_j + 1, n_{j+1}, \dots, n_k\rangle,$$

by which the different time steps are connected in the manner corresponding to the Turing machine computation of  $M$ . In the case of  $n \notin S$ , the machine  $M$  with the input  $n$  does not halt. Consequently, the length of the whole computation history is infinite and therefore the set  $\{(n_1, \dots, n_k) \mid c_{n_1, \dots, n_k} \neq 0\}$  is infinite, which implies, because all coefficients belong to the finite set  $\mathcal{C}$  of complex numbers, that the norm of  $|\Psi\rangle$  is indefinite and hence  $|\Psi\rangle$  is not an eigenvector of  $A$ . In this manner, any eigenvalue of  $A$  is limited to a member of  $S$ .

Note that there are many computation histories of a Turing machine depending on its input. In the proposed quantum mechanical computer, the measurement of  $A$  chooses one of the computation histories stochastically and the input corresponding to the computation history is obtained as a measurement outcome. The above analysis shows that Conjecture 8 is likely to be true.



The main feature of the proposed quantum mechanical computer is that *the evolution of computation does not correspond to the time-evolution of the underlying quantum system*. Hence, in contrast with a conventional quantum computer, the evolution of computation does not have to form a unitary time-evolution, so it is not negatively influenced by *decoherence*<sup>2</sup>, a serious obstacle to the physical realisation of a conventional quantum computer.

Again, in contrast with a conventional quantum computer, this proposed quantum mechanical computer can be physically realisable even as a solid-state device at room temperature (the lattice vibration of solid crystal, i.e., *phonons*), which strongly interacts with the external environment. A member of  $S$  is obtained as a measurement outcome in an instant by measuring the observable  $A$ . For example, in the case when the observable  $A$  is the Hamiltonian of a quantum system, the measurement outcome corresponds to the energy of the system. In this case, we can probabilistically decide—with sufficiently small error probability—whether a given  $n \in \mathbb{N}$  is in  $S$ : the quantum system is first prepared in a state  $|\Psi\rangle$  such that the expectation value  $\langle\Psi|A|\Psi\rangle$  of the measurement of the energy over  $|\Psi\rangle$  is approximately  $n$ , and then the measurement is actually performed. This computation deciding the membership of  $n$  to  $S$  terminates in an instant if sufficiently high amount of energy (i.e., around  $n$ ) is pumped.

Kieu [7] proposed a quantum computation based on a Hamiltonian of the form of (15), which is a very special case of (11) used in Conjecture 8. The purpose and method of Kieu's are both quite different from ours. His purpose is to perform hypercomputation, i.e., to solve the membership problem for every c.e. set. On the other hand, his method is based on adiabatic quantum computation and uses the MRDP theorem in a direct manner. Kieu uses only ground states (i.e., the quantum states with the lowest energy), while the whole energy spectrum is needed in our approach. These facts suggest that his method may not be useful for proving Conjecture 8.

## 7 Quantum Proving without Giving the Proof

In Section 5 we discussed the quantum provability for a formal system whose theorems can be coded by a set  $S$  defined as in (14). When an element  $m$  is obtained as an outcome of the measurement, we can extract the proof  $n_1, \dots, n_k$  which certifies that  $m$  is a theorem of the formal system  $\mathcal{S}$ , i.e., it satisfies  $m = F(n_1, \dots, n_k)$ , by performing the second measurement over the state immediately after the first measurement.

Actually, the proof  $n_1, \dots, n_k$  may be generated slightly before the theorem  $F(n_1, \dots, n_k)$  is obtained, like in the classical scenario. As we saw in Section 4, the measurement of  $F(N_1, \dots, N_k)$  can first be performed by simultaneous measurements of the number operators  $N_1, \dots, N_k$  to produce the

---

<sup>2</sup> Decoherence, which is induced by the interaction of quantum registers with the external environment, destroys the superposition of states of the quantum registers, which plays an essential role in a conventional quantum computation.

outcomes  $n_1, \dots, n_k \in \mathbb{N}$ ; then, the theorem  $m = F(n_1, \dots, n_k)$ , classically calculated from  $n_1, \dots, n_k$ , can be regarded as the outcome of the measurement of  $F(N_1, \dots, N_k)$  itself.

In general, the set of all theorems of a (recursively axiomatisable) formal system, such as Peano Arithmetic or ZFC, forms a computably enumerable set and not a computable set of the form (14). In what follows, we argue the plausibility that, for general formal systems, the proof cannot be obtained immediately after the theorem was obtained via the quantum procedure proposed in the previous section.

Fix a formal system whose theorems form a computably enumerable set. As before we identify a formula with a natural number. Let  $M$  be a Turing machine such that, given a formula  $F$  as an input,  $M$  searches all proofs one by one and halts if  $M$  finds the proof of  $F$ . Assume that Conjecture 8 holds. Then there exists an observable  $A$  of an infinite dimensional quantum system such that  $A$  is constructive and the eigenvalue spectrum of  $A$  is exactly the set of all provable formulae. Thus, we obtain a provable formula as a measurement outcome each time we perform a measurement of  $A$ ; it is stochastically determined which provable formula is obtained. The probability of getting a specific provable formula  $F$  as a measurement outcome depends on the choice of the state  $|\Psi\rangle$  on which we perform the measurement of  $A$ . In some cases the probability can be very low, and therefore we may be able to get the provable formula  $F$  as a measurement outcome only once, even if we repeat the measurement of  $A$  on  $|\Psi\rangle$  many times.

Suppose that, in this manner, we have performed the measurement of  $A$  once and then we have obtained a specific provable formula  $F$  as a measurement outcome. Then, where is the proof of  $F$ ? In the quantum mechanical computer discussed in Section 6, the computation history of the Turing machine  $M$  is encoded in an eigenstate of the observable  $A$ , hence the proof of  $F$  is encoded in the eigenstate of  $A$ , which is the state of the underlying quantum system immediately after the measurement.

Is it possible to extract the proof of  $F$  from this eigenstate? In order to extract the proof of  $F$  from this eigenstate, it is necessary to perform an additional measurement on this eigenstate. However, it is impossible to determine the eigenstate in terms of the basis  $\{|n_1, \dots, n_k\rangle\}$  completely by a *single* measurement due the principle of quantum mechanics. In other words, there does not exist a POVM measurement which can determine all the expansion coefficients  $\{c_{n_1, \dots, n_k}\}$  of the eigenstate with respect to the basis  $\{|n_1, \dots, n_k\rangle\}$  up to a global factor with nonzero probability. This eigenstate is destroyed after the additional measurement and therefore we cannot perform any measurement on it any more. We cannot copy the eigenstate prior to the additional measurement due to the no-cloning theorem (see [3]); and even if we start again from the measurement of  $A$ , we may have little chance of getting the same provable formula  $F$  as a measurement outcome.

The above analysis suggests that even if we get a certain provable formula  $F$  as a measurement outcome through the measurement of  $A$  it is very difficult

or unlikely to simultaneously obtain the proof of  $F$ .<sup>3</sup> This argument suggests that *for a general formal system proving that a formula is a theorem is different from writing up the proof of the formula*. Of course, since  $F$  is provable, there is a proof of  $F$ , hence the Turing machine  $M$  with the input  $F$  will eventually produce that proof. However, this classical computation may take a long time in contrast with the fact—via the measurement of  $A$ —it took only a moment to know that the formula  $F$  is provable.

As mathematicians guess true facts for no apparent reason we can speculate that human intuition might work as in the above described quantum scenario. As the proposed quantum mechanical computer can operate at room temperature it may be even possible that a similar quantum mechanical process works in the human brain those offering an argument in favour of the quantum mind hypothesis [10]. The argument against this proposition according to which quantum systems in the brain decohere quickly and cannot control brain function (see [11]) could be less relevant as decoherence plays no role in the quantum computation discussed here.

**Acknowledgement.** We thank Professor K. Svozil for useful comments.

## References

1. Aigner, M., Schmidt, V.A.: Good proofs are proofs that make us wiser: interview with Yu. I. Manin. The Berlin Intelligencer, 16–19 (1998), <http://www.ega-math.narod.ru/Math/Manin.html>
2. Berndt, B.C.: Ramanujan’s Notebooks, Part V. Springer, Heidelberg (2005)
3. Buzek, V., Hillery, M.: Quantum cloning. Physics World 14(11), 25–29 (2001)
4. Calude, C.S., Calude, E., Marcus, S.: Proving and programming. In: Calude, C.S. (ed.) Randonness & Complexity, from Leibniz to Chaitin, pp. 310–321. World Scientific, Singapore (2007)
5. Dirac, P.A.M.: The Principles of Quantum Mechanics, 4th edn. Oxford University Press, London (1958)
6. Hiai, F., Yanagi, K.: Hilbert Spaces and Linear Operators. Makino-Shoten (1995) (in Japanese)
7. Kieu, T.D.: A reformulation of Hilbert’s tenth problem through quantum mechanics. Proc. R. Soc. Lond. A 460, 1535–1545 (2004)
8. Mahan, G.D.: Many-Particle Physics, 3rd edn. Kluwer Academic/Plenum Publishers, New York (2010)
9. Matijasevič, Y.V.: Hilbert’s Tenth Problem. The MIT Press, Cambridge (1993)
10. Penrose, R., Hameroff, S.: Consciousness in the universe: Neuroscience, quantum space-time geometry and Orch OR theory. Journal of Cosmology 14 (2011), <http://journalofcosmology.com/Consciousness160.html>
11. Tegmark, M.: Importance of quantum decoherence in brain processes. Physical Review E 61(4), 4194–4206 (2000)

---

<sup>3</sup> For the formal system  $\mathcal{S}$  in Section 5, we can obtain a theorem and its proof simultaneously via measurements since the observable  $F(N_1, \dots, N_k)$  whose measurements produce “theorems” is a function of the commuting observables  $N_1, \dots, N_k$  whose measurements produce “proofs”. However, this is unlikely to be true for general formal systems.

# On the Power of P Automata<sup>\*</sup>

Erzsébet Csuhaj-Varjú<sup>1</sup> and György Vaszil<sup>2</sup>

<sup>1</sup> Department of Algorithms and Their Applications, Faculty of Informatics  
Eötvös Loránd University

Pázmány Péter sétány 1/c, 1117 Budapest, Hungary  
csuhaj@inf.elte.hu

<sup>2</sup> Department of Computer Science, Faculty of Informatics  
University of Debrecen

P.O. Box 12, 4010 Debrecen, Hungary  
vaszil.gyorgy@inf.unideb.hu

**Abstract.** We study the computational power of P automata, variants of symport/antiport P systems which characterize string languages by applying a mapping to the sequence of multisets entering the system during computations. We consider the case when the input mapping is defined in such a way that it maps a multiset to the set of strings consisting of all permutations of its elements. We show that the computational power of this type of P automata is strictly less than so called restricted logarithmic space Turing machines, and we also exhibit a strict infinite hierarchy within the accepted language class based on the number of membranes present in the system.

## 1 Introduction

P systems or membrane systems are distributed parallel computing devices, inspired by the functioning and the architecture of the living cell. The basic concept was introduced by Gheorghe Păun, see [8]. A P system consists of a hierarchically embedded structure of membranes where each membrane encloses a region that contains objects and might also contain other membranes. There are rules associated to the regions describing the evolution and communication of the objects present in the membranes. A sequence of configurations following each other is a computation. P systems have intensively been studied in the last years; the interested reader might consult the handbook [10] for a detailed overview of the area.

Important variants of membrane systems are P automata, symport/antiport P systems which accept strings in an automaton-like fashion. They were introduced in [3] (for a summary on P automata, see [2]). Strings in the language of a P automaton are obtained as mappings of the multiset sequences which enter the P system through the skin membrane during an accepting computation.

---

<sup>\*</sup> Supported in part by the Hungarian Scientific Research Fund, “OTKA”, grant no. K75952, and by the European Union through the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project which is co-financed by the European Social Fund.

In [1], the case of simple, non-erasing mappings was examined. It was shown that if the rules of the P automaton are applied sequentially (one rule in each region at every computation step), then the accepted language class is strictly included in the class of languages accepted by nondeterministic one-way Turing machines with a logarithmically bounded workspace, or, if the rules are applied in the maximally parallel manner (as many rules are applied in the regions simultaneously as possible), then the class of context-sensitive languages is obtained. A closely related concept is the *analyzing P system* [4], which corresponds to a P automaton where the mapping to obtain the accepted language is defined in such a way that a multiset is mapped by a mapping  $f_{perm}$  to the set of strings which consists of all permutations of the elements of the multiset.

In this paper we deal with the power of P automata when the mapping  $f_{perm}$  is used to obtain the accepted language. Although this P automata variant has been widely studied, its precise language accepting power has not been described. Answering the open questions from [5], we show that both in the sequential and in the maximally parallel rule application mode these P automata variants describe a class of languages properly included in the class of languages accepted by nondeterministic Turing machines working with restricted logarithmically bounded workspace. This class of languages is of special interest since in this case for every accepted input of length  $n$ , there is an accepting computation where the number of nonempty cells on the work-tape(s) is bounded by  $\log d$  where  $d \leq n$  is the number of input tape cells already read, that is, the distance of the reading head from the left end of the one-way input tape. This property can be interpreted as follows: the workspace that can be used for computation is provided by the objects of the already consumed input, a property also typical for natural systems. Furthermore, we also show that the number of membranes in P automata defined over unary alphabet and using  $f_{perm}$  in arbitrary working mode induces a strict infinite hierarchy of the corresponding language classes according to inclusion.

To obtain the results we introduce two variants of counter machines, making it possible to read multisets (represented as sets of all permutations) and manipulating counters in a conventional manner.

## 2 Preliminaries and Definitions

Throughout the paper we assume that the reader is familiar with the basics of formal language theory and membrane computing; for details we refer to [12] and [10].

An alphabet is a finite non-empty set of symbols. Given an alphabet  $V$ , we denote by  $V^*$  the set of all strings over  $V$ . If the empty string,  $\lambda$ , is not included, then we use notation  $V^+$ . The length of a string  $x \in V^*$  is denoted by  $|x|$ . For any set of symbols  $A \subseteq V$  and for any symbol  $a \in V$ , the number of occurrences of symbols from  $A$  in  $x$  is denoted by  $|x|_A$ , while  $|x|_a$  denotes the number of occurrences of the symbol  $a$  in  $x$ .

A finite multiset over an alphabet  $V$  is a mapping  $M : V \rightarrow \mathbb{N}$  where  $\mathbb{N}$  is the notation for the set of non-negative integers;  $M(a)$  is said to be the multiplicity of  $a$  in  $M$ .  $M$  can also be represented by any permutation of a string  $x = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$ , where if  $M(x) \neq 0$ , then there exists  $j$ ,  $1 \leq j \leq n$ , such that  $x = a_j$ . To simplify the notations, the set of all finite multisets over an alphabet  $V$  is denoted by  $V^*$ , and we use the notation  $V^+$  for denoting the set of nonempty (finite) multisets. The empty multiset is denoted by  $\lambda$  as in the case of the empty string. If confusion may arise, we make explicit whether we speak of a string or a multiset.

A P system is a structure of hierarchically embedded membranes (a rooted tree), each having a unique label and enclosing a region containing a multiset of objects. The outermost membrane which is unique and usually labeled with 1, is called the skin membrane.

An antiport rule is of the form  $(u, in; v, out)$ , where  $u, v \in V^*$  are finite multisets over  $V$ . If such a rule is applied in a region, then the objects of  $u$  enter from the parent region and, in the same step, objects of  $v$  leave to the parent region. If only  $(u, in)$  or  $(u, out)$  is indicated, then we speak of symport rules. The rules can also be equipped with promoters or inhibitors, written as  $(u, in; v, out)|_z$ , or  $(u, in; v, out)|_{\bar{z}}$ ,  $z \in V^*$ . In the first case the rule can only be applied if the objects of the promoter multiset  $z$  are all present in the given region, in the second case, the rule can be applied if no element of  $z$  is present. Analogously, promoters or inhibitors can be added to symport rules as well. We assume that the environment contains an unlimited supply of objects, thus, if an antiport rule is applied in the skin region, then the requested multiset is always able to enter the system from the environment (which is the parent region of the skin membrane).

A *P automaton* (of degree  $k$ ) is a membrane system  $\Pi = (V, \mu, w_1, \dots, w_k, P_1, \dots, P_k)$  with object alphabet  $V$ , membrane structure  $\mu$ , initial contents (multisets) of the  $i$ th region  $w_i \in V^*$ ,  $1 \leq i \leq k$ , and sets of antiport rules with promoters or inhibitors  $P_i$ ,  $1 \leq i \leq k$ . Furthermore,  $P_1$  must not contain any rule of the form  $(a, in)$  where  $a$  is an object from  $V$ .

The configurations of the P automaton can be changed by transitions in the sequential mode (*seq*) or in the non-deterministic maximally parallel mode (*par*). In the first case one rule is applied in each region in every step, in the second case as many rules are applied simultaneously in the regions at the same step as possible. Thus, a transition in the P automaton  $\Pi$  is  $(v_1, \dots, v_m) \in \delta_{\Pi, X}(u_0, u_1, \dots, u_m)$ , where  $\delta_{\Pi, X}$  denotes the transition relation,  $X \in \{seq, par\}$ ,  $u_1, \dots, u_k$  are the contents of the  $k$  regions,  $u_0$  is the multiset entering the system from the environment, and  $v_1, \dots, v_k$ , respectively, are the contents of the  $k$  regions after performing the transition in the working mode.

In this way, there is a sequence of multisets which enter the system from the environment during the steps of its computations. If the computation is accepting, that is, if it halts, then this multiset sequence is called an accepted multiset sequence.

From any accepted multiset sequence over  $V$ , a string of the accepted language, that is, a string over some alphabet  $\Sigma$  is obtained by the application of a mapping  $f : V^* \rightarrow 2^{\Sigma^*}$ , mapping each multiset to a finite set of strings.

Let  $\Pi$  be a P automaton as above, and let  $f$  be a mapping  $f : V^* \rightarrow 2^{\Sigma^*}$  for some finite alphabet  $\Sigma$ . The *language* over  $\Sigma$  accepted by  $\Pi$  in the  $X$ -mode of rule application, where  $X \in \{seq, par\}$  with respect to  $f$  is defined as  $L_X(\Pi, f, \Sigma) = \{f(v_1) \dots f(v_s) \mid v_1, \dots, v_s \text{ is an accepted multiset sequence of } \Pi \text{ in mode } X\}$ .

In [5] the authors consider P automata with  $f$  defined in such a way that a multiset over  $V$  is mapped by  $f$  to the set of strings which consists of all permutations of the elements of the multiset. In the following, this mapping will be denoted by  $f_{perm}$ . Since in this case  $\Sigma$  does not differ from  $V$ , we denote the accepted language by  $L_X(\Pi, f_{perm})$ . The class of languages accepted by  $\Pi$  automata working in the  $X$ -mode,  $X \in \{seq, par\}$  defined by mapping  $f_{perm}$  is denoted by  $\mathcal{L}_X(PA, f_{perm})$ .

Finally, we recall a notion from [1]. A nondeterministic Turing machine with a one-way input tape is *restricted logarithmic space bounded* if for every accepted input of length  $n$ , there is an accepting computation where the number of nonempty cells on the work-tape(s) is bounded by  $O(\log d)$  where  $d \leq n$  is the number of input tape cells already read, that is, the *distance* of the reading head from the left end of the one-way input tape. The class of languages accepted by such machines is denoted by r1LOGSPACE. In [1] it is shown that this class is strictly included in 1LOGSPACE, the class of languages accepted by Turing machines with a one-way input tape in logarithmic space, and that sequential P automata with certain types of simple input mappings exactly characterize this language class.

### 3 Results

We first define *restricted counter machine acceptors* corresponding to the language class r1LOGSPACE. For the sake of readability, we present only the necessary formal details.

A *restricted  $k$ -counter machine acceptor*  $M$ , an RCMA in short, is a (non-deterministic) counter machine with  $k$  counters (holding non-negative integers) and a one-way read only input tape. Thus,  $M = (Q, \Sigma, k, \delta, q_0, F)$  for some  $k \geq 1$ , where  $Q$  is the set of internal states,  $\Sigma$  is the input alphabet,  $q_0 \in Q$  is the initial state,  $F \subseteq Q$  is the set of final states, and  $\delta : Q \times \Sigma^* \times C^k \rightarrow 2^{Q \times D^k}$ , where  $C = \{zero, nonzero\}$ , denoting the two types of observations the machine can make on its counters,  $D = \{increment, decrement, none\}$  denoting the operations the machine can execute on its counters. (Note that  $\delta$  is finitely defined, that is, defined for a finite subset of  $\Sigma^*$ , and a counter can be incremented/decremented by one at any computational step.) Moreover,

- the transition relation is defined in such a way that the reading head can read more than one input symbol in one computational step in the “multiset sense”, that is,  $\delta(q, x, \alpha) = \delta(q, y, \alpha)$  for each  $x, y \in \Sigma^*$  which represent the same multiset. Moreover,

- the sum of the values stored in the counters can only increase as much in one computational step as the number of symbols read in that same step, that is, for all  $(q', \beta) \in \delta(q, x, \alpha)$  we have  $|\beta|_{\text{increment}} - |\beta|_{\text{decrement}} \leq |x|$ .

Let the class of languages accepted by restricted counter machine acceptors be denoted by  $\mathcal{L}(\text{RCMA})$ .

We also introduce the notion of the restricted version of a special multihead finite automata defined in [6]. A *special multihead finite automaton* with  $k$  heads is a multihead finite automaton having a one-way reading head, and  $k - 1$  two-way “counter” heads which are only able to distinguish if they are positioned on the left end marker, the right end marker, or somewhere in between.

Let  $\mathcal{L}(\text{RSMFA})$  denote the class of languages which can be accepted by special multihead finite automata in such a way that none of the counter heads ever moves beyond (to the right of) the one-way reading head.

As it is also mentioned in [6], special multihead finite automata are equivalent to logarithmic space Turing machines with a one-way read only input tape. It is not difficult to see that the restricted variant defined above is equivalent to restricted one-way logarithmic space Turing machines, that is, it also characterizes  $\text{rLOGSPACE}$ .

Now we show that restricted  $k$ -counter machine acceptors also characterize  $\text{rLOGSPACE}$ .

**Lemma 1.**  $\mathcal{L}(\text{RCMA}) = \text{rLOGSPACE}$ .

*Proof.* We show that restricted counter machine acceptors (RCMA) characterize  $\mathcal{L}(\text{RSMFA})$ , which in turn, is equal to  $\text{rLOGSPACE}$ . It is clear that  $\mathcal{L}(\text{RCMA}) \subseteq \mathcal{L}(\text{RSMFA})$  since RCMA with  $k$  counters can be simulated by RSMFA with  $k + 1$  heads. As the sum of the counter contents of the RCMA is not larger than the number of symbols read from the input, none of the “counter” heads of the RSMFA needs to move beyond the one-way input reading head. The only issue we need to solve concerns the ability of RCMA to read strings representing multisets with possibly more than one elements in a single computational step. This can be simulated with RSMFA by modifying the finite control in such a way that the strings read by these transitions of the RCMA are simulated in several steps of the RSMFA.

To see that  $\mathcal{L}(\text{RSMFA}) \subseteq \mathcal{L}(\text{RCMA})$ , we show how to simulate an RSMFA  $M$  having  $k$  heads with an RCMA  $M'$  having  $k$  counters. The one-way reading head of  $M$  is simulated by the one-way reading head of  $M'$ , while the  $k - 1$  counter heads of  $M$  are simulated by the first  $k - 1$  counters of  $M'$ , the increment and decrement of the counters simulating the left and right movements of the tape heads. Suppose now that the one-way reading head of  $M$  has advanced  $m$  cells to the right from the left end of the input tape. This means that to record the positions of each of the  $k$  counter heads of  $M$ , the simulating RCMA  $M'$  should record  $k$  non-negative integers, each having the value of at most  $m$ . Since the sum of the values of the counters of  $M'$  cannot be more than  $m$ , it cannot store these  $k$  integers in its counters directly. But, instead of storing a value  $0 \leq j_i \leq m$ , it can store a value  $0 \leq l_i \leq \lfloor k/m \rfloor$  such that  $j_i = l_i \cdot k + s_i$  where



$s_i \in \{0, 1, \dots, k-1\}$ . Thus  $s_i$  can be recorded in the state of the finite control for each counter  $1 \leq i \leq k$ .

Furthermore, when a counter of  $M'$  is decremented, then the  $k$ th counter (which is not necessary for the simulation of the  $k-1$  heads of  $M$ ) is incremented in order not to “lose” the possibility of incrementing another counter without reading more input symbols (this might be necessary if the right move of a counter head can be executed by  $M$  without further advancing the one-way reading head).  $\square$

Now we use RCMA to show that the class of languages characterized by P automata with the mapping  $f_{perm}$  are strictly included in rLOGSPACE.

**Theorem 2.**  $\mathcal{L}_X(\text{PA}, f_{perm}) \subset \text{rLOGSPACE}$  where  $X \in \{\text{seq}, \text{par}\}$ .

*Proof.* First we show that  $\mathcal{L}_X(\text{PA}, f_{perm}) \subseteq \text{rLOGSPACE}$  for  $X \in \{\text{seq}, \text{par}\}$ . Let  $L = L_X(\Pi, f_{perm})$ , for one of the modes  $X \in \{\text{seq}, \text{par}\}$ , and let  $\Pi = (V, \mu, w_1, \dots, w_m, P_1, \dots, P_m)$ .

We show how to construct an RCMA,  $M = (Q, \Sigma, k, \delta, q_{ini}, F)$ , such that  $L = L(M)$ . The machine  $M$  is able to simulate the computations of  $\Pi$  by keeping track of the number of different objects in the different regions of  $\Pi$ .  $M$  has three counters for each symbol-region pair, these are called storage counters, temporary counters, and assistant counters, one additional counter for each symbol which are called input counters, and  $d$  additional counters called input assistant counters where  $d$  is the maximal number of objects which can enter the skin membrane from the environment by the application of one antiport rule ( $d = \max(\{|v| \mid (u, \text{out}; v, \text{in})|_z \in P_1\})$ ). Thus,  $k = 3 \cdot |V| \cdot m + |V| + d$ . These counters are initially empty, so the number of objects in the initial configuration of  $M$  is recorded in the components  $(c_1, \dots, c_{k'})$  of the internal state  $q_{ini} = (q_0, c_1, \dots, c_{k'}) \in Q' \times \mathbb{N}^{k'}$  with  $k' = k - |V|$ .

The simulation of a computational step  $(v_1, \dots, v_m) \in \delta_\Pi(u_0, u_1, \dots, u_m)$  of  $\Pi$  by  $M$  can be described as follows. (We omitted the indication of the working mode in  $\delta$  since it is irrelevant at the moment.) In the first phase,  $M$  reads  $w_{u_0}$  from the input tape, where  $w_{u_0}$  is a string corresponding to the multiset  $u_0$ , and for each symbol  $a_i \in V$  with  $|w_{u_0}|_{a_i} = j_i$ , sets the corresponding input counter to  $j_i$ ,  $1 \leq i \leq |V|$ . This is done in several input reading steps: If the skin region of  $\Pi$  contains a rule  $(x, \text{out}; y, \text{in})|_z$ ,  $x, y, z$  being multisets over  $\Sigma$ , then  $M$  is able to read in one computational step all strings  $w_y \in \Sigma^*$  representing  $y$ . Thus, by the (repeated) use of the input reading transitions  $M$  is able to read the strings corresponding to the multisets which can be imported into  $\Pi$  by the (parallel) use of its antiport rules. During such an input reading transition corresponding to a rule  $(x, \text{out}; y, \text{in})|_z$ ,  $M$  first increments the first  $|y|$  of the input assistant counters, then increments the input counters corresponding to the objects of  $y$  by decrementing one of the input assistant counters for each increment.

Now  $M$  nondeterministically chooses symport/antiport rules (possibly with promoters or inhibitors) from the sets  $P_i$ ,  $1 \leq i \leq m$ , of  $\Pi$ , and updates the counters which keep track of the configuration of  $\Pi$  according to the chosen rule. The storage counters corresponding to the region and the objects which leave

the region are decremented by the necessary amount, and the number of objects entering the region are added to the corresponding temporary counters. If an object enters from the environment in the skin region of  $\Pi$ , the corresponding input counter of  $M$  is decremented. (Note that the “counter components” of the internal states are also taken into account: their value and the value of the corresponding “real” counter together represent the number of objects in  $\Pi$ . When such an “internal counter” is decremented, then the increment of the necessary temporary counter also happens in the corresponding “internal” version of that counter. This way this nondeterministic rule choosing and configuration modifying phase of the computation of  $M$  does not increase the overall sum of the values stored in the different counters.) If  $\Pi$  works in the sequential mode, at most one rule is chosen from each set, otherwise, if  $\Pi$  works in the maximally parallel mode, then several rules can be chosen from the same set.

When this phase is finished,  $M$  checks whether the configuration change implied by the above chosen rules corresponds to the required mode of operation (sequential or maximally parallel). In both cases, it makes sure first that the input counters are empty.

In the sequential case,  $M$  needs to check that in those regions where no rules have been chosen, none of the available rules are applicable. To this end, it can keep a record of the regions where rules were used in its finite control, and check the applicability of the rules of that region one by one, using the corresponding assistant counters to store the numbers which are subtracted from various counters during the process in order to be able to easily restore the original configuration when the checking of the applicability of a rule fails. In the maximally parallel case, the checking phase must check the applicability of rules in each region, which can be done in the same way as described above (including the skin region, to check whether the multiset entering from the environment was also maximal).

After the checking of the chosen rule set,  $M$  realizes the configuration change by updating the storage counters using the values from the temporary counters, and the simulation of the next computational step can start by reading a new string from the input tape. Before continuing with the simulation,  $M$  can check whether the current configuration is final or not, and decide to proceed or to stop accordingly. (A configuration is final if it is halting, thus, if no rule can be applied in any of the regions.)

Now we show that the inclusion of  $\mathcal{L}_X(\text{PA}, f_{perm})$  in  $\text{r1LOGSPACE}$  is proper,  $X \in \{seq, par\}$ . Consider the language  $L_1 = \{(ab)^n \# w \mid w \in \{1\}\{0,1\}^* \text{ such that } val(w) = n > 1\}$  where  $val(w)$  denotes the value of  $w$  as a binary integer. It is not difficult to see that  $L_1 \in \text{r1LOGSPACE}$ . On the other hand,  $L_1 \notin \mathcal{L}_X(\text{PA}, f_{perm})$ , as follows from Lemma 4.1 of [5]. This lemma states that any non-regular language over an alphabet  $V$  in  $\mathcal{L}_{par}(\text{PA}, f_{perm})$  contains a word  $w$  which can be written as  $w = v_1abv_2$ ,  $a, b \in V$ , such that  $w' = v_1bav_2$  is also a word of the language. The argument proving the lemma also holds for systems working in the sequential mode, thus, as no word of  $L_1$  contains such a subword, we can conclude that  $L_1 \notin \mathcal{L}_X(\text{PA}, f_{perm})$  for  $X \in \{seq, par\}$ .  $\square$

Next we are going to show that there exists an infinite hierarchy of language classes in connection to P automata with input mapping  $f_{perm}$ . To this aim, we introduce the notion of a special restricted counter machine acceptor.

A *special restricted  $k$ -counter machine acceptor*, an SRCMA in short, is a restricted  $k$ -counter machine acceptor  $M = (Q, \Sigma, k, \delta, q_0, F)$ , but in addition, the transition relation  $\delta$  is defined in such a way, that if the length of the string  $x$  read in one computational step is  $l$ , then the sum of the values stored in the counters can only increase at most as much as  $l - 1$  in the same computational step. Thus,

$$- \text{ for all } (q', \beta) \in \delta(q, x, \alpha), \text{ we have } |\beta|_{increment} - |\beta|_{decrement} \leq |x| - 1.$$

Note that an SRCMA which reads at most one symbol in one computational step cannot accept any non-regular language. (The counter values cannot increase, thus, the machine is equivalent to a finite automaton.)

Let  $\mathcal{L}(\text{SRCMA})$  denote the class of languages accepted by SRCMA.

We will make use of the following two lemmas about special restricted counter machine acceptors (SRCMA).

**Lemma 3.** *A language  $L$  is accepted by a P automaton with input mapping  $f_{perm}$ , working in any of the sequential or maximally parallel modes, if and only if  $L$  can be accepted by an SRCMA.*

*Proof.* First we show that any language  $L \subseteq \Sigma^*$  accepted by a P automaton with input mapping  $f_{perm}$  is also in  $\mathcal{L}(\text{SRCMA})$ . For any such  $L = L_X(\Pi, f_{perm})$ , for one of the modes  $X \in \{seq, par\}$  accepted by a P automaton  $\Pi$ , we can construct an SRCMA  $M$  in a similar way as in the first part of the proof of Theorem 2. Since  $M$  is a special RCMA (SRCMA), it can increase the values of its counters in an even more restricted manner as an RCMA can. But still, as we assume that rules of the form  $(a, in)$  cannot be used in the skin membrane of the P automaton  $\Pi$ , the allowed sum of the counter contents of  $M$  corresponds to the number of symbols entering the membrane system during the computation. This means that using a similar technique as in the proof of Lemma 1,  $M$  can still record the configurations of  $\Pi$  in its counters.

Now we show how an SRCMA  $M = (Q, \Sigma, k, \delta, q_0, F)$  can be simulated by P automata in the maximally parallel mode. Let the transitions defined by  $\delta$  be labeled in a one-to-one manner by the set  $lab(\delta)$ , and let the simulating P automaton be defined as  $\Pi = (V, \mu, w_1, \dots, w_{k+2}, P_1, \dots, P_{k+2})$  with  $V = \Sigma \cup \{q_0, C, D, E, F \mid 1 \leq i \leq 5\} \cup \{B_{i,t}, t_1, t_2, t_3 \mid 1 \leq i \leq 5, t \in lab(\delta)\} \cup \{A_i A'_i \mid 3 \leq i \leq k+2\}$ , membrane structure  $\mu = [ [ ]_2 [ ]_3 \dots [ ]_{k+2} ]_1$ , where the rule sets with the initial membrane contents are as follows. (For easier readability, instead of the string notation, we denote the initial multisets by enumerating their elements between parentheses.)

$$w_1 = \{q_0, C, D\},$$

$$P_1 = \{(a, out; u, in)|_{t_1} \mid a \in \Sigma, t \in lab(\delta) \text{ is a transition of } M \\ \text{which reads a string representing } u \text{ from the input tape}\}$$

$w_2 = \{a, B_{i,t}, t_1, (t_2)^k, (t_3)^k \mid 1 \leq i \leq 5, t \in \text{lab}(\delta)\}$  where  $a$  is some element of  $\Sigma$  and  $(t_i)^k$  denotes  $k$  copies of the object  $t_i$ ,

$$P_2 = \{(t_1 a, \text{out}; q_0 D, \text{in}) \mid a \in \Sigma, t \in \text{lab}(\delta) \text{ labels a transition from } q_0 \in Q\} \cup \\ \{(B_{1,t} D (t_2)^k, \text{out}; t_1, \text{in}) \mid t \in \text{lab}(\delta)\} \cup \{(a, \text{out})|_D \mid a \in \Sigma\} \cup \\ \{(B_{2,t} (t_3)^k, \text{out}; B_{1,t}, \text{in}), (B_{3,t}, \text{out}; B_{2,t}, \text{in}), (B_{4,t}, \text{out}; B_{3,t} (t_2)^k, \text{in}), \\ (B_{5,t}, \text{out}; (t_3)^k B_{4,t} C a, \text{in}), (s_1 a, \text{out}; B_{5,t} D, \text{in}) \mid t, s \in \text{lab}(\delta) \text{ where } s \\ \text{is a transition which can follow } t, a \in \Sigma\} \cup \\ \{(E, \text{out}; B_{5,t}, \text{in}) \mid t \in \text{lab}(\delta) \text{ is a transition leading to a final state of } M\} \cup \\ \{(a, \text{in})|_C, (C, \text{out}) \mid a \in \Sigma\},$$

and for  $3 \leq i \leq k+2$ , let

$$w_i = \{A_i, A'_i, F, F\}, \\ P_i = \{(A_i, \text{out}; t_2, \text{in}), (A'_i, \text{out})|_{t_2}, (A_i A'_i, \text{in}), (F, \text{in}; F, \text{out})\} \cup \\ \{(t_2 a, \text{out}, t_3, \text{in}), (t_2 F, \text{out}), (t_3, \text{out}) \mid t \in \text{lab}(\delta) \text{ is a transition which decrements the value of counter } i - 2\} \cup \\ \{(t_2, \text{out}; t_3, \text{in}), (t_3, \text{out}; a, \text{in}) \mid t \in \text{lab}(\delta) \text{ is a transition which increments the value of counter } i - 2\} \cup \\ \{(F a, \text{out})|_{t_2}, (t_2, \text{out}; t_3, \text{in}), (t_3, \text{out}) \mid t \in \text{lab}(\delta) \text{ is a transition which requires that the value of counter } i - 2 \text{ is zero}\}$$

The above defined system has a skin region, a region representing the finite control (region 2), and  $k$  regions corresponding to the  $k$  counters of  $M$  (regions  $3 \leq i \leq k+2$ , referred to as the counter regions). The counter regions represent the values stored in the counters of  $M$  with objects from  $\Sigma$ , region  $i$  contains as many such objects as the values stored in counter  $i - 2$ . The object  $q_0$  present in the skin region in the initial configuration is exchanged for a symbol  $t_1$  for a transition symbol  $t \in \text{lab}(\delta)$  denoting a transition from the initial state.

The simulation of a computational step of  $M$  starts by having one terminal object  $a \in \Sigma$ , and a transition symbol  $t_1$  for some transition  $t \in \text{lab}(\delta)$  of  $M$  in the skin membrane. The terminal object  $a$  is used by a rule  $(a, \text{out}; u, \text{in})|_{t_1}$  to import a multiset  $u \in \Sigma^*$  which is read by  $M$  during the transition  $t$ . Now the transition symbol is imported into region 2, and  $k$  copies of  $t_2$  (corresponding to the same transition, but indexed with 2) are exported to the skin region together with all the copies of objects from  $\Sigma$  which are not used inside the counter regions (these are stored in region 2 until they are needed). In the next five steps, the values stored in the  $k$  counter regions are modified as necessary while the symbol  $B_{1,t}$  is changed to  $B_{5,t}$ , increasing its index by one in every step. If a counter needs to be decremented or checked for being zero, then the objects  $t_2$  enter and take with them a terminal object to the skin region or perform the zero

check as necessary. Meanwhile  $k$  copies of  $t_3$  are released from region 2 which continue the process by bringing in terminal objects to the counter regions when the counter in question needs to be incremented during transition  $t \in \text{lab}(\delta)$ . After the modification of the counter values, the remaining terminal objects are transported back to region 2, and the symbol  $s_1$  for the next transition appears, together with exactly one terminal object  $a \in \Sigma$ , so the simulation of the next computational step of  $M$  can start in the same manner.

The simulation finishes when, after executing a transition leading to a final state of  $M$ , the symbol  $E$  is exported from region 2 to the skin region and the system halts.

Due to space restrictions, we do not describe here how SRCMA can be simulated by P automata with sequential rule application. We could use a similar construction to the one presented in the proof of Theorem 6.2 in [2].  $\square$

In the following lemma, we denote the class of languages accepted by SRCMA with  $k$  heads by  $\mathcal{L}(\text{SRCMA}, k)$ , and the class of languages accepted by (two-way)  $k$ -head finite automata (MFA) by  $\mathcal{L}(\text{MFA}, k)$ .

**Lemma 4.**  $\mathcal{L}(\text{SRCMA}, k) \subset \mathcal{L}(\text{SRCMA}, 2k + 4)$ ,  $k \geq 1$ .

*Proof.* It is clear that SRCMA can be simulated by two-way multihead finite automata, since a two-way head can also be used as a counter if the numbers which need to be stored do not exceed the length of the input, which is the case in SRCMA. As SRCMA may read strings of more than one symbol in a single computational step, we need to modify the finite control of the simulating multihead automaton (as in the proof of Lemma 1) in such a way that the strings read by these transitions are also read, although in several steps. This means that  $\mathcal{L}(\text{SRCMA}, k) \subseteq \mathcal{L}(\text{MFA}, k + 1)$  holds for  $k \geq 1$ . Further, it is shown in [7] that there are unary languages which cannot be accepted by two-way multihead finite automata with  $k$  heads, but can be accepted by such automata with  $k + 1$  heads, so we have  $\mathcal{L}(\text{MFA}, k) \subset \mathcal{L}(\text{MFA}, k + 1)$ . Considering multihead automata over unary languages, there is no difference between a “real” reading head and a counter head, thus, over unary inputs, a two way multihead automaton  $M$  with  $k$  heads can be simulated by an SRCMA  $M'$  with  $2k$  counters. To this aim,  $M'$  uses a pair of counters,  $c_{i,1}$  and  $c_{i,2}$ , to simulate the  $i$ th reading head of  $M$ . The value of  $c_{i,1}$  corresponds to the number of symbols left of the  $i$ th head of  $M$ , the value of  $c_{i,2}$  corresponds to the number of the remaining symbols to right (together with the one that is being read). The simulation begins with an initialization phase, when  $M'$  records the number of symbols found on its input tape in the second “component” of its counters,  $c_{i,2}$ ,  $1 \leq i \leq k$ , achieving this way a configuration corresponding to the initial configuration of  $M$ , when all heads are positioned at the beginning of the tape. The sum of the numbers which need to be recorded is  $k \cdot n$  (where  $n$  is the length of the input). If during this first phase,  $M'$  reads its input by two symbols in every step, then it has  $\lfloor n/2 \rfloor$  as an upper bound for the sum of its counter contents, so  $M'$  uses the same technique as in the proof of Lemma 1 to record the configurations of  $\Pi$  with actually using only the capacity of its counters. Thus, if we consider unary

languages, we have  $\mathcal{L}(\text{MFA}, k) \subseteq \mathcal{L}(\text{SRCMA}, 2k)$ . Combining the three relations above, we obtain the statement of the lemma.  $\square$

Using the lemma above, we can separate the language classes accepted by P automata with input mapping  $f_{perm}$  having different number of membranes.

**Theorem 5.** *For every  $r$ , there is an  $s > r$  and a unary language  $L$  which can be accepted by a P automaton with input mapping  $f_{perm}$  and  $s$  membranes, but not by any such P automaton with  $r$  membranes.*

*The statement holds for P automata working in both the sequential and the maximally parallel modes.*

*Proof.* We use a similar reasoning as in [6]. As we have shown in the proof of Theorem 2, P automata with input mapping  $f_{perm}$  can be simulated by an SRCMA with  $(3m+1) \cdot |V| + d$  counters, where  $|V|$  is the cardinality of the object alphabet, and  $d$  is a constant based on the antiport rules of the skin region. Now, if a P automaton accepts a language over a unary alphabet  $\Sigma = \{a\}$ , then the number of objects different from  $a$  inside the system cannot increase during any computation. Thus, the information about their positions in the system can be recorded in the finite control of the simulating SRCMA. This means that the number of necessary counters to simulate a P automaton accepting a unary language is  $3m + 1 + d$  where  $m$  is the number of membranes.

Assume now, that there is an  $r$ , such that all unary languages which can be accepted by any P automaton can also be accepted by a P automaton with  $r$  membranes. This would mean, that all unary P automata languages could also be accepted by SRCMA with  $3r + 1 + d$  counters. But according to Lemma 3 above, all SRCMA languages can also be accepted by P automata, thus, all unary languages accepted by SRCMA can be also be accepted by  $3r + 1 + d$  counters. This contradicts Lemma 4, so our statement is proved.  $\square$

## 4 Conclusion

We have shown that P automata working any of the sequential or nondeterministic maximally parallel mode and using mapping  $f_{perm}$  to obtain the words of its language from its accepted multiset sequences are less powerful than nondeterministic Turing machines working with restricted logarithmically bounded workspace, where only the already consumed input can be used as workspace for computation. We also show that if these P automata variants operate over unary object alphabets, then the number of membranes in the P automata induces an infinite strict hierarchy of language classes with respect to inclusion. The mapping  $f_{perm}$  is a natural concept since it reflects to the fact that the objects which appear simultaneously can be observed in any order. Thus our results suggest that certain variants of natural systems that use only existing resources for computation and observation may not exhibit much computational power.

## References

1. Csuhaj-Varjú, E., Ibarra, O.H., Vaszil, G.: On the computational complexity of  $P$  automata. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 76–89. Springer, Heidelberg (2005)
2. Csuhaj-Varjú, E., Oswald, M., Vaszil, G.:  $P$  automata. In: [10], ch. 6, pp. 144–167
3. Csuhaj-Varjú, E., Vaszil, G.:  $P$  automata or purely communicating accepting  $P$  systems. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC 2002. LNCS, vol. 2597, pp. 219–233. Springer, Heidelberg (2003)
4. Freund, R., Oswald, M.: A short note on analysing  $P$  systems. Bulletin of the EATCS 78, 231–236 (2002)
5. Freund, R., Kogler, M., Păun, G., Pérez-Jiménez, M.J.: On the power of  $P$  and  $dP$  automata. Annals of Bucharest University Mathematics-Informatics Series LVIII, 5–22 (2009)
6. Ibarra, O.H.: Membrane hierarchy in  $P$  systems. Theoretical Computer Science 334(1-3), 115–129 (2005)
7. Monien, B.: Two-way multihead automata over a one-letter alphabet. RAIRO - Informatique théorique/Theoretical Informatics 14(1), 67–82 (1980)
8. Păun, G.: Membrane Computing: An Introduction. Natural Computing Series. Springer, Berlin (2002)
9. Păun, G., Pérez-Jiménez, M.J.: Solving problems in a distributed way in membrane computing:  $dP$  systems. International Journal of Computers, Communication and Control V(2), 238–250 (2010)
10. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
11. Pérez-Jiménez, M.J.: A computational complexity theory in membrane computing. In: Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., Rozenberg, G., Salomaa, A. (eds.) WMC 2009. LNCS, vol. 5957, pp. 125–148. Springer, Heidelberg (2010)
12. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Berlin (1997)
13. Vaszil, G.: On the parallelizability of languages accepted by  $P$  automata. In: Kelemen, J., Kelemenová, A. (eds.) Computation, Cooperation, and Life. LNCS, vol. 6610, pp. 170–178. Springer, Heidelberg (2011)

# Array Insertion and Deletion P Systems

Henning Fernau<sup>1</sup>, Rudolf Freund<sup>2</sup>, Sergiu Ivanov<sup>3</sup>,  
Markus L. Schmid<sup>1</sup>, and K.G. Subramanian<sup>4</sup>

<sup>1</sup> Fachbereich 4 – Abteilung Informatikwissenschaften, Universität Trier  
D-54296 Trier, Germany  
{fernau,MSchmid}@uni-trier.de

<sup>2</sup> Technische Universität Wien, Institut für Computersprachen  
Favoritenstr. 9, A-1040 Wien, Austria  
rudi@emcc.at

<sup>3</sup> Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est  
Créteil Val de Marne, 61, Av. Gén. de Gaulle, 94010 Créteil, France  
sergiu.ivanov@u-pec.fr

<sup>4</sup> School of Computer Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia  
kgsmani1948@yahoo.com

**Abstract.** We consider the ( $d$ -dimensional) array counterpart of string insertion and deletion grammars and use the operations of array insertion and deletion in the framework of P systems where the applicability of the rules depends on the membrane region. In this paper, we especially focus on examples of two-dimensional array insertion and deletion P systems and show that we can already obtain computational completeness using such P systems with a membrane structure of tree height of at most two and only the targets *here*, *in*, and *out*.

## 1 Introduction

In the string case, the insertion operation was first considered in [13–15] and after that related insertion and deletion operations were investigated, e.g., in [16, 17]. Backed by linguistic motivation, checking of insertion contexts was considered in [18]. These *contextual* grammars start from a set of strings (*axioms*), and new strings are obtained by using rules of the form  $(s, c)$ , where  $s$  and  $c$  are strings to be interpreted as inserting  $c$  in the context of  $s$ , either only at the ends of strings (*external* case, [18]) or in the *interior* of strings ([21]). The fundamental difference between contextual grammars and Chomsky grammars is that in contextual grammars we do not *rewrite* symbols, but we only *adjoin* symbols to the current string, i.e., contextual grammars are pure grammars. Hence, among the variants of these grammars as, for example, considered in [5–7, 22, 23, 19], the variant where we can retain only the set of strings produced by blocked derivations, i.e., derivations which cannot be continued, is of special importance. This corresponds to the maximal mode of derivation (called t-mode) in cooperating grammar systems (see [2]) as well as to the way results in P systems are obtained by halting computations; we refer the reader to [20, 24] and to the web page [25] for more details on P systems.



With the length of the contexts and/or of the inserted and deleted strings being big enough, the insertion-deletion closure of a finite language leads to computational completeness. There are numerous results establishing the descriptive complexity parameters sufficient to achieve this goal; for an overview of this area we refer to [29, 28]. In [12] it was shown that computational completeness can also be obtained with using only insertions and deletions of just one symbol at the ends of a string using the regulating framework of P systems, where the application of rules depends on the membrane region.

The contextual style of generating strings was extended to  $d$ -dimensional arrays in a natural way (see [11]): a contextual array rule is a pair  $(s, c)$  of two arrays to be interpreted as inserting the new subarray  $c$  in the context of the array  $s$  provided that the positions where to put  $c$  are not yet occupied by a non-blank symbol. With retaining only the arrays produced in maximal derivations, interesting languages of two-dimensional arrays can be generated. In [8], contextual array rules in P systems are considered. A contextual array rule  $(s, c)$  can be interpreted as array insertion rule; by inverting the meaning of this operation, we get an array deletion rule  $(s, c)$  deleting the subarray  $c$  in the relative context of the subarray  $s$ .

In this paper, we exhibit some illustrative examples of P systems with (two-dimensional) array insertion rules (corresponding to contextual array rules). The main result of the paper exhibits computational completeness of (two-dimensional) array insertion and deletion P systems.

## 2 Definitions and Examples

The set of integers is denoted by  $\mathbb{Z}$ , the set of non-negative integers by  $\mathbb{N}$ . An *alphabet*  $V$  is a finite non-empty set of abstract *symbols*. Given  $V$ , the free monoid generated by  $V$  under the operation of concatenation is denoted by  $V^*$ ; the elements of  $V^*$  are called strings, and the *empty string* is denoted by  $\lambda$ ;  $V^* \setminus \{\lambda\}$  is denoted by  $V^+$ . The family of recursively enumerable string languages is denoted by  $RE$ . For more details of formal language theory the reader is referred to the monographs and handbooks in this area such as [4] and [27].

### 2.1 A General Model for Sequential Grammars

In order to be able to introduce the concept of membrane systems (P systems) for various types of objects, we first define a general model ([10]) of a grammar generating a set of terminal objects by derivations where in each derivation step exactly one rule is applied (sequential derivation mode) to exactly one object.

A (*sequential*) *grammar*  $G$  is a construct  $(O, O_T, w, P, \Longrightarrow_G)$  where  $O$  is a set of *objects*,  $O_T \subseteq O$  is a set of *terminal objects*,  $w \in O$  is the *axiom (start object)*,  $P$  is a finite set of *rules*, and  $\Longrightarrow_G \subseteq O \times O$  is the *derivation relation* of  $G$ . We assume that each of the rules  $p \in P$  induces a relation  $\Longrightarrow_p \subseteq O \times O$  with respect to  $\Longrightarrow_G$  fulfilling at least the following conditions: (i) for each object  $x \in O$ ,  $(x, y) \in \Longrightarrow_p$  for only finitely many objects  $y \in O$ ; (ii) there exists a

finitely described mechanism (as, for example, a Turing machine) which, given an object  $x \in O$ , computes all objects  $y \in O$  such that  $(x, y) \in \Longrightarrow_p$ . A rule  $p \in P$  is called *applicable* to an object  $x \in O$  if and only if there exists at least one object  $y \in O$  such that  $(x, y) \in \Longrightarrow_p$ ; we also write  $x \Longrightarrow_p y$ . The derivation relation  $\Longrightarrow_G$  is the union of all  $\Longrightarrow_p$ , i.e.,  $\Longrightarrow_G = \cup_{p \in P} \Longrightarrow_p$ . The reflexive and transitive closure of  $\Longrightarrow_G$  is denoted by  $\Longrightarrow_G^*$ .

In the following we shall consider different types of grammars depending on the components of  $G$ , especially on the rules in  $P$ ; these may define a special type  $X$  of grammars which then will be called *grammars of type  $X$* .

Usually, the *language generated by  $G$*  (in the  $*$ -mode) is the set of all terminal objects (we also assume  $v \in O_T$  to be decidable for every  $v \in O$ ) derivable from the axiom, i.e.,  $L_*(G) = \{v \in O_T \mid w \Longrightarrow_G^* v\}$ . The *language generated by  $G$  in the  $t$ -mode* is the set of all terminal objects derivable from the axiom in a halting computation, i.e.,  $L_t(G) = \{v \in O_T \mid (w \Longrightarrow_G^* v) \wedge \nexists z (v \Longrightarrow_G z)\}$ . The family of languages generated by grammars of type  $X$  in the derivation mode  $\delta$ ,  $\delta \in \{*, t\}$ , is denoted by  $\mathcal{L}_\delta(X)$ . If for every  $G$  of type  $X$ ,  $G = (O, O_T, w, P, \Longrightarrow_G)$ , we have  $O_T = O$ , then  $X$  is called a *pure type*, otherwise it is called *extended*.

## 2.2 String Grammars

In the general notion as defined above, a *string grammar*  $G_S$  is represented as  $((N \cup T)^*, T^*, w, P, \Longrightarrow_P)$  where  $N$  is the alphabet of *non-terminal symbols*,  $T$  is the alphabet of *terminal symbols*,  $N \cap T = \emptyset$ ,  $w \in (N \cup T)^+$  is the *axiom*,  $P$  is a finite set of *string rewriting rules*, and the derivation relation  $\Longrightarrow_{G_S}$  is the classic one for string grammars defined over  $V^* \times V^*$ , with  $V := N \cup T$ . As classic types of string grammars we consider string grammars with arbitrary rules of the form  $u \rightarrow v$  with  $u \in V^+$  and  $v \in V^*$  as well as context-free rules of the form  $A \rightarrow v$  with  $A \in N$  and  $v \in V^*$ . The corresponding types of grammars are denoted by *ARB* and *CF*, thus yielding the families of languages  $\mathcal{L}(ARB)$  and  $\mathcal{L}(CF)$ , i.e., the family of recursively enumerable languages *RE* and the family of context-free languages, respectively.

In [12], left and right insertions and deletions of strings were considered; the corresponding types of grammars using rules inserting strings of length at most  $k$  and deleting strings of length at most  $m$  are denoted by  $D^m I^k$ .

## 2.3 Array Grammars

We now introduce the basic notions for  $d$ -dimensional arrays and array grammars in a similar way as in [9, 11]. Let  $d \in \mathbb{N}$ ; then a  *$d$ -dimensional array*  $\mathcal{A}$  over an alphabet  $V$  is a function  $\mathcal{A} : \mathbb{Z}^d \rightarrow V \cup \{\#\}$ , where  $shape(\mathcal{A}) = \{v \in \mathbb{Z}^d \mid \mathcal{A}(v) \neq \#\}$  is finite and  $\# \notin V$  is called the *background* or *blank symbol*. We usually write  $\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in shape(\mathcal{A})\}$ . The set of all  $d$ -dimensional arrays over  $V$  is denoted by  $V^{*d}$ . The *empty array* in  $V^{*d}$  with empty shape is denoted by  $A_d$ . Moreover, we define  $V^{+d} = V^{*d} \setminus \{A_d\}$ . Let  $v \in \mathbb{Z}^d$ ,

$v = (v_1, \dots, v_d)$ ; the norm of  $v$  is defined as  $\|v\| = \max\{|v_i| : 1 \leq i \leq d\}$ ; the translation  $\tau_v : \mathbb{Z}^d \rightarrow \mathbb{Z}^d$  is defined by  $\tau_v(w) = w + v$  for all  $w \in \mathbb{Z}^d$ . For any array  $\mathcal{A} \in V^{*d}$  we define  $\tau_v(\mathcal{A})$ , the corresponding  $d$ -dimensional array translated by  $v$ , by  $(\tau_v(\mathcal{A}))(w) = \mathcal{A}(w - v)$  for all  $w \in \mathbb{Z}^d$ . The vector  $(0, \dots, 0) \in \mathbb{Z}^d$  is denoted by  $\Omega_d$ .

Usually (e.g., see [1, 26, 30]) arrays are regarded as equivalence classes of arrays with respect to linear translations, i. e., only the relative positions of the symbols different from  $\#$  in the plane are taken into account: the equivalence class  $[\mathcal{A}]$  of an array  $\mathcal{A} \in V^{*d}$  is defined by

$$[\mathcal{A}] = \{\mathcal{B} \in V^{*d} \mid \mathcal{B} = \tau_v(\mathcal{A}) \text{ for some } v \in \mathbb{Z}^d\}.$$

The set of all equivalence classes of  $d$ -dimensional arrays over  $V$  with respect to linear translations is denoted by  $[V^{*d}]$  etc.

A  $d$ -dimensional array grammar  $G_A$  is represented as

$$\left( \left[ (N \cup T)^{*d} \right], [T^{*d}], [\mathcal{A}_0], P, \Longrightarrow_{G_A} \right)$$

where  $N$  is the alphabet of *non-terminal symbols*,  $T$  is the alphabet of *terminal symbols*,  $N \cap T = \emptyset$ ,  $\mathcal{A}_0 \in (N \cup T)^{*d}$  is the *start array*,  $P$  is a finite set of  *$d$ -dimensional array rules* over  $V$ ,  $V := N \cup T$ , and  $\Longrightarrow_{G_A} \subseteq \left[ (N \cup T)^{*d} \right] \times \left[ (N \cup T)^{*d} \right]$  is the derivation relation induced by the array rules in  $P$ .

A “classical”  $d$ -dimensional array rule  $p$  over  $V$  is a triple  $(W, \mathcal{A}_1, \mathcal{A}_2)$  where  $W \subseteq \mathbb{Z}^d$  is a finite set and  $\mathcal{A}_1$  and  $\mathcal{A}_2$  are mappings from  $W$  to  $V \cup \{\#\}$ . In the following, we shall also write  $\mathcal{A}_1 \rightarrow \mathcal{A}_2$ , because  $W$  is implicitly given by the finite arrays  $\mathcal{A}_1, \mathcal{A}_2$ . We say that the array  $\mathcal{C}_2 \in V^{*d}$  is *directly derivable* from the array  $\mathcal{C}_1 \in V^{*d}$  by  $(W, \mathcal{A}_1, \mathcal{A}_2)$  if and only if there exists a vector  $v \in \mathbb{Z}^d$  such that  $\mathcal{C}_1(w) = \mathcal{C}_2(w)$  for all  $w \in \mathbb{Z}^d - \tau_v(W)$  as well as  $\mathcal{C}_1(w) = \mathcal{A}_1(\tau_{-v}(w))$  and  $\mathcal{C}_2(w) = \mathcal{A}_2(\tau_{-v}(w))$  for all  $w \in \tau_v(W)$ , i. e., the subarray of  $\mathcal{C}_1$  corresponding to  $\mathcal{A}_1$  is replaced by  $\mathcal{A}_2$ , thus yielding  $\mathcal{C}_2$ ; we also write  $\mathcal{C}_1 \Longrightarrow_p \mathcal{C}_2$ . Moreover we say that the array  $\mathcal{B}_2 \in [V^{*d}]$  is *directly derivable* from the array  $\mathcal{B}_1 \in [V^{*d}]$  by the  $d$ -dimensional array production  $(W, \mathcal{A}_1, \mathcal{A}_2)$  if and only if there exist  $\mathcal{C}_1 \in \mathcal{B}_1$  and  $\mathcal{C}_2 \in \mathcal{B}_2$  such that  $\mathcal{C}_1 \Longrightarrow_p \mathcal{C}_2$ ; we also write  $\mathcal{B}_1 \Longrightarrow_p \mathcal{B}_2$ .

A  $d$ -dimensional array rule  $p = (W, \mathcal{A}_1, \mathcal{A}_2)$  in  $P$  is called *monotonic* if  $\text{shape}(\mathcal{A}_1) \subseteq \text{shape}(\mathcal{A}_2)$  and  *$\#$ -context-free* if  $\text{shape}(\mathcal{A}_1) = \{\Omega_d\}$ ; if it is  *$\#$ -context-free* and, moreover,  $\text{shape}(\mathcal{A}_2) = W$ , then  $p$  is called *context-free*. A  $d$ -dimensional array grammar is said to be of type  $X$ ,  $X \in \{d\text{-ARBA}, d\text{-MONA}, d\text{-}\#\text{-CFA}, d\text{-CFA}\}$  if every array rule in  $P$  is of the corresponding type, the corresponding families of array languages of equivalence classes of  $d$ -dimensional arrays by  $d$ -dimensional array grammars are denoted by  $\mathcal{L}_*(X)$ . These families form a Chomsky-like hierarchy, i. e.,  $\mathcal{L}_*(d\text{-CFA}) \subsetneq \mathcal{L}_*(d\text{-MONA}) \subsetneq \mathcal{L}_*(d\text{-ARBA})$  and  $\mathcal{L}_*(d\text{-CFA}) \subsetneq \mathcal{L}_*(d\text{-}\#\text{-CFA}) \subsetneq \mathcal{L}_*(d\text{-ARBA})$ . Two  $d$ -dimensional arrays  $\mathcal{A}$  and  $\mathcal{B}$  in  $[V^{*d}]$  are called *shape-equivalent* if and only if  $\text{shape}(\mathcal{A}) = \text{shape}(\mathcal{B})$ . Two  $d$ -dimensional array languages  $L_1$  and  $L_2$  from  $[V^{*d}]$  are called *shape-equivalent* if and only if  $\{\text{shape}(\mathcal{A}) \mid \mathcal{A} \in L_1\} = \{\text{shape}(\mathcal{B}) \mid \mathcal{B} \in L_2\}$ .

### 2.4 Contextual, Insertion and Deletion Array Rules

A *d-dimensional contextual array rule* (see [11]) over the alphabet  $V$  is a pair of finite  $d$ -dimensional arrays  $((W_1, \mathcal{A}_1), (W_2, \mathcal{A}_2))$  where  $W_1 \cap W_2 = \emptyset$  and  $shape(\mathcal{A}_1) \cup shape(\mathcal{A}_2) \neq \emptyset$ . The effect of this contextual rule is the same as of the array rewriting rule  $(W_1 \cup W_2, \mathcal{A}_1, \mathcal{A}_1 \cup \mathcal{A}_2)$ , i.e., in the context of  $\mathcal{A}_1$  we insert  $\mathcal{A}_2$ . Hence, such an array rule  $((W_1, \mathcal{A}_1), (W_2, \mathcal{A}_2))$  can also be called an *array insertion rule*, and then we write  $I((W_1, \mathcal{A}_1), (W_2, \mathcal{A}_2))$ ; if  $shape(\mathcal{A}_i) = W_i, i \in \{1, 2\}$ , we simply write  $I(\mathcal{A}_1, \mathcal{A}_2)$ . Yet we may also interpret the pair  $((W_1, \mathcal{A}_1), (W_2, \mathcal{A}_2))$  as having the effect of the array rewriting rule  $\mathcal{A}_1 \cup \mathcal{A}_2 \rightarrow \mathcal{A}_1$ , i.e., in the context of  $\mathcal{A}_1$  we delete  $\mathcal{A}_2$ ; in this case, we speak of an *array deletion rule* and write  $D((W_1, \mathcal{A}_1), (W_2, \mathcal{A}_2))$  or  $D(\mathcal{A}_1, \mathcal{A}_2)$ .

Let  $G_A$  be a  $d$ -dimensional array grammar  $([V^{*d}], [T^{*d}], [\mathcal{A}_0], P, \Longrightarrow_{G_A})$  with  $P$  containing array insertion and deletion rules. Then we can consider the array languages  $L_*(G_A)$  and  $L_t(G_A)$  generated by  $G_A$  in the modes  $*$  and  $t$ , respectively; the corresponding families of array languages are denoted by  $\mathcal{L}_\delta(d-DIA), \delta \in \{*, t\}$ ; if only array insertion (i.e., contextual) rules are used, we have the case of pure grammars, and we also write  $\mathcal{L}_\delta(d-CA)$ . For interesting relations between the families of array languages  $\mathcal{L}_*(d-CA)$  and  $\mathcal{L}_t(d-CA)$  as well as  $\mathcal{L}_*(d-\#-CFA)$  and  $\mathcal{L}_*(d-CFA)$  we refer the reader to [11].

As a first example we illustrate that the generative power of contextual array grammars can exceed that of context-free array grammars (also see [8], [11]):

*Example 1.* Consider the set  $R_H$  of hollow rectangles with arbitrary side lengths  $p, q \geq 3$  (over the singleton alphabet  $\{a\}$ ). By extending arguments used for similar problems in [3], it is easy to see that there cannot exist a grammar of type  $d-CFA$  generating an array language which is shape-equivalent to  $R_H$ ; on the other hand, the following grammar of type  $2-CA$  yields such an array language in the  $t$ -mode, i.e.,  $shape(L_t(G_1)) = shape(R_H)$ :  $G_1 = (\{a, b\}^{*2}, \{a, b\}^{*2}, P, \mathcal{A}_0, \Longrightarrow_{G_1})$  with  $\mathcal{A}_0 = \begin{smallmatrix} a \\ a \end{smallmatrix}$ ; for a graphical representation of the rules in  $P$ , we use the convention that the symbols from the two arrays  $\mathcal{A}_1, \mathcal{A}_2$  in the contextual array production  $I(\mathcal{A}_1, \mathcal{A}_2)$  are shown in one array and those from  $\mathcal{A}_1$  are marked by a box frame:

$$\begin{aligned}
 p_1 &= \begin{smallmatrix} a \\ a \end{smallmatrix}, \quad p_2 = \boxed{a} \boxed{a} a, \quad p_3 = \begin{smallmatrix} b & b \\ \boxed{a} & \end{smallmatrix}, \quad p_4 = \boxed{a} \boxed{a} \begin{smallmatrix} b \\ \end{smallmatrix} \\
 p_5 &= \boxed{b} \boxed{b} b, \quad p_6 = \begin{smallmatrix} b \\ \boxed{b} \end{smallmatrix}, \quad p_7 = \begin{smallmatrix} \boxed{b} & \boxed{b} & b \\ & & \boxed{b} \end{smallmatrix}.
 \end{aligned}$$

Starting from the axiom  $\mathcal{A}_0$ , we can go up using the rule  $p_1$   $p - 3$  times and go to the right using the rule  $p_2$   $q - 3$  times, where  $p, q \geq 3$  can be chosen arbitrarily. Then we turn to the right from the vertical line by once using the rule  $p_3$  and turn up from the horizontal line by once using the rule  $p_4$ , respectively; in both

cases symbols  $b$  are appended to the growing lines of symbols  $a$ , whereafter the rules  $p_1, p_2, p_3$ , and  $p_4$  cannot be applied anymore. The rules  $p_5$  and  $p_6$  then complete the upper and the right edge of the rectangle. The derivation only halts if the rule  $p_7$  is applied at the end.  $\square$

The following example shows how we can generate an array language of coated filled squares with a specified middle point, which will be an essential part in the proof of our main theorem in Section 4:

*Example 2.* The contextual array grammar

$$G_2 = \left( \{\bar{S}, E, Q\}^{*2}, \{\bar{S}, E, Q\}^{*2}, P, \mathcal{A}_0, \implies_{G_2} \right)$$

generates an array language of squares filled by the symbol  $E$ , coated by a layer of symbols  $Q$ , with the central position being marked by the symbol  $\bar{S}$  :

$$\begin{array}{rcc}
 & & Q\ Q \dots Q \dots Q\ Q \\
 & & Q\ E \dots E \dots E\ Q \\
 E\ E\ E\ E & & \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \\
 E\ E\ E\ E\ E & & \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \\
 \mathcal{A}_0 = E\ E\ \bar{S}\ E\ E & \xrightarrow{G_2} & Q\ E \dots \bar{S} \dots E\ Q \\
 E\ E\ E\ E\ E & & \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \\
 E\ E\ E\ E\ E & & Q\ E \dots E \dots E\ Q \\
 & & Q\ Q \dots Q \dots Q\ Q
 \end{array}$$

The final arrays are constructed in such a way that, starting from the axiom, layer by layer, another layer of symbols  $E$  is added, by applying the rules  $p_0$  to  $p_7$ ; the final layer of symbols  $Q$  is added by using the rules  $q_0$  to  $q_8$ . In sum,  $P$  contains the following contextual rules:

$$\begin{aligned}
 p_0 &= \begin{array}{c} E \\ \boxed{E} \end{array} \begin{array}{c} E \\ \boxed{E} \end{array}, \quad q_0 = \begin{array}{c} Q \\ \boxed{E} \end{array} \begin{array}{c} Q \\ \boxed{E} \end{array}, \quad p_1 = \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} E \\ \boxed{E} \end{array} \begin{array}{c} E \\ \boxed{E} \end{array}, \quad q_1 = \begin{array}{c} \boxed{Q} \\ \boxed{E} \end{array} \begin{array}{c} Q \\ \boxed{E} \end{array} \begin{array}{c} E \\ \boxed{E} \end{array}, \\
 p_2 &= \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} E \\ \boxed{E} \end{array} \begin{array}{c} E \\ \boxed{E} \end{array}, \quad q_2 = \begin{array}{c} \boxed{Q} \\ \boxed{E} \end{array} \begin{array}{c} Q \\ \boxed{E} \end{array} \begin{array}{c} Q \\ \boxed{E} \end{array}, \quad p_3 = \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} \boxed{E} \\ E \end{array}, \quad q_3 = \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} \boxed{Q} \\ Q \end{array}, \\
 p_4 &= \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} \boxed{E} \\ E \end{array}, \quad q_4 = \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} \boxed{Q} \\ Q \end{array}, \quad p_5 = \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} \boxed{E} \\ E \end{array} \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array}, \quad q_5 = \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} \boxed{E} \\ Q \end{array} \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array}, \\
 p_6 &= \begin{array}{c} E \\ E \end{array} \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array} \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array}, \quad q_6 = \begin{array}{c} Q \\ Q \end{array} \begin{array}{c} \boxed{E} \\ Q \end{array} \begin{array}{c} \boxed{E} \\ Q \end{array}, \quad p_7 = \begin{array}{c} \boxed{E} \\ \boxed{E} \end{array}, \quad q_7 = \begin{array}{c} \boxed{Q} \\ \boxed{E} \end{array}, \quad \text{and} \\
 q_8 &= \begin{array}{c} Q \\ Q \end{array} \begin{array}{c} \boxed{Q} \\ \boxed{E} \end{array}.
 \end{aligned}$$

A derivation in  $G_2$  halts if and only if we end up with applying the rule  $q_8$ , thus finishing the coating layer of symbols  $Q$ .  $\square$

### 3 (Sequential) P Systems

A (sequential) P system of type  $X$  with tree height  $n$  is a construct

$$\Pi = (G, \mu, R, i_0) \text{ where}$$

- $G = (O, O_T, A, P, \Longrightarrow_G)$  is a sequential grammar of type  $X$ ;
- $\mu$  is the membrane (tree) structure of the system with the height of the tree being  $n$  ( $\mu$  usually is represented by a string containing correctly nested marked parentheses); we assume the membranes to be the nodes of the tree representing  $\mu$  and to be uniquely labelled by labels from a set  $Lab$ ;
- $R$  is a set of rules of the form  $(h, r, tar)$  where  $h \in Lab$ ,  $r \in P$ , and  $tar$ , called the *target indicator*, is taken from the set  $\{here, in, out\} \cup \{in_h \mid h \in Lab\}$ ;  $R$  can also be represented by the vector  $(R_h)_{h \in Lab}$ , where  $R_h = \{(r, tar) \mid (h, r, tar) \in R\}$  is the set of rules assigned to membrane  $h$ ;
- $i_0$  is the initial membrane containing the axiom  $A$ .

As we only have to follow the trace of a single object during a computation of the P system, a configuration of  $\Pi$  can be described by a pair  $(w, h)$  where  $w$  is the current object (e.g., string or array) and  $h$  is the label of the membrane currently containing the object  $w$ . For two configurations  $(w_1, h_1)$  and  $(w_2, h_2)$  of  $\Pi$  we write  $(w_1, h_1) \Longrightarrow_{\Pi} (w_2, h_2)$  if we can pass from  $(w_1, h_1)$  to  $(w_2, h_2)$  by applying a rule  $(h_1, r, tar) \in R$ , i.e.,  $w_1 \Longrightarrow_r w_2$  and  $w_2$  is sent from membrane  $h_1$  to membrane  $h_2$  according to the target indicator  $tar$ . More specifically, if  $tar = here$ , then  $h_2 = h_1$ ; if  $tar = out$ , then the object  $w_2$  is sent to the region  $h_2$  immediately outside membrane  $h_1$ ; if  $tar = in_{h_2}$ , then the object is moved from region  $h_1$  to the region  $h_2$  immediately inside region  $h_1$ ; if  $tar = in$ , then the object  $w_2$  is sent to one of the regions immediately inside region  $h_1$ .

A sequence of transitions between configurations of  $\Pi$ , starting from the initial configuration  $(A, i_0)$ , is called a *computation* of  $\Pi$ . A *halting computation* is a computation ending with a configuration  $(w, h)$  such that no rule from  $R_h$  can be applied to  $w$  anymore;  $w$  is called the *result* of this halting computation if  $w \in O_T$ . As the language generated by  $\Pi$  we consider  $L_t(\Pi)$  which consists of all terminal objects from  $O_T$  being results of a halting computation in  $\Pi$ .

By  $\mathcal{L}_t(X-LP)$  ( $\mathcal{L}_t(X-LP^{(n)})$ ) we denote the family of languages generated by P systems (of tree height at most  $n$ ) using grammars of type  $X$ . If only the targets *here*, *in*, and *out* are used, then the P system is called *simple*, and the corresponding families of languages are denoted by  $\mathcal{L}_t(X-LsP)$  ( $\mathcal{L}_t(X-LsP^{(n)})$ ).

In the string case (see [12]), every language  $L \subseteq T^*$  in  $\mathcal{L}_*(D^1I^1)$  can be written in the form  $T_l^*ST_r^*$  where  $T_l, T_r \subseteq T$  and  $S$  is a finite subset of  $T^*$ . Using the regulating mechanism of P systems, we get  $\{a^{2^n} \mid n \geq 0\} \in \mathcal{L}_t(D^1I^2-LP^{(1)})$  and even obtain computational completeness:

**Theorem 1.** (see [12])  $\mathcal{L}_t(D^1I^1-LsP^{(8)}) = RE$ .

One-dimensional arrays can also be interpreted as strings; left/right insertion (deletion) of a symbol  $a$  corresponds to taking the set containing all rules  $I(a \boxed{b})/I(\boxed{b} a)$  ( $D(a \boxed{b})/D(\boxed{b} a)$ ) for all  $b$ ; hence, from Theorem 1,

we immediately infer the following result, which with respect to the tree height of the simple P systems will be improved considerably in Section 4:

**Corollary 1.**  $\mathcal{L}_t(1\text{-DIA-}LsP^{(8)}) = \mathcal{L}_*(1\text{-ARBA})$ .

For the array case we now restrict ourselves to the 2-dimensional case. First we go back to Example 1 and show how we can take advantage of having different contextual array rules to be applied in different membranes:

*Example 3.* Consider the contextual array grammar  $G_1$  from Example 1 and the P system  $\Pi_1 = (G_1, [{}_0 [{}_1 ]_1 [{}_2 ]_2 ]_0, R, 0)$  with  $R$  containing the rules  $(0, p_1, in_1)$ ,  $(1, p_2, out)$ ,  $(0, p_3, in_2)$ ,  $(2, p_4, out)$ ,  $(0, p_5, in_1)$ ,  $(1, p_6, out)$  and  $(0, p_7, in_2)$ . By synchronizing the growth of the left and the lower edge of the rectangle, we guarantee that the resulting rectangle finally appearing in membrane 2 is a square. Hence, we have shown that with  $L_t(\Pi_1)$  in  $\mathcal{L}_t(2\text{-CA-LP}^{(1)})$  we can find an array language which is shape-equivalent to the set of hollow squares.  $\square$

## 4 Computational Completeness of Array Insertion and Deletion P Systems

We now show our main result that any recursively enumerable 2-dimensional array language can be generated by an array insertion and deletion P system which only uses the targets *here*, *in*, and *out* and whose membrane structure has only tree height 2.

**Theorem 2.**  $\mathcal{L}_t(2\text{-DIA-}LsP^{(2)}) = \mathcal{L}_*(2\text{-ARBA})$ .

*Proof.* The main idea of the proof is to construct the simple P system  $\Pi$  of type 2-DIA with a membrane structure of height two generating a recursively enumerable 2-dimensional array language  $L_A$  given by a grammar  $G_A$  of type 2-ARBA in such a way that we first generate the coated squares described in Example 2 and then simulate the rules of the 2-dimensional array grammar  $G_A$  inside this square; finally, the superfluous symbols  $E$  and  $Q$  have to be erased to obtain the terminal array.

Now let  $G_A = \left( \left[ (N \cup T)^{*d} \right], [T^{*d}], [\mathcal{A}_0], P, \Longrightarrow_{G_A} \right)$  be an array grammar of type 2-ARBA generating  $L_A$ . In order to make the simulation in  $\Pi$  easier, without loss of generality, we may make some assumptions on the forms of the array rules in  $P$ : First of all, we may assume that the array rules are in a kind of Chomsky normal form (e.g., compare [9]), i.e., only of the following forms:  $A \rightarrow B$  for  $A \in N$  and  $B \in N \cup T \cup \{\#\}$  as well as  $AvD \rightarrow BvC$  with  $\|v\| = 1$ ,  $A, B, C \in N \cup T$ , and  $D \in N \cup T \cup \{\#\}$  (we would like to emphasize that usually  $A, B, C, D$  in the array rule  $AvD \rightarrow BvC$  would not be allowed to be terminal symbols, too); in a more formal way, the rule  $AvD \rightarrow BvC$  represents the rule  $(W, \mathcal{A}_1, \mathcal{A}_2)$  with  $W = \{(\Omega_d, v)\}$ ,  $\mathcal{A}_1 = \{(\Omega_d, A), (v, D)\}$ , and  $\mathcal{A}_2 = \{(\Omega_d, B), (v, C)\}$ . As these rules in fact are simulated in  $\Pi$  with the symbol  $E$  representing the blank symbol  $\#$ , a rule  $Av\# \rightarrow BvC$  now corresponds to a

rule  $AvE \rightarrow BvC$ . Moreover, a rule  $A \rightarrow B$  for  $A \in N$  and  $B \in N \cup T$  can be replaced by the set of all rules  $AvD \rightarrow BvD$  for all  $D \in N \cup T \cup \{E\}$  and  $v \in \mathbb{Z}^d$  with  $\|v\| = 1$ , and  $A \rightarrow \#$  can be replaced by the set of all rules  $AvD \rightarrow EvD$  for all  $D \in N \cup T \cup \{E\}$  and  $v \in \mathbb{Z}^d$  with  $\|v\| = 1$ .

After these replacements described above, in the P system  $\Pi$  we now only have to simulate rules of the form  $AvD \rightarrow BvC$  with  $\|v\| = 1$  as well as  $A, B, C, D \in N \cup T \cup \{E\}$ . Yet in order to obtain a P system  $\Pi$  with the required features, we make another assumption for the rules to be simulated: any intermediate array obtained during a derivation contains exactly one symbol marked with a bar; as we only have to deal with sequential systems where at each moment exactly one rule is going to be applied, this does not restrict the generative power of the system as long as we can guarantee that the marking can be moved to any place within the current array. Instead of a rule  $AvD \rightarrow BvC$  we therefore take the corresponding rule  $\bar{A}vD \rightarrow Bv\bar{C}$ ; moreover, to move the bar from one position in the current array to another position, we add all rules  $\bar{A}vC \rightarrow Av\bar{C}$  for all  $A, C \in N \cup T \cup \{E\}$  and  $v \in \mathbb{Z}^d$  with  $\|v\| = 1$ . We collect all these rules obtained in the way described so far in a set of array rules  $P'$  and assume them to be uniquely labelled by labels from a set of labels  $Lab'$ , i.e.,  $P' = \{l : \bar{A}_l v D_l \rightarrow B_l v \bar{C}_l \mid l \in Lab'\}$ .

After all these preparatory steps we now are able to construct the simple P system  $\Pi$  with array insertion and deletion rules:

$$\Pi = (G, [{}_0 [{}_{I_1} [{}_{I_2} ]_{I_2} ]_{I_1} \cdots [{}_{l_1} [{}_{l_2} ]_{l_2} ]_{l_1} \cdots [{}_{F_1} [{}_{F_2} ]_{F_2} ]_{F_1} ]_0, R, I_2)$$

with  $I_1$  and  $I_2$  being the membranes for generating the initial squares,  $F_1$  and  $F_2$  are the membranes to extract the final terminal arrays in halting computations, and  $l_1$  and  $l_2$  for all  $l \in Lab'$  are the membranes to simulate the corresponding array rule from  $P'$  labelled by  $l$ . The components of the underlying array grammar  $G$  can easily be collected from the description of the rules in  $R$  as described below.

We start with the initial array  $\mathcal{A}_0$  from Example 2 and take all rules  $(I_2, I(r), here)$  with all rules  $r \in \{p_i, q_i \mid 0 \leq i \leq 7\}$  taken as array insertion rules; instead of the array insertion rule  $q_8$  we now take the rule  $q'_8$  instead and  $(I_2, I(q'_8), out)$ , where  $q'_8$  introduces the new control symbols  $K$  and  $K_I$ . Using  $(I_1, D(q_9), out)$  we move the initial square out into the skin membrane.

$$q'_8 = \begin{array}{|c|c|} \hline K & K_I \\ \hline Q & \boxed{Q} \\ \hline \boxed{Q} & E \\ \hline \end{array}, q_9 = \begin{array}{|c|c|} \hline \boxed{K} & K_I \\ \hline Q & \boxed{Q} \\ \hline \end{array}$$

To be able to simulate a derivation from  $G_A$  for a specific terminal array, the workspace in this initial square has to be large enough, but as we can generate such squares with arbitrary size, such an initial array can be generated for any terminal array in  $L_*(G_A)$ .

An array rule from  $P' = \{l : \bar{A}_l v D_l \rightarrow B_l v \bar{C}_l \mid l \in Lab'\}$  is simulated by applying the following sequence of array insertion and deletion rules in the



membranes  $l_1$  and  $l_2$ , which send the array twice the path from the skin membrane to membrane  $l_2$  via membrane  $l_1$  and back to the skin membrane:

$$\begin{aligned} & \left(0, I \left( \boxed{K} K_l \right), in \right), \left( l_1, D \left( \boxed{\bar{A}_l} v D_l \right), in \right), \\ & \left( l_2, I \left( \boxed{\bar{A}_l} v \bar{D}_l^{(l)} \right), out \right), \left( l_1, D \left( \boxed{\bar{D}_l^{(l)}} (-v) \bar{A}_l \right), out \right), \\ & \left( 0, I \left( \boxed{\bar{D}_l^{(l)}} (-v) B_l \right), in \right), \left( l_1, D \left( \boxed{B_l} v \bar{D}_l^{(l)} \right), in \right), \\ & \left( l_2, D \left( \boxed{K} K_l \right), out \right), \left( l_1, I \left( \boxed{B_l} v \bar{C}_l \right), out \right). \end{aligned}$$

Whenever reaching the skin membrane, the current array contains exactly one barred symbol. If we reach any of the membranes  $l_1$  and/or  $l_2$  with the wrong symbols (which implies that none of the rules listed above is applicable), we introduce the trap symbol  $F$  by the rules  $\left( m, I \left( F \boxed{K} \right), out \right)$  and  $\left( m, I \left( F \boxed{F} \right), out \right)$  for  $m \in \{l_1, l_2 \mid l \in Lab' \cup \{I\}\}$ ; as soon as  $F$  has been introduced once, with  $\left( 0, I \left( F \boxed{F} \right), in \right)$  we can guarantee that the computation in  $\Pi$  will never stop.

As soon as we have obtained an array representing a terminal array, the corresponding array computed in  $\Pi$  is moved into membrane  $F_1$  by the rule  $(0, D(K), in)$  (for any  $X$ ,  $D(X) / I(K)$  just means deleting/inserting  $X$  without taking care of the context). In membrane  $F_1$ , all superfluous symbols  $E$  and  $Q$  as well as the marked blank symbol  $\bar{E}$  (without loss of generality we may assume that at the end of the simulation of a derivation from  $G_A$  in  $\Pi$  the marked symbol is  $\bar{E}$ ) are erased by using the rules  $(F_1, D(X), here)$  with  $X \in \{E, \bar{E}, Q\}$ . The computation in  $\Pi$  halts with yielding a terminal array in membrane  $F_1$  if and only if no other non-terminal symbols have occurred in the array we have moved into  $F_1$ ; in the case that non-terminal symbols occur, we start an infinite loop between membrane  $F_1$  and membrane  $F_2$  by introducing the trap symbol  $F$ :  $(F_1, D(X), in)$  for  $X \notin T \cup \{E, \bar{E}, Q\}$  and  $(F_2, I(F), out)$ .

As can be seen from the description of the rules in  $\Pi$ , we can simulate all terminal derivations in  $G_A$  by suitable computations in  $\Pi$ , and a terminal array  $\mathcal{A}$  is obtained as the result of a halting computation (always in membrane  $F_1$ ) if and only if  $\mathcal{A} \in L_*(G_A)$ ; hence, we conclude  $L_t(\Pi) = L_*(G_A)$ .  $\square$

## 5 Conclusion

In this paper, we have extended the notions of insertion and deletion from the string case to the case of  $d$ -dimensional arrays. Array insertion grammars have already been considered as contextual array grammars in [11], whereas the inverse interpretation of a contextual array rule as a deletion rule has newly been introduced here. Moreover, we have also introduced P systems using these array insertion and deletion rules, thus continuing the research on P systems with left and right insertion and deletion of strings, see [12].

In the main part of our paper, we have restricted ourselves to exhibit examples of 2-dimensional array languages that can be generated by array insertion (contextual array) grammars and P systems using array insertion rules as well as to show that array insertion and deletion P systems are computationally complete.

As can be seen from the proof of our main result, Theorem 2, the second part of the proof showing how to simulate array rules of the form  $\bar{A}vD \rightarrow Bv\bar{C}$  is not restricted to the 2-dimensional case and can directly be taken over to the  $d$ -dimensional case for arbitrary  $d$ ; hence, for  $d > 2$ , the main challenge is to generate  $d$ -dimensional cuboids of symbols  $E$  coated by a layer of symbols  $Q$ , with the central position marked by the start symbol  $\bar{S}$ . With regulating mechanisms such as matrix or programmed grammars without the feature of appearance checking, this challenge so far has turned out to be intractable for  $d > 2$  when using #-context-free array rules, e.g., see [9]; when using array insertion rules, this problem seems to become solvable. Moreover, we would also like to avoid the target *here* in the simple P systems using array insertion and deletion rules constructed in Theorem 2, as with avoiding the target *here*, the applications of the rules could be interpreted as being carried out when passing a membrane, in the sense of molecules passing a specific membrane channel from one region to another one. We shall return to these questions and related ones in an extended version of this paper.

**Acknowledgements.** We gratefully acknowledge the support by the project Universität der Großregion — UniGR enabling the visit of K.G. Subramanian at the University of Trier; Markus L. Schmid was supported by a scholarship of the German Academic Exchange Service (DAAD) for returning scientists.

## References

1. Cook, C.R., Wang, P.S.-P.: A Chomsky hierarchy of isotonic array grammars and languages. *Computer Graphics and Image Processing* 8, 144–152 (1978)
2. Csuhaĵ-Varjű, E., Dassow, J., Kelemen, J., Păun, G.: *Grammar Systems. A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach, London (1994)
3. Dassow, J., Freund, R., Păun, G.: Cooperating array grammar systems. *International Journal of Pattern Recognition and Artificial Intelligence* 9(6), 1029–1053 (1995)
4. Dassow, J., Păun, G.: *Regulated Rewriting in Formal Language Theory*. Springer (1989)
5. Ehrenfeucht, A., Mateescu, A., Păun, G., Rozenberg, G., Salomaa, A.: On representing RE languages by one-sided internal contextual languages. *Acta Cybernetica* 12(3), 217–233 (1996)
6. Ehrenfeucht, A., Păun, G., Rozenberg, G.: The linear landscape of external contextual languages. *Acta Informatica* 35(6), 571–593 (1996)
7. Ehrenfeucht, A., Păun, G., Rozenberg, G.: On representing recursively enumerable languages by internal contextual languages. *Theoretical Computer Science* 205(1-2), 61–83 (1998)

8. Fernau, H., Freund, R., Schmid, M.L., Subramanian, K.G., Wiederhold, P.: Contextual array grammars and array P systems (submitted)
9. Freund, R.: Control mechanisms on #-context-free array grammars. In: Păun, G. (ed.) *Mathematical Aspects of Natural and Formal Languages*, pp. 97–137. World Scientific, Singapore (1994)
10. Freund, R., Kogler, M., Oswald, M.: A general framework for regulated rewriting based on the applicability of rules. In: Kelemen, J., Kelemenová, A. (eds.) *Computation, Cooperation, and Life. LNCS*, vol. 6610, pp. 35–53. Springer, Heidelberg (2011)
11. Freund, R., Păun, G., Rozenberg, G.: Contextual array grammars. In: Subramanian, K.G., Rangarajan, K., Mukund, M. (eds.) *Formal Models, Languages and Applications. Series in Machine Perception and Artificial Intelligence*, vol. 66, pp. 112–136. World Scientific (2007)
12. Freund, R., Rogozhin, Y., Verlan, S.: P systems with minimal left and right insertion and deletion. In: Durand-Lose, J., Jonoska, N. (eds.) *UCNC 2012. LNCS*, vol. 7445, pp. 82–93. Springer, Heidelberg (2012)
13. Galiukschov, B.: *Semicontextual grammars. Logica i Matem. Lingvistika*, pp. 38–50. Tallin University (1981) (in Russian)
14. Haussler, D.: *Insertion and Iterated Insertion as Operations on Formal Languages. PhD thesis, Univ. of Colorado at Boulder* (1982)
15. Haussler, D.: Insertion languages. *Information Sciences* 31(1), 77–89 (1983)
16. Kari, L.: *On Insertion and Deletion in Formal Languages. PhD thesis, University of Turku* (1991)
17. Kari, L., Păun, G., Thierrin, G., Yu, S.: At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. In: *Proc. of 3rd DIMACS Workshop on DNA Based Computing, Philadelphia*, pp. 318–333 (1997)
18. Marcus, S.: Contextual grammars. *Rev. Roum. Math. Pures Appl.* 14, 1525–1534 (1969)
19. Păun, G.: *Marcus Contextual Grammars. Kluwer, Dordrecht* (1997)
20. Păun, G.: *Membrane Computing. An Introduction. Springer* (2002)
21. Păun, G., Nguyen, X.M.: On the inner contextual grammars. *Rev. Roum. Math. Pures Appl.* 25, 641–651 (1980)
22. Păun, G., Rozenberg, G., Salomaa, A.: Contextual grammars: erasing, determinism, one-sided contexts. In: Rozenberg, G., Salomaa, A. (eds.) *Developments in Language Theory*, pp. 370–388. World Scientific Publ., Singapore (1994)
23. Păun, G., Rozenberg, G., Salomaa, A.: Contextual grammars: parallelism and blocking of derivation. *Fundamenta Informaticae* 25, 381–397 (1996)
24. Păun, G., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing. Oxford University Press* (2010)
25. The P systems Web page, <http://ppage.psyste.ms.eu/>
26. Rosenfeld, A.: *Picture Languages. Academic Press, Reading* (1979)
27. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages*, vol. 3. Springer, Berlin (1997)
28. Verlan, S.: Recent developments on insertion-deletion systems. *Comp. Sci. J. of Moldova* 18(2), 210–245 (2010)
29. Verlan, S.: *Study of Language-theoretic Computational Paradigms Inspired by Biology. Habilitation thesis, University of Paris Est* (2010)
30. Wang, P.S.-P.: Some new results on isotonic array grammars. *Information Processing Letters* 10, 129–131 (1980)

# Boolean Logic Gates from a Single Memristor via Low-Level Sequential Logic

Ella Gale, Ben de Lacy Costello, and Andrew Adamatzky

Unconventional Computing Group, University of the West of England,  
Dept. of Applied Sciences & Dept. of Computer Science and Creative Technology,  
Frenchay Campus, Coldhambour Lane, Bristol, BS16 5SR, UK  
{ella.gale,ben.delacycostello,andrew.adamatzky}@uwe.ac.uk  
<http://uncomp.uwe.ac.uk>

**Abstract.** By using the memristor's memory to both store a bit and perform an operation with a second input bit, simple Boolean logic gates have been built with a single memristor. The operation makes use of the interaction of current spikes (occasionally called current transients) found in both memristors and other devices. The sequential time-based logic methodology allows two logical input bits to be used on a one-port by sending the bits separated in time. The resulting logic gate is faster than one relying on memristor's state switching, low power and requires only one memristor. We experimentally demonstrate working OR and XOR gates made with a single flexible Titanium dioxide sol-gel memristor.

**Keywords:** Memristor, sequential logic, ReRAM, OR, XOR, Boolean logic, Time-separated logic.

## 1 Introduction

The memristor is the recently-discovered [1] fourth fundamental element, joining the set of the resistor, inductor and capacitor. It was predicted to exist based on an expectation of symmetry in electromagnetic phenomena when applied to circuit theory [2], specifically in that it would be passive two-terminal device that would relate the two as-then-unrelated circuit measurables: charge,  $q$ , and magnetic flux,  $\varphi$ <sup>1</sup>. From knowledge about its electronic properties, Chua predicted that it would be a non-linear version of a resistor that possesses a memory, hence the name memristor, a contraction of memory-resistor.

Whilst Chua's theoretical contributions were not known to the wider chemical and physics communities, devices highly similar in constitution and operation to Strukov's memristor [1] were created and dubbed ReRAM, for Resistive Random Access Memory, after the use their inventors intended for them. What exactly constitutes a memristor or ReRAM device is a matter of debate, although it

---

<sup>1</sup> The other measurables being current and voltage, and the other five relationships being the definitions of current and voltage and the constitutive relations of the other three fundamental circuit elements.

has been suggested that they may be the same thing [3]. Both memristors and ReRAM have suggested uses as computer memory and both are believed to possess the same physical interactions and thus, in this paper, we shall deal with both under the name of memristors, where it is understood that a large part of the results presented here should be tested on ReRAM devices and are expected to work in the same way.

Both memristors [1] and ReRAM [4] have been suggested as possible low-power next-generation computer memory technology, however the field of ReRAM has been around for 20 years and has not yet produced a commercial product and Hewlett-Packard (the company that discovered the Strukov memristor) has been delaying their computer memory offering based on their memristor.

Chua's theoretical model of the memristor has been used to model neuronal synapses (see for example [5–7]) and to update the Hodgkin-Huxley model of neuronal membranes and axonal transport [8, 9]. It has been shown [10] that the experimental memristor spikes in a similar manner to those seen in axonal transport, where it is understood that neurons demonstrate a voltage spike in response to the current influx, and the spikes shown in [10] are current spikes in response to the voltage change. These current spikes have been seen in other memristor systems to ours and are generally ignored or dismissed as current transients. A view which will be tested with our devices in a forthcoming paper. Regardless of how these spikes arise, they are impossible (as far as we know and the literature states) to remove and thus it is our opinion that future uses of memristor technology will have to involve these spikes. Based on the relation to the brain's operation, we consider that memristor networks will be useful for neuromorphic computing, however in this paper we will demonstrate how a single memristor can be used as a Boolean logic gate by making use of the physical property of these current spikes, which can be done if we take an unconventional approach to logic assignment.

This is not the first paper on how to make logic gates with memristors. Strukov et al [11] resorted to using implication logic to design logic gates which required two memrsitors (IMP-FALSE logic is Turing complete, but somewhat unfamiliar to computer scientists). The most notable Boolean logic gates were simulated by Pershin and di Ventra [12] and required a memcapacitor, three or four memristive systems and a resistor. Before the gate was sent the two bits of data, a set of initialization pulses were required to be sent to put the gate into the correct state to give the correct answer. This system, however, is not true Boolean logic because these initialization pulses were different dependent on what the logic to follow would be. Thus the gate can not be considered to be operating only on the two bits of input data and is not a simple Boolean logic gate (it is a Turing machine doing a computation on several bits of data (Boolean input pulses and initialization pulses) which is capable of modeling a Boolean logic gate). Note also that this scheme was tested with memristor emulators, not real devices. There have been other more complex designs for memristor based Boolean logic gates, the simplest of which requires 11 circuit elements [13]. In this paper, we will demonstrate how to perform Boolean logic with a single memristor.

Although the memristor is credited with being the first computational device to combine memory and processing functionality in one, with the suggestion of an entirely new type of computer, remarkably there have been relatively few papers on how this new computer might work: most people have chosen to focus on stateful memory applications [11].

We will now demonstrate the physical properties of our memristors [14] and validate their reproducibility (section 3), explain the concept of how these spikes can be used to perform Boolean logic (section 4) and, as an example, demonstrate experimentally that a single memristor can act an OR (section 4.1) or and XOR (section 4.2) gate.

## 2 Methodology

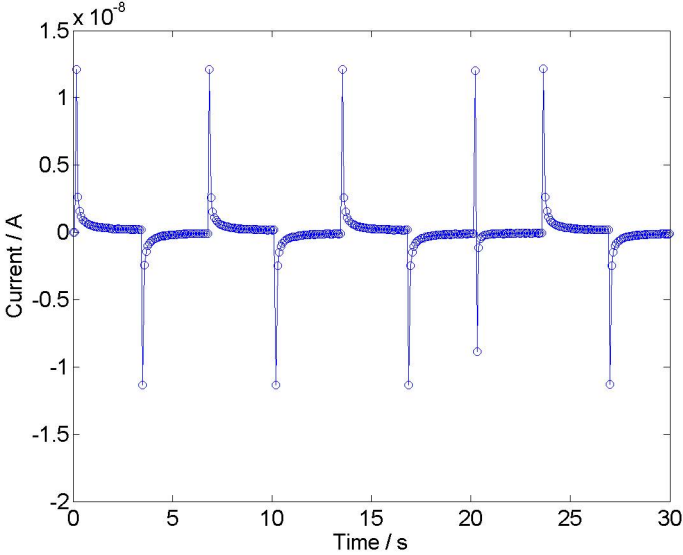
The memristors are flexible  $\text{TiO}_2$  sol-gel memristors with aluminium electrodes and were made as in [14] with the sol-gel created as in [15] (the memristor chosen was a curved-type memristor (see [14])). All tests on the memristor were performed with a Keithley 2400 Sourcemeter and data was recorded and analysed using MatLab. Each timestep was 0.02s. The voltages used and voltage waveforms varied as are discussed below.

## 3 Physical Properties of the Memristor

When there is a change in voltage,  $\Delta V$ , across a memristor the device exhibits a current spike, the physical cause of which is discussed at length in [10]. This spike is highly reproducible and repeatable and is related to the size of the voltage change ( $\Delta V$ ) [10]. The spike's size (as measured by the first measurement after the Keithley's changed voltage) is highly reproducible, the current then relaxes to a stable long-term value (this value is predictable and reproducible), and it takes approximately 2-3 seconds to get to this value.

This slow relaxation is thought to be the d.c. response of the memristor [10] and if a second voltage change happens within this time frame, its resulting current spike is different to that expected from the  $\Delta V$  alone. The size and direction of this current spike depends on the direction of  $\Delta V$ , the magnitude of  $\Delta V$  and the short-term memory of the memristor.

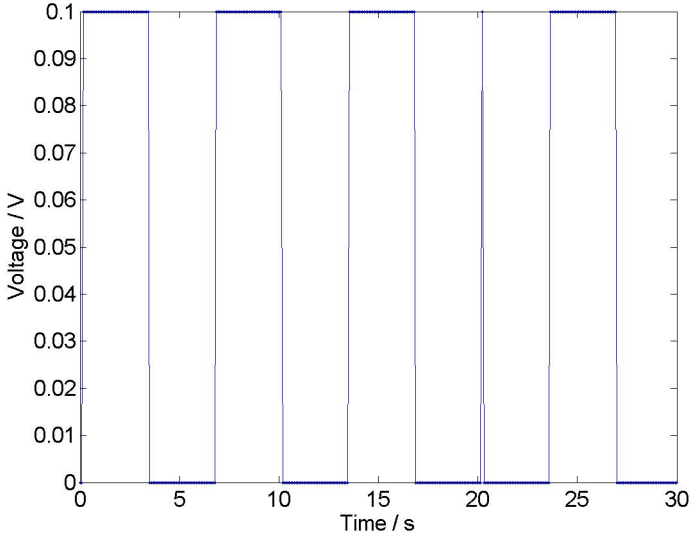
As an example, consider a memristor pulsed with a positive 1V voltage square wave as in figure 2 (where the pulses are repeated to demonstrate the repeatability) with a timestep of  $\approx 0.02s$ . The current response is shown in figure 1 and we can see there is a positive current spike associated with the  $+\Delta V$  and a perhaps less obvious negative current spike associated with the  $-\Delta V$  transition from  $+1V \rightarrow 0V$ . At approximately 20s, we shortened the square wave to a single time step, and the memory of the system has caused the response spike (responding to the  $-\Delta V$  to be smaller (and as it is smaller, it suggests that there is some physical property of the device which has not adjusted to its  $+V$



**Fig. 1.** The effect of adding spikes close in time. The response spikes are the negative current spikes. When a positive spike is included but not allowed to relax the corresponding negative spike is smaller.

value. See [16] for an discussion on why this physical property is predicted to be the oxygen vacancies in the  $\text{TiO}_2$ .) Thus the response is subtractive in current and additive in resistance state.

To try and understand the subtleties of this apparent ‘addition’, consider the following system: two voltages are sent to the memristor, one after the other separated by one timestep (i.e. before the memristor has equilibrated), where  $V_B > V_A$  and  $V_B = 0.12V$ , and figure 3 shows the size of the two resulting spikes as a function of increasing  $V_A$ . We look at two situations: 1,  $V_A(t) \rightarrow V_B(t+1)$ ; 2,  $V_B(t) \rightarrow V_A(t+1)$ . These two situations are drastically different if we look at the transitions,  $\Delta V$ , as situation 2 has a negative  $\Delta V_{B \rightarrow A}$ , all the other transitions are positive. Situation 1 shows that if the smaller voltage is sent first ( $V_A \rightarrow V_B$ ), the current of the first transition  $\Delta i_{0 \rightarrow A}$  increases with the size of  $V_A$ , and the second transition  $\Delta i_{A \rightarrow B}$  decreases with the size of  $\Delta V_A$ , due to the decrease in the effective  $\Delta V_{A \rightarrow B}$ . However, the sum of these two effects is non-linear, so that the total current transferred (approximated as the sum of the spikes here, but actually the area under the two current transients) is not the same as that shown for situation 2 (until  $V_B = V_A$ ). This shows that more current is being transferred and demonstrates that the spikes are dependent on  $\Delta V$ . Furthermore, it makes it clear that  $\Delta i_{0 \rightarrow A} + \Delta i_{A \rightarrow B} \neq \Delta i_{0 \rightarrow B} + \Delta i_{B \rightarrow A}$ , (except in the trivial case where  $V_B = V_A$ ) and that spike based ‘addition’ is non-commutative and therefore the order in which the spikes are sent is relevant.



**Fig. 2.** The input voltage for Test 1

## 4 Boolean Logic Using Current Spikes in Memristors

We can do Boolean logic with the spike interactions by sending the second bit of information one timestep (0.02s) after the first. We take the input as the current spikes from the voltage level. The output is the response current as measured after the 2<sup>nd</sup> bit of information. After a logic operation the device is zeroed by being taken to 0V for approximately 4s, and this removes the memristor’s memory.

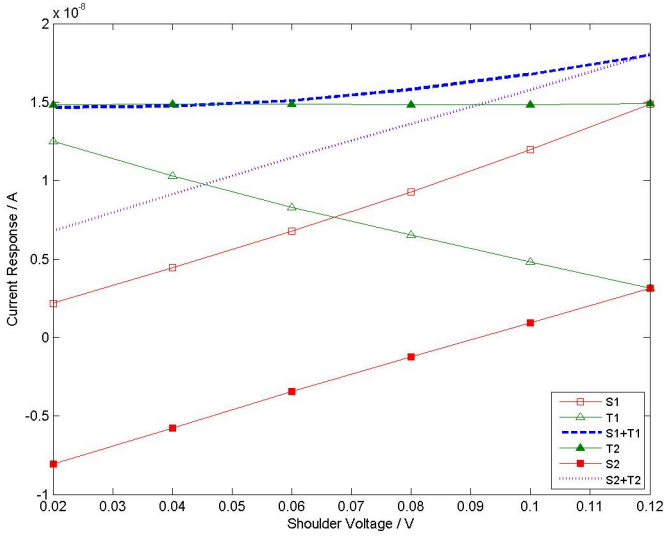
We have some freedom in how we assign the ‘1’ and ‘0’ states to device properties and these give different logic. The following examples will demonstrate some approaches and build an OR gate or an XOR gate.

### 4.1 OR Gate

The truth table for an OR gate is given in table 1, essentially, the output should be ‘1’ if either of the inputs was ‘1’. We take the ‘0’ output as being below a threshold current and the ‘1’ output as being above a threshold. The threshold is set to  $>18\text{nA}$  with the ‘0’ input being set of 0.01V and the ‘1’ as  $0.2\text{V}$ <sup>2</sup>, which gives the voltages below:

<sup>2</sup> Using ‘0’ as 0V was also tested, it works and is lower power but was not chosen as an example as it is a trivial case.





**Fig. 3.** The effects of the order the spikes are sent in to show that spike addition is non-commutative.  $S1 = \Delta i_{0 \rightarrow A}(t)$ ,  $T1 = i_{A \rightarrow B}(t + 1)$ ,  $T2 = \Delta i_{0 \rightarrow B}(t)$  and  $S2 = \Delta i_{B \rightarrow A}(t + 1)$ .  $S1$  and  $T1$  refer to the shoulder ( $S1$ ) and peak ( $T1$ ) currents resulting switching from  $0V \rightarrow V_A \rightarrow V_B$ .  $T2$  and  $S2$  refer to the peak ( $T2$ ) and shoulder ( $S2$ ) of switching from  $0V \rightarrow V_B \rightarrow V_A$ . In both cases  $V_B > V_A$ .

**Table 1.** OR Truth Table (inclusive OR)

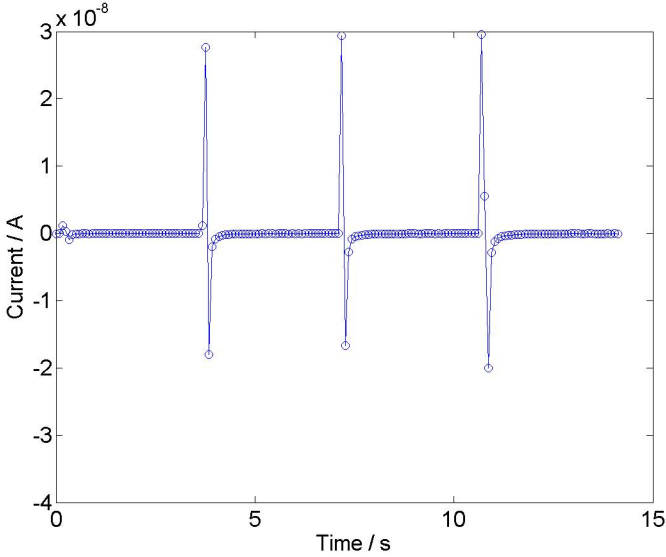
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

- 0, 0 = 0.01V, 0.01V
- 0, 1 = 0.01V, 0.2V
- 1, 0 = 0.2V, 0.01V
- 1, 1 = 0.2V, 0.2V.

Figure 4 shows the current data from the voltage inputs above. It can be seen that when a ‘1’ is input, there is a large spike output. To read the logical state of the device, one merely takes the current value as the second bit is read in.

## 4.2 A Logical System to Create an XOR Gate

The XOR truth table is shown in table 2. If we take logical ‘1’ to be the current resulting from a positive voltage and a logical ‘0’ to be the current resulting from



**Fig. 4.** OR Gate. Using ‘1’ equal to a current spike caused by a voltage change to 0.2V and ‘0’ equal to a current spike caused by a voltage change to 0.01V we can make a serial OR gate (where logical 1 is considered to be a current which is more than 5nA). At 0.04s ‘0, 0’ was input, giving peaks below the threshold i.e. ‘0’ as an output. The three large peaks are ‘1’ outputs resulting from ‘0,1’, ‘1,0’ and ‘1,1’ inputs.

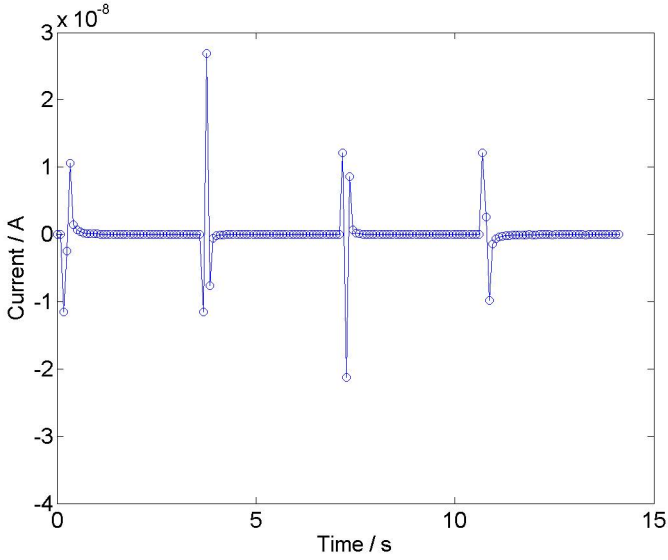
a negative voltage, then, the response is the current when the 2<sup>nd</sup> bit is input (not after, although it could be designed that way but it is slower). We get a high absolute value of current if and only if the two inputs are of different signs, i.e. we have 1 0 or 0 1 which gives us an exclusive OR operation. For this logical system, we used the same voltage level and allowed a change in sign to indicate logical zero or logical one:

- 0, 0 = -0.1V, -0.1V
- 0, 1 = -0.1V, +0.1V
- 1, 0 = +0.1V, -0.1V
- 1, 1 = +0.1V, +0.1V.

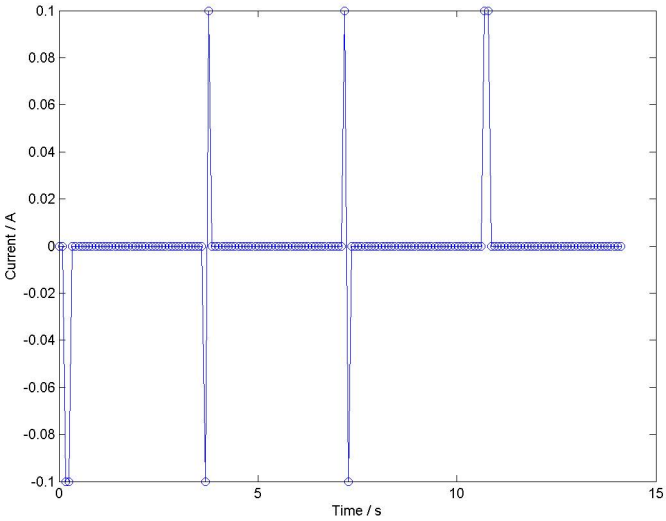
As an example, the input voltage is shown in figure 6 and the current output is shown in figure 5.

**Table 2.** XOR Truth Table (exclusive OR)

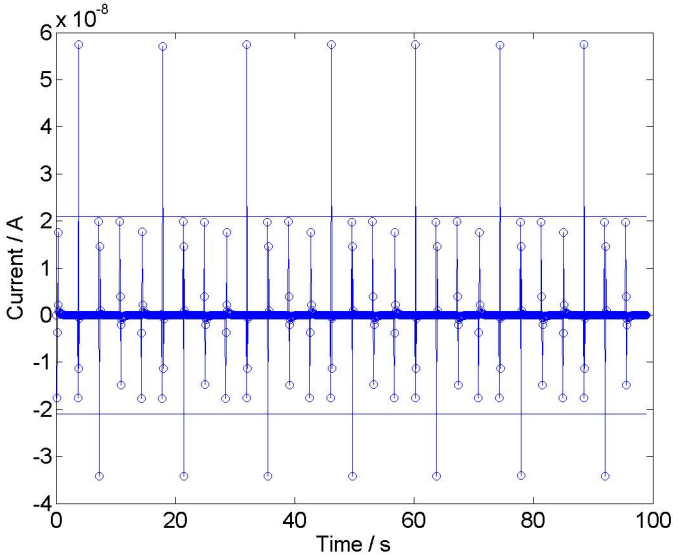
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0



**Fig. 5.** XOR gate, where a current response over  $\pm 1.25 \times 10^{-8}$  A is taken as one, as current response under that threshold is taken as zero



**Fig. 6.** The programming voltage for the XOR gate



**Fig. 7.** Reproducibility test of XOR function. Here the XOR truth table is run 7 times. The threshold between ‘1’ and ‘0’ is marked as shown.

With a pause between operations to allow the memristor to lose its memory, the XOR operation is reproducible, as shown in figure 7.

As  $\text{XOR } A = \text{NOT } A$ , if we always take the 2<sup>nd</sup> point after the first (and only bit in this case) as being the response bit (as we did above for the XOR gate), we have a NOT gate.

## 5 Conclusions

This type of approach is a serial logic gate where the bits are separated in time. This allows us to do logic operations with one memristor at the speed of the spikes (fast) rather than at the speed of equilibration (slow). This approach also allows us to do logic with a two terminal (one-port) device, the extra ‘complexity’ of the operation is contained within the time domain. Essentially, we use the memristor’s short-term memory to hold the first bit and do the calculation. This demonstrates that memristors can act as the processor and memory store in one. It also shows the bizarre property of the memory in a system being used to perform memoryless logic.

The memristor is acting similarly to a sequential logic circuit, where the combinatorial logic is combined with the memory store. Furthermore, the memristor logic gate is asynchronous because there is no need for a clock pulse, but there are issues of race hazard because the second bit must arrive within the time window of the memristor’s memory.

The speed of these operations is not too fast in this proof-of-principle, however, this is because of the speed at which the electrometer can properly measure a current response. Circuit theory suggests that these spikes should exist at shorter times, so we are confident that the devices can be sped up by sending the second spike in faster.

The memristor is very low power, especially if operated at the voltages and currents shown in the paper (it is possible to work at higher voltages if desired).

At the moment the output is a different circuit measurable to the input (i.e. the output is current and the input is voltage), it is necessary to convert from one to the other to enable the creation of logical circuits. However, we expect that a current pulse should propagate through a circuit [17] and cause a change in voltage across the next memristor, which could then be used to do the next operation and thus allow the creation of larger memristor logical circuits. We plan to do further work on testing this and investigating the possibility of using a second memristor as a  $V \rightarrow I$  transformer (based on the fact that previous  $\Delta V$  produced a  $\Delta I$ ).

**Acknowledgments.** This work was funded by EPSRC on grant EP/H014381/1.

## References

1. Strukov, D.B., Snider, G.S., Stewart, D.R., Williams, R.S.: The Missing Memristor Found. *Nature* 453, 80–83 (2008)
2. Chua, L.O.: Memristor - The Missing Circuit Element. *IEEE Trans. Circuit Theory* 18, 507–519 (1971)
3. Chua, L.O.: Resistance Switching Memories are Memristors. *Applied Physics A: Materials Science & Processing*, 765–782 (2011)
4. Waser, R.: *Nanoelectronics and Information Technology*. Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim (2003)
5. Cantley, K., Subramaniam, A., Stiegler, H., Chapman, R., Vogel, E.: Hebbian Learning in Spiking Neural Networks with Nano-Crystalline Silicon TFTs and Memristive Synapses. *IEEE Tran. Nanotechnology* 10, 1066–1073 (2011)
6. Zamarreno-Ramos, C., Carmuñas, L.A., Pérez-Carrasco, J.A., Masquelier, T., Serrano-Gotarredona, T., Linares-Barranco, B.: On Spike-Timing Dependent Plasticity, Memristive Devices and Building a Self-Learning Visual Cortex. *Frontiers in Neuromorphic Engineering* 5, 26(1)–26(20) (2011)
7. Howard, G.D., Gale, E., Bull, L., de Lacy Costello, B., Adamatzky, A.: Evolution of Plastic Learning in Spiking Networks via Memristive Connection. *IEEE Trans. Evolutionary Computation* 26, 711–719 (2012)
8. Chua, L., Sbitnev, V., Kim, H.: Hodgkin-Huxley Axon is made of Memristors. *Int. J. Bifur. Chaos* 22, 1230011(1)–1230011(48) (2012)
9. Chua, L., Sbitnev, V., Kim, H.: Neurons are Poised Near the Edge of Chaos. *Int. J. Bifur. Chaos* 22, 1250098(1)–1250098(49) (2012)
10. Gale, E.M., de Lacy Costello, B., Adamatzky, A.: Observation and Characterization of Memristor Current Spikes and their Application to Neuromorphic Computation. In: 2012 International Conference on Numerical Analysis and Applied Mathematics, ICNAAM 2012. AIP Conference Proceedings, vol. 1479, pp. 1898–1901. AIP Melvil, New York (2012)

11. Borghetti, J., Snider, G.D., Kuekes, P.J., Yang, J.J., Stewart, D.R., Williams, R.S.: ‘Memristive’ switches enable ‘stateful’ logic operations via material implication. *Nature* 464, 873–876 (2010)
12. Pershin, Y.V., di Ventra, M.: Neuromorphic, Digital and Quantum Computation with Memory Circuit Elements. *Proc. IEEE* 100, 2071–2080 (2012)
13. Pino, R.E., Bohl, J.W.: Self-Reconfigurable Memristor-Based Analog Resonant Computer, US Patent, US 8,274,312 B2
14. Gale, E., Pearson, D., Kitson, S., Adamatzky, A., de Lacy Costello, B.: Aluminium Electrodes Effect the Operation of Titanium Dioxide Sol-Gel Memristors, arXiv:1106:6293v1 (cond-mat.mtrl-ci), <http://arxiv.org/abs/1106.6293>
15. Gale, E., Mayne, R., Adamatzky, A., de Lacy Costello, B.: Drop-coated Titanium Dioxide Memristors, arXiv:1205:2885v2 (cond-mat.mtrl-ci), <http://arxiv.org/abs/1205.2885>
16. Gale, E.: The Missing Magnetic Flux in the HP Memristor Found, arXiv:1106:3170v1 (cond-mat.mtrl-ci), <http://arxiv.org/abs/1106.3170>
17. Gale, E., Matthews, O., de Lacy Costello, B., Adamatzky, A.: Beyond Markov Chains, Towards Adaptive Memristor Network-based Music Generation. In: Proceedings of the Annual Convention of Society of the study of Artificial Intelligence and the Simulation of Behaviour, AISB 2013, vol. 8, pp. 28–49 (2013)

# Light Ray Concentration Reduces the Complexity of the Wavelength-Based Machine on PSPACE Languages

Sama Goliaei and Mohammad-Hadi Foroughmand-Araabi

University of Tehran, Tehran, Iran  
{sgoliaei,foroughmand}@ut.ac.ir

**Abstract.** The wavelength-based machine, or simply  $w$ -machine, is an optical computational model, dealing with light rays and simple optical devices.  $w$ -Machine benefits from the parallel nature of light and co-existence of different wavelengths in a light ray to perform computation. In this paper, we have introduced a novel operation for  $w$ -machine, called the concentration operation, which enables to concentrate light rays as a single light ray, and check if the obtained light ray is dark or not, using white-black imaging. In this paper, we have investigated the impact of the concentration operation to computational complexity of  $w$ -machine for Turing PSPACE languages, and we have shown that every Turing PSPACE language is computable by a uniform series of concentration enabled  $w$ -machines, in polynomial time and exponential size.

**Keywords:** Unconventional Computing, Optical Computing, Wavelength-Based Machine, Concentration Enabled  $w$ -Machine, Computational Complexity

## 1 Introduction

Optical problem solving, a branch of unconventional computing, tries to use the light and optical devices to outperform existing methods to solve the problems. In a thread of this research area, specific solutions are provided for particular problems, by designing a suitable optical system for each instance of the given problem. Some research works have used Fourier optics to compute number and matrix multiplication [1]. Another method is provided by designing a graph-like structure by optical devices and optical fibers. Light rays are then passed through graph edges, and since the speed of light is limited, it is supposed that reaching light to a certain vertex at certain time is correspond to a solution for the problem. This method is used for some NP-complete problems, such as the Hamiltonian path and the subset sum problems [2–5]. It has been proved that, the length of required optical fibers in this method is exponential, unless  $P = NP$  [6]. Another technique in this research thread is based on optical filters, devices that restrict the path of light rays, for solving NP-complete problems, such as 3-SAT and Hamiltonian path problems [7]. The inverse of this method is

defined as the ray-tracing problem, which is the problem of computing whether a zero-width light ray reaches to a point within an optical system of reflective and refractive devices, and it is shown that the problem is undecidable in general from [8]. It is also shown that the ray tracing problem, for any  $d$ -dimension ( $d \geq 2$ ) optical systems consisting of a finite set of perpendicular and parallel reflective surfaces represented by a system of rational linear equalities, is PSPACE [8]. Wavelengths, as a property of light rays, are also used in this thread to solve the string matching problem [9]. Also, we have previously provided a solution for the 3-SAT problem, based on the property that different wavelengths pass through optical devices simultaneously [10, 11].

In contrast to the mentioned thread, another research thread in optical problem solving, tries to generalize optical ideas to solve the problems as an abstract computational machine, and analyze its computability and complexity issues. The continuous space machine (CSM), is a general computational machine, manipulating optical images by optical devices. It is assumed that the optical images in CSM have infinite resolution. Upper and lower bounds on the complexity of restricted versions of CSM have been provided, and it has been proved that a restricted version of CSM solves polynomial space problems on Turing machine in polynomial time and satisfies the parallel thesis [12, 13]. VLSIO is another abstract computing framework consisting of optical transformation components, and wirings between components. Upper and lower bounds for computing various optical functions on VLSIO have been provided, including results are used to obtain lower bounds for some more specific problems, such as Fourier transformation, convolution, multiplication and division [14, 15].

We have previously provided an abstract wavelength-based machine, named  $w$ -machine. This machine manipulates light rays with different wavelengths via basic operations, implemented by simple optical devices. We have provided Upper and lower bound on the complexity of  $w$ -machine, such as, the uniform  $w$ -machine simulates non-deterministic Turing machine in polynomial time plus the time required by Turing machine, and non-uniform  $w$ -machine computes all languages in polynomial time and exponential size [10]. We have also shown that almost all languages require exponential size to be solved on non-uniform  $w$ -machine [16]. In this paper, we define a new operation, the concentration operation, for the  $w$ -machine, and provide an optical implementation schema for it. We show that concentration enabled  $w$ -machine, the  $w$ -machine with concentration operation, computes any PSPACE language in polynomial time.

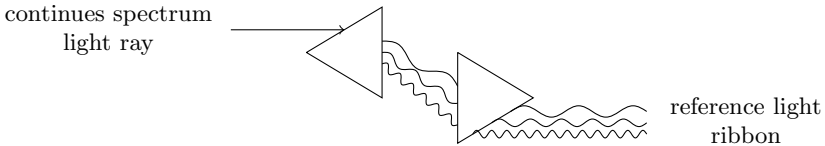
## 2 Concentration Enabled $w$ -Machine

The wavelength-based machine with concentration operation, or shortly the concentration enabled  $w$ -machine, is an abstract computational machine, which models computation via special forms of parallel traversing rays with possibly different wavelengths, and operates on them via simple optical devices. In this section, we first define  $w$ -tuples,  $w$ -sets, and concentrated  $w$ -sets to model parallel traversing light rays and their concentrated images. Then, we define basic operations and other basic concepts of the concentration enabled  $w$ -machine.

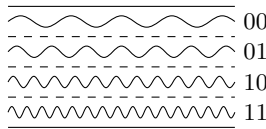


### 2.1 $w$ -Tuples, $w$ -Sets, and Concentrated $w$ -Sets

The  $n$ -bit concentration enabled  $w$ -machine operates on special forms of parallel traversing light rays, which form a light ribbon. The reference form of light ribbon which we call it the reference light ribbon, is obtained by passing a continuous spectrum light ray through two prisms, as it is shown in Fig. 1. In the obtained light ribbon, different wavelengths are sorted. Let us divide the width of the reference light ribbon into  $2^n$  wavelength intervals, and assign the obtained wavelength intervals into  $n$ -bit binary strings  $0^n$  to  $1^n$  in ascending order, as the example provided in Fig. 2. Let  $l$  be the width of the reference light ribbon. It is possible in the concentration enabled  $w$ -machine, to deal with light ribbons with width  $l/2^\Delta$ , for  $0 \leq \Delta \leq n$ . For a given light ribbon with width  $l/2^\Delta$ , we divide the width of the light ribbon into  $2^{n-\Delta}$  positions, each of which of length  $l/2^n$ , and assign each position to a binary string of length  $d = n - \Delta$ , from  $0^d$  to  $1^d$ , in ascending order, as the example provided in Fig. 3.



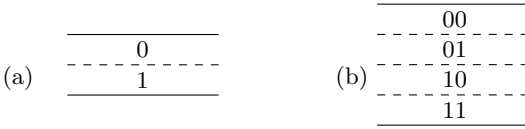
**Fig. 1.** Creating the reference light ribbon from a continuous spectrum light ray



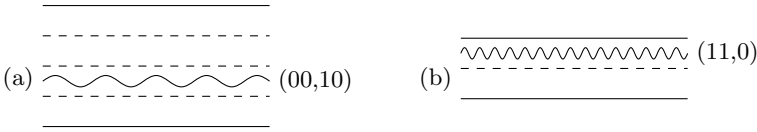
**Fig. 2.** Dividing the the reference light ribbon into  $2^n$  wavelength intervals, for  $n = 2$

A light ray with wavelengths from wavelength interval  $p$  ( $p \in \{0, 1\}^n$ ) traversing in position  $q$  ( $q \in \{0, 1\}^d$ ), is modeled as tuple  $(p, q)$ , which we call it a  $w$ -tuple. In the other words, an  $n$ -bit  $w$ -**tuple** of degree  $d$ , is a tuple  $(p, q)$  for  $p \in \{0, 1\}^n$  and  $q \in \{0, 1\}^d$ , which models wavelengths from wavelength interval  $p$ , traversing in position  $q$ , in a light ribbon with width  $l/2^{n-d}$ . In this definition,  $p$  is the *wavelength element* and  $q$  is the *position element* of the  $w$ -tuple. Examples of  $w$ -*tuples* are presented in Fig. 4.

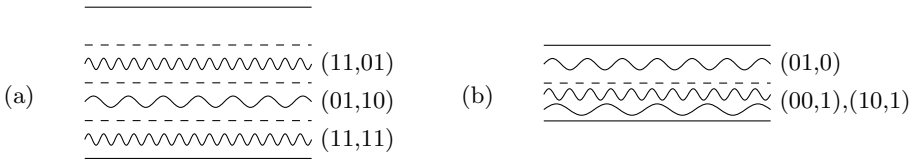
Now, a light ribbon with width  $l/2^{n-d}$  containing different wavelengths is modeled as an  $n$ -bit  $w$ -set of degree  $d$ , which is a set of  $n$ -bit  $w$ -tuples of degree  $d$ . Example of modeling light ribbons as  $w$ -sets are presented in Fig. 5.



**Fig. 3.** Dividing the width of a light ribbon into  $2^d$  positions (a)  $d = 2$  (b)  $d = 1$



**Fig. 4.** Examples of  $w$ -tuples(a)  $(00, 10)$  (b)  $(11, 0)$



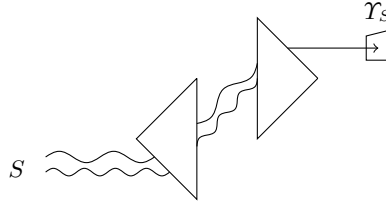
**Fig. 5.** Examples of  $w$ -sets (a)  $\{(11, 01), (01, 10), (11, 11)\}$  (b)  $\{(01, 0), (00, 1), (10, 1)\}$

In the concentration enabled  $w$ -machine, it is possible for wavelengths to traverse in different positions in comparison to reference light ribbon. We say that a  $w$ -tuple  $(p, q)$  is a **normal  $w$ -tuple** if and only if  $p = q$ . In the other words, a  $w$ -tuple is normal if and only if the wavelength is traverse in the same position as reference light ribbon. Similarly, a  $w$ -set is a **normal  $w$ -set**, if and only if it contains only normal  $w$ -tuples.

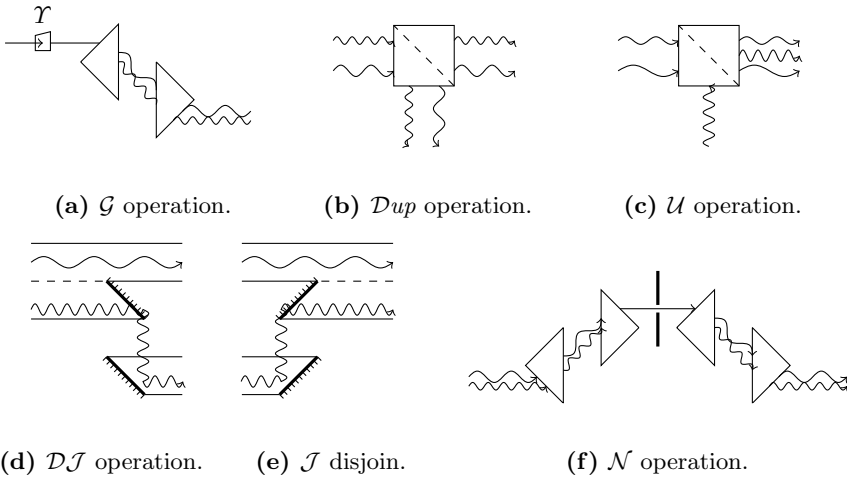
Suppose we pass a normal  $w$ -set  $N$  from two prisms, to concentrate all wavelengths (regardless of their places in the  $w$ -set) to obtain a single light ray. The obtained light ray is dark if and only if  $N = \emptyset$ . If we capture a positive white-black image from this light ray with a white-black optical sensitive sheet, as it is shown in Fig. 6, the obtained sheet is transparent in position of light ray if and only if  $S \neq \emptyset$ . We call this sheet, a **concentrated  $w$ -set**, denoted by  $\Upsilon_S$ , which is modeled as a Boolean variable, where  $\Upsilon_S = 1$  if  $S \neq \emptyset$ , and  $\Upsilon_S = 0$  is 0 otherwise.

## 2.2 Basic Concepts and Operations

An  $n$ -bit concentration enabled  $w$ -machine  $w$ , performs operations on  $n$ -bit  $w$ -sets via basic operations, and creates a set of  $m$ -bit binary strings as output,



**Fig. 6.** Creating a concentrated  $w$ -set  $\Upsilon_S$  from a given normal  $w$ -set  $S$



**Fig. 7.** Optical implementation schema for basic operations

for some  $1 \leq m \leq n$ . The basic operations of the machine are defined requiring constant number of physical devices and constant time to be applied, as follows:

- Concentration ( $\mathcal{C}$ ): Creates a concentrated  $w$ -set which is equal to 1 if and only if the given normal  $w$ -set is not empty. The operation is optically implemented by passing the given normal  $w$ -set from two prisms, and placing an optical sensitive sheet to capture positive white-black image of the outgoing ray from the prisms (see Fig. 6).
- Double concentration ( $\mathcal{DC}$ ): Creates a concentrated  $w$ -set  $\Upsilon_{A,B}$  which is equal to 1 if and only if both of the given normal  $w$ -sets  $A$  and  $B$  are not empty. The operation is implemented optically by passing two given  $w$ -sets from two prisms and capturing positive white-black images from two obtained light rays. Then, we place the obtained images on each other, and then, we capture a positive white-black image from them. Since this positive image is transparent if and only if  $A \neq \emptyset$  and  $B \neq \emptyset$ , hence, the image implements  $\Upsilon_{A,B}$ .

- Complete normal set creation ( $\mathcal{G}$ ): Creates a normal  $n$ -bit  $w$ -set containing all possible normal  $n$ -bit  $w$ -tuples. It is possible for this operation to take a concentrated  $w$ -set  $\mathcal{Y}$  as input. In this case, the operation creates a complete normal  $w$ -set if the given concentrated  $w$ -set is 1, and creates an empty  $w$ -set otherwise. The operation is optically implemented by passing a continuous spectrum light ray from two prisms, to obtain the reference light ribbon. In case of taking a concentrated  $w$ -set  $\mathcal{Y}$  as input,  $\mathcal{Y}$  is placed on the path of the light ray before reaching the first prism. Hence, if  $\mathcal{Y} = 0$ , the sheet is dark and no light reaches the prisms, and in case of  $\mathcal{Y} = 1$ , the reference light ribbon is created (see Fig. 7a).
- Duplication ( $\mathcal{D}up$ ): Duplicates a given  $w$ -set, and is optically implemented by passing the given  $w$ -set from a cubic beam-splitter (see Fig. 7b).
- Union ( $\mathcal{U}$ ): Creates the union of two given  $w$ -sets, and is optically implemented by passing the given  $w$ -sets from two different sides of a cubic beam-splitter (see Fig. 7c).
- Disjoin ( $\mathcal{D}\mathcal{J}$ ): Drops the first bit of the position element of each member of the given  $w$ -set, and classifies the members according to the dropped bit into two new  $w$ -sets. The operation is optically implemented by superposing the second half of a given  $w$ -set on the first half of it, using mirrors and a cubic beam-splitter (see Fig. 7d).
- Join ( $\mathcal{J}$ ): Creates the union of two given  $w$ -sets, and adds a 0 (1) bit to the position elements of the members of the first (second) given  $w$ -set. This operation can be considered as the reverse of the disjoin operation (see Fig. 7e).
- Normalization ( $\mathcal{N}$ ): Creates a  $w$ -set containing normal members of a given  $w$ -set. The operation is optically implemented by passing the given  $w$ -set through two prisms and passing the light ray coming out with the same angle as the ray used to create the reference light ribbon from two other prisms. Note that other light rays with different angles are blocked. (see Fig. 7f).
- Output declaration ( $\mathcal{O}_m$ ): Declares the set of  $m$ -bit prefixes of the members of a given  $w$ -set as the output set of the machine. In optical view, the given  $w$ -set is the output of the machine, in such a way that only the wavelengths are considered, and positions are ignored.

A concentration enabled  $w$ -machine is represented by a directed acyclic graph, where the vertices are machine operations and the edges explain how the output of one operation is used as the input for another operation. The input edges of the  $\mathcal{J}$  operations and the output edges of the  $\mathcal{D}\mathcal{J}$  operation are labeled to specify the value of the added or dropped bit. Two or more operations may be performed simultaneously if there is no directed path between them.

An  $n$ -bit concentration enabled  $w$ -machine  $w$  computes its output set, the output set the **language** of the machine, and is denoted by  $\mathbf{lang}(w)$ . The length of binary strings in  $\mathbf{lang}(w)$  is denoted by  $\mathbf{outwordlen}(w)$ . The parameter  $n$  is denoted by  $\mathbf{wordlen}(w)$ , which is the logarithm of the number of different required wavelengths. The length of the longest directed path in  $w$  is denoted by  $\mathbf{time}(w)$ , which indicates the time required by light to pass through  $w$  and

compute the output set. The total number of vertices in  $w$  is denoted by  $size(w)$ , which indicates the total number of optical devices in  $w$ . Each binary language (possibly with containing different length of strings) is computed by a series  $\{w\}_{n \in \mathbb{N}_0}$  of  $w$ -machines, where  $w_n$  computes  $n$ -bit binary strings in the language. A series  $\{w\}_{n \in \mathbb{N}_0}$  is **uniform** if and only if there is a Turing machine which takes  $1^n$  as input and creates the description of  $w_n$ , for all  $n \in \mathbb{N}_0$ .

*Example 1.* An example of a 3-bit concentration enabled  $w$ -machine  $w$  is presented in Fig. 8. In the presented machine, first, a  $\mathcal{G}$  operation creates a complete 3-bit normal  $w$ -set. Then, a  $\mathcal{DJ}$  operation is applied to create  $w$ -set  $A$  of degree 2, containing all  $w$ -tuples which the dropped bit is 0.

$$A = \{(000, 00), (001, 01), (010, 10), (011, 11)\}$$

Now, a  $\mathcal{J}$  operation is applied on  $A$ , to add a 1 to position elements, and create  $w$ -set  $B$ :

$$B = \{(000, 100), (001, 101), (010, 110), (011, 111)\}$$

A  $\mathcal{N}$  operation is applied on  $B$  to obtain normal  $w$ -set  $C$ . Since  $B$  does not contain any normal  $w$ -tuple, hence,  $C = \emptyset$ . Now, a concentration operation is applied on  $C$ . Since  $C = \emptyset$ , hence,  $\Upsilon_C$  is 0, and the  $w$ -set  $D$  created by the next  $\mathcal{G}$  operation is equal to  $\emptyset$ . A  $\mathcal{DJ}$  operation is then applied on  $D$ . Since,  $D = \emptyset$ , hence, the obtained  $w$ -set from  $\mathcal{DJ}$  operation is also empty. In the second branch, a  $\mathcal{G}$  operation is used to create a complete 3-bit normal  $w$ -set, and a  $\mathcal{DJ}$  operation is used to drop the first bit from the position element, and create  $w$ -set  $E$  containing all  $w$ -tuples which the dropped bit is 1.

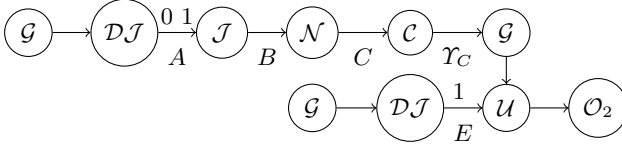
$$E = \{(100, 00), (101, 01), (110, 10), (111, 11)\}$$

Now, the union of  $E$  and the empty set obtained from the first branch is computed, which is equal to  $E$ , and the 2-bit prefixes of this  $w$ -set are declared as the output of the machine. Hence, the presented machine computes the language  $\{10, 11\}$ . In the presented example,  $wordlen(w) = 3$ ,  $outwordlen(w) = 2$ ,  $time(w) = 8$ , and  $size(w) = 10$ .

### 3 Concentration Enabled $w$ -Machines for PSPACE Languages

**Theorem 1.** *For each binary language  $L$  in PSPACE, there is a uniform series of concentration enabled  $w$ -machines computing  $L$  in polynomial time.*

*Proof.* Let  $T$  be a Turing machine computing  $L$  in polynomial space, and let  $s_T(n)$  and  $t_T(n)$  be the maximum space and time required by  $T$  to halt over  $n$ -bit input strings. We design a uniform series  $\{w\}_{n \in \mathbb{N}_0}$  of concentration enabled  $w$ -machines in such a form that  $\bigcup_{n \in \mathbb{N}_0} lang(w_n) = L$  and  $time(w_n)$  be a polynomial function of  $n$ . The idea of designing  $w_n$  is to find all  $n$ -bit input strings  $\sigma$  and configurations  $\phi_{first}$  and  $\phi_{last}$  of  $T$ , where  $\phi_{last}$  is the next configuration of  $\phi_{first}$ ,



**Fig. 8.** Example of a concentration enabled  $w$ -machine

if  $\sigma$  is given as the input string to  $T$ . Then, at the  $k$ -th step ( $1 \leq k \leq \lceil \lg t_T(n) \rceil$ ), we find all  $n$ -bit input strings  $\sigma$  and configurations  $\phi_{first}$  and  $\phi_{last}$  of  $T$ , where  $T$  reaches  $\phi_{last}$  from  $\phi_{first}$  after at most  $2^k$  steps, when  $\sigma$  is given as the input string to  $T$ .

Let  $Q$  = be the set of states,  $q_0$  be the starting state,  $q_a$  be the accepting state of  $T$ . We represent each configuration of  $T$  as a binary string with polynomial length  $l_c = \lceil \lg |Q| \rceil + \lceil \lg s_T(n) \rceil + s_T(n)$  as follows:

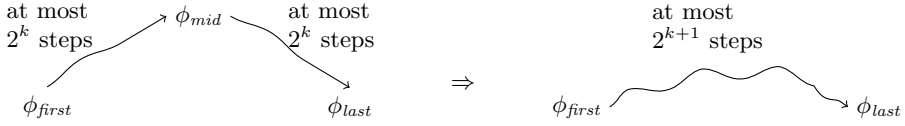
- $\lceil \lg |Q| \rceil$  bits are reserved to represent the current state
- $\lceil \lg s_T(n) \rceil$  bits are reserved to represent the current position of the head
- $s_T(n)$  bits are reserved to represent the tape

Let  $m = n + 2l_c$ . Note that since  $l_c$  is a polynomial function of  $n$ , hence,  $m$  is also a polynomial function of  $n$ . We design  $w_n$  in such a form that  $\text{wordlen}(w_n) = m$ , and the bits of the wavelength elements in  $w_n$  are reserved as follows:  $n$  bits are reserved to represent the input strings and  $2l_c$  bits are reserved to represent two configuration of  $T$ . Let  $A_{2^k}$  ( $0 \leq k \leq \lceil \lg t_T(n) \rceil$ ) be the normal  $w$ -set containing all normal  $w$ -tuples  $(\sigma\phi_{first}\phi_{last}, \sigma\phi_{first}\phi_{last})$ , where  $T$  reaches  $\phi_{last}$  from  $\phi_{first}$  after at most  $2^k$  steps, for input string  $\sigma$ . First we show how to create  $A_0$ , and then, we show how to create  $A_{2^{k+1}}$  from  $A_{2^k}$ .

To create  $A_{2^0}$ , we start from a  $\mathcal{G}$  operation and create a  $w$ -set  $S$  containing all  $m$ -bit normal  $w$ -tuples. Then, we use  $2^m - 1$   $\mathcal{DJ}$  operations, in the structure of a binary tree with depth  $m$ , to create  $2^m$   $w$ -sets  $S_0^m, \dots, S_1^m$ , where  $S_p = \{(p, \cdot)\}$  ( $p \in \{0, 1\}^m$ ). Now we ignore all  $w$ -sets  $S_{\sigma\phi_{first}\phi_{last}}$  which  $\phi_{last}$  is not the next configuration after  $\phi_{first}$  in  $T$  (for input string  $\sigma$  and configurations  $\phi_{first}$  and  $\phi_{last}$ ). Then,  $2^m - 1$   $\mathcal{J}$  operations in the structure of a binary tree with depth  $m$  are used to find  $w$ -set  $A_{2^0}$  as the union of all valid branches. Note that here, the transitive function of  $T$  is hard-wired in  $w_n$ . In the provided structure, the longest directed path for computing  $A_{2^0}$  is  $O(m)$ .

Now suppose we have already created  $A_{2^k}$  ( $0 \leq k < \lceil \lg t_T(n) \rceil$ ). Note that  $T$  reaches configuration  $\phi_{last}$  from configuration  $\phi_{first}$  over input string  $\sigma$  in at most  $2^{k+1}$  steps, if and only if there is a configuration  $\phi_{mid}$  which  $T$  reaches  $\phi_{mid}$  from  $\phi_{first}$ , and reaches  $\phi_{last}$  from  $\phi_{mid}$  in at most  $2^k$  steps, over input string  $\sigma$  (see Fig. 9). Thus,

$$\begin{aligned}
 & (\sigma\phi_{first}\phi_{last}, \sigma\phi_{first}\phi_{last}) \in A_{2^{k+1}} \Leftrightarrow \\
 & \exists \phi_{mid} : \{(\sigma\phi_{first}\phi_{mid}, \sigma\phi_{first}\phi_{mid}), (\sigma\phi_{mid}\phi_{last}, \sigma\phi_{mid}\phi_{last})\} \subseteq A_{2^k}
 \end{aligned}$$



**Fig. 9.** Obtaining members of  $A_{2^{k+1}}$  from the members of  $A_{2^k}$

We use  $2^m - 1$   $\mathcal{DJ}$  operations, in the structure of a binary tree with depth  $m$ ,  $m$  pairs of  $\mathcal{Dup}$  and  $\mathcal{J}$  operations, and a  $\mathcal{N}$  operation, to create  $2^m$  normal  $w$ -sets  $S_0^m, \dots, S_{1^m}^m$ , where  $S_p = \{(p, p)\}$  ( $p \in \{0, 1\}^m$ ), as it is shown in Fig. 10. Note that  $S_{\sigma\phi_{first}\phi_{last}} = \{(\sigma\phi_{first}\phi_{last}, \sigma\phi_{first}\phi_{last})\}$  if  $\phi_{last}$  is reachable from  $\phi_{first}$  in at most  $2^k$  steps, and is empty otherwise. Since we have to compare each  $w$ -set  $S_{\sigma\phi_{first}\phi_{last}}$  to  $2^{l_c}$  other  $w$ -sets  $S_{\sigma\phi_{last}p}$  ( $p \in \{0, 1\}^{l_c}$ ), hence, we use  $2^{l_c} - 1$  duplication operations for each  $S_{\sigma\phi_{first}\phi_{last}}$  in the structure of a binary tree with depth  $l_c$ , to create  $2^{l_c}$  copies of  $w$ -set  $S_{\sigma\phi_{first}\phi_{last}}$ .

Now, for each  $n$ -bit string  $\sigma$  and configurations  $\phi_{first}, \phi_{last}$ , and  $\phi_{mid}$ , we use a double concentration operation on  $S_{\sigma\phi_{first}\phi_{mid}}$  and  $S_{\sigma\phi_{mid}\phi_{last}}$  to obtain concentrated  $w$ -set  $\Upsilon_{\sigma\phi_{first}\phi_{last}}^{\phi_{mid}}$ . By definition of the  $\mathcal{DC}$  operation,  $\Upsilon_{\sigma\phi_{first}\phi_{last}}^{\phi_{mid}}$  is 1 if and only if  $S_{\sigma\phi_{first}\phi_{mid}} \neq \emptyset$  and  $S_{\sigma\phi_{mid}\phi_{last}} \neq \emptyset$ . In the other words,

$$\Upsilon_{\sigma\phi_{first}\phi_{last}}^{\phi_{mid}} = 1 \Leftrightarrow$$

$$\{(\sigma\phi_{first}\phi_{mid}, \sigma\phi_{first}\phi_{mid}), (\sigma\phi_{mid}\phi_{last}, \sigma\phi_{mid}\phi_{last})\} \subseteq A_{2^k}$$

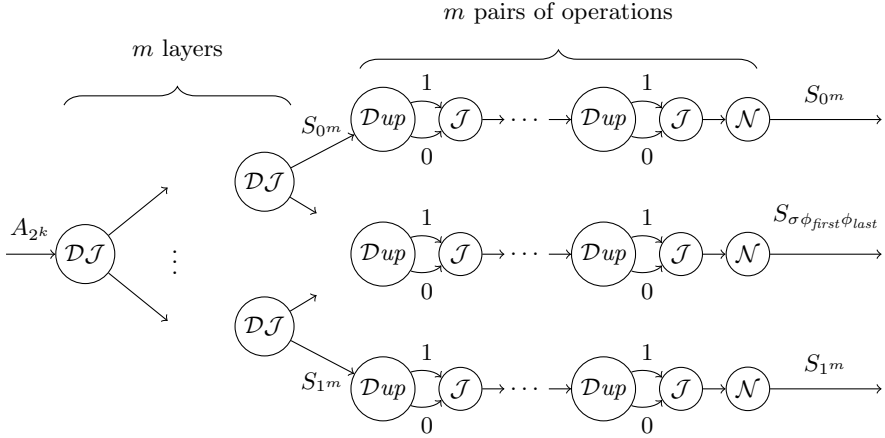
Thus,

$$\Upsilon_{\sigma\phi_{first}\phi_{last}}^{\phi_{mid}} = 1 \Rightarrow (\sigma\phi_{first}\phi_{last}, \sigma\phi_{first}\phi_{last}) \in A_{2^{k+1}}$$

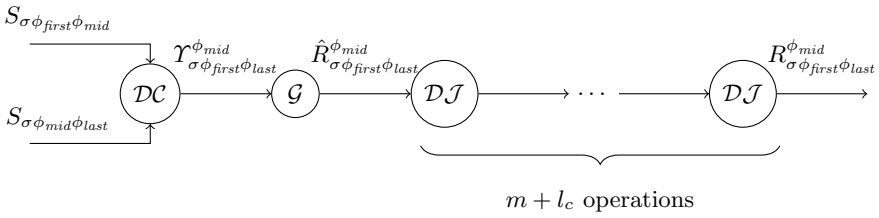
Now, we use a  $\mathcal{G}$  operation taking  $\Upsilon_{\sigma\phi_{first}\phi_{last}}^{\phi_{mid}}$  as input and creating complete normal  $w$ -set  $\hat{R}_{\sigma\phi_{first}\phi_{last}}^{\phi_{mid}}$  as output. Now we use  $m$  disjoint operations to obtain  $R_{\sigma\phi_{first}\phi_{last}}^{\phi_{mid}}$ , as it is shown in Fig. 11, which is equal to  $\{(\sigma\phi_{first}\phi_{last}, \sigma\phi_{first}\phi_{last})\}$  if and only if  $\phi_{last}$  is reached from  $\phi_{first}$  via  $\phi_{mid}$  in at most  $2^{k+1}$  steps, over input string  $\sigma$ .

Now, we use  $2^{m+l_c} - 1$  union operations over  $w$ -sets  $R_{\sigma\phi_{first}\phi_{last}}^{\phi_{mid}}$  in such a form that for all input strings  $\sigma$  and configurations  $\phi_{first}, \phi_{last}, \phi_{mid}$ , to obtain  $w$ -set  $A_{2^{k+1}}$ , where  $(\sigma\phi_{first}\phi_{last}, \sigma\phi_{first}\phi_{last}) \in A_{2^{k+1}}$  if and only if  $(\sigma\phi_{first}\phi_{last}, \sigma\phi_{first}\phi_{last}) \in A_{2^{k+1}}$ . Thus, by  $m$   $\mathcal{J}$  operations and a normalization operation,  $A_{2^{k+1}}$  is obtained, as it is shown in Fig. 12.

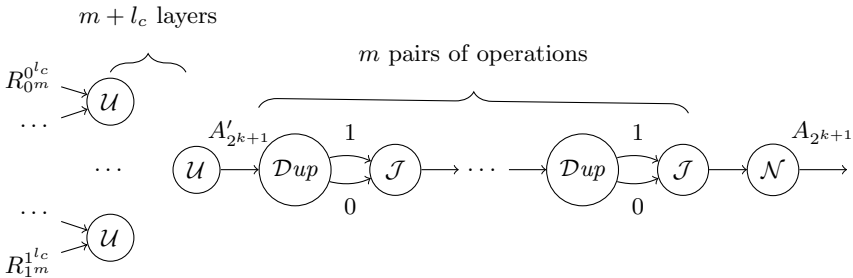
After  $\lceil \lg t_T(n) \rceil$  steps,  $A_{2^{\lceil \lg t_T(n) \rceil}}$  is obtained. Since  $T$  halts after at most  $t_T(n)$  steps over each  $n$ -bit input string, thus, each  $n$ -bit input string  $\sigma$  is accepted by  $T$  if and only if for the initialization configuration  $\phi_{init}$  and some accepting configuration  $\phi_{accept}$ ,  $(\sigma\phi_{init}\phi_{accept}, \sigma\phi_{init}\phi_{accept}) \in A_{2^{\lceil \lg t_T(n) \rceil}}$ . Thus, we use  $2^m - 1$   $\mathcal{DJ}$  operations to find  $w$ -set  $F$  containing all  $w$ -tuples  $(\sigma\phi_{init}\phi_{accept}, \sigma\phi_{init}\phi_{accept})$ , where  $\phi_{init}$  is the initialization configuration for  $T$  over input string  $\sigma$ , and the state in  $\phi_{accept}$  is equal to  $q_{accept}$ . Now, an output declaration operation  $\mathcal{O}_n$  declares the set of all  $n$ -bit strings which  $T$  accept them.



**Fig. 10.** Creating  $w$ -sets  $S_0^m$  to  $S_1^m$ , indicating reachable configurations in at most  $2^k$  steps



**Fig. 11.** Creating  $w$ -set  $R_{instr\phi_{first}\phi_{last}}^{\phi_{mid}}$  which is equal to  $\{(\sigma\phi_{first}\phi_{last}, \sigma\phi_{first}\phi_{last})\}$  if and only if  $\phi_{mid}$  is reachable from  $\phi_{first}$ , and  $\phi_{last}$  is reachable from  $\phi_{mid}$  in at most  $2^k$  steps



**Fig. 12.** Creating  $w$ -set  $A_{2^{k+1}}$  containing configuration pairs which are reachable in at most  $2^{k+1}$  steps over input string  $\sigma$



Since the length of the longest directed path to compute  $A_{2^0}$  is  $O(m)$ , we compute  $A_{2^{k+1}}$  from  $A_{2^k}$  by adding length  $O(m)$  to the longest directed path,  $m$  and  $\lceil \lg t_T(n) \rceil$  are polynomial functions of  $n$ , and the output is generated from  $A_{2^{\lceil \lg t_T(n) \rceil}}$  by adding length  $O(m)$  to the longest directed path, hence,  $time(w_n)$  is a polynomial function of  $n$ . Note that  $wordlen(w_n) = m$  is also a polynomial function of  $n$ . Thus, the uniform concentration enabled  $w$ -machines compute PSPACE languages in polynomial *time*. Note that since  $size(w)$  is exponential, there is no polynomial time Turing machine generating the description of  $w_n$  in polynomial time. But, the provided series  $\{w_n\}_{n \in \mathbb{N}_0}$  to compute PSPACE languages are uniform in a sense that there is a Turing machine  $T_w$  generating the next operation of each operation, in polynomial time.

## 4 Conclusion and Future Works

$w$ -Machine is a computational model that performs computation by applying basic operations on  $w$ -sets, which are sets of binary string tuples. There is a pure optical implementation schema for  $w$ -machine, in which each  $w$ -set is implemented as a set of parallel light rays with different wavelengths, and basic operations are implemented by simple optical devices such as prisms, beam-splitters, and mirrors.

In this paper, we have defined a new operation in  $w$ -machine, called concentration operation. This operation detects the emptiness of  $w$ -sets, and is optically implemented by white-black positive imaging from a light ray obtained concentration of light rays in a  $w$ -set via prisms. We have shown that adding the concentration operation to  $w$ -machine, improves its computational power, such that PSPACE languages can be computed by uniform series of concentration enabled  $w$ -machines in polynomial time. Although the provided series of concentration enabled  $w$ -machines to compute a PSPACE language may have exponential size, but, they are uniform in a sense that there is a Turing machine which computes the next operations of a each operation in polynomial time.

As a future work, we will try to figure out how concentration enabled  $w$ -machine would be powerful in computation, exactly. We may also extend the idea of the concentration operation by defining optical logical operation on concentrated  $w$ -sets. Note that complexity of simple  $w$ -machine is not completely understood yet, specifically for Turing complexity classes such as PSPACE, which is a possible thread in our future studies. Another subject of the future works is to try to physically construct concentration enabled  $w$ -machines with larger sizes in comparison to previous experiments on physical construction of  $w$ -machine.

## References

1. Yu, F.T.S., Jutamulia, S., Yin, S. (eds.): Introduction to Information Optics, 1st edn. Academic Press (2001)
2. Haist, T., Osten, W.: An optical solution for the traveling salesman problem. Optics Express 15(16), 10473–10482 (2007)

3. Oltean, M., Muntean, O.: Exact cover with light. *New Generation Computing* 26(4), 329–346 (2008)
4. Oltean, M., Muntean, O.: Solving the subset-sum problem with a light-based device. *Natural Computing* 8(2), 321–331 (2009)
5. Muntean, O., Oltean, M.: Deciding whether a linear diophantine equation has solutions by using a light-based device. *Journal of Optoelectronics and Advanced Materials* 11(11), 1728–1734 (2009)
6. Oltean, M.: Solving the hamiltonian path problem with a light-based computer. *Natural Computing* 6(1), 57–70 (2008)
7. Dolev, S., Fitoussi, H.: Masking traveling beams: Optical solutions for NP-complete problems, trading space for time. *Theoretical Computer Science* 411, 837–853 (2010)
8. Reif, J.H., Tygar, D., Yoshida, A.: The computability and complexity of ray tracing. *Discrete and Computational Geometry* 11, 265–287 (1994)
9. Oltean, M.: Light-based string matching. *Natural Computing* 8(1), 121–132 (2009)
10. Goliaei, S., Jalili, S.: An optical wavelength-based computational machine. *International Journal of Unconventional Computing* (in press)
11. Goliaei, S., Jalili, S.: An optical wavelength-based solution to the 3-SAT problem. In: Dolev, S., Oltean, M. (eds.) *OSC 2009*. LNCS, vol. 5882, pp. 77–85. Springer, Heidelberg (2009)
12. Woods, D., Naughton, T.J.: Optical computing. *Applied Mathematics and Computation* 215(4), 1417–1430 (2009)
13. Woods, D.: Upper bounds on the computational power of an optical model of computation. In: Deng, X., Du, D. (eds.) *ISAAC 2005*. LNCS, vol. 3827, pp. 777–788. Springer, Heidelberg (2005)
14. Reif, J.H., Tyagi, A.: Energy complexity of optical computations. In: *Proceedings of the 2nd IEEE Symposium on Parallel and Distributed Processing*, pp. 14–21 (1990)
15. Barakat, R., Reif, J.H.: Lower bounds on the computational efficiency of optical computing systems. *Applied Optics* 26(6), 1015–1018 (1987)
16. Goliaei, S., Foughmand-Araabi, M.-H.: Lower bounds on the complexity of the wavelength-based machine. In: Durand-Lose, J., Jonoska, N. (eds.) *UCNC 2012*. LNCS, vol. 7445, pp. 94–105. Springer, Heidelberg (2012)

# Small Steps toward Hypercomputation via Infinitary Machine Proof Verification and Proof Generation

Naveen Sundar Govindarajulu, John Licato, and Selmer Bringsjord

Department of Computer Science  
Department of Cognitive Science  
Rensselaer AI & Reasoning Laboratory  
Rensselaer Polytechnic Institute  
110 8<sup>th</sup> Street, Troy, NY 12180 USA  
{govinn,licatj,selmer}@rpi.edu

**Abstract.** After setting a context based on two general points (that humans appear to reason in infinitary fashion, and two, that actual hypercomputers aren't currently available to directly model and replicate such infinitary reasoning), we set a humble engineering goal of taking initial steps toward a computing machine that can reason in infinitary fashion. The initial steps consist in our outline of automated proof-verification and proof-discovery techniques for theorems independent of PA that seem to require an understanding and use of infinitary concepts (e.g., Goodstein's Theorem). We specifically focus on proof-discovery techniques that make use of a marriage of analogical and deductive reasoning (which we call *analogico-deductive reasoning*).

## 1 Context: Infinitary Reasoning, Hypercomputation, and Humble Engineering

Bringsjord has repeatedly pointed out the obvious fact that the behavior of formal scientists, taken at face value, involve various infinitary structures and reasoning. (We say "at face value" to simply indicate we don't presuppose some view that denies the reality of infinite entities routinely involved in the formal sciences.) For example, in (Bringsjord & van Heuveln 2003), Bringsjord himself operates as such a scientist in presenting an infinitary paradox which to his knowledge has yet to be solved. And he has argued that apparently infinitary behavior constitutes a grave challenge to AI and the Church-Turing Thesis (e.g., see Bringsjord & Arkoudas 2006, Bringsjord & Zenzen 2003). More generally, Bringsjord conjectures that every human-produced proof of a theorem independent of Peano Arithmetic (PA) will make use of infinitary structures and reasoning, when these structures are taken at face value.<sup>1</sup> We have ourselves

---

<sup>1</sup> A weaker conjecture along the same line has been ventured by Isaacson, and is elegantly discussed by Smith (2007).

designed logico-computational logics for handling infinitary reasoning (e.g., see the treatment of the infinitized wise-man puzzle: Arkoudas & Bringsjord 2005), but this work simply falls back on the human ability to carry out induction on the natural numbers: it doesn't dissect and explain this ability. Finally, it must be admitted by all that there is simply no systematic, comprehensive model or framework anywhere in the formal/computational approach to understanding human knowledge and intelligence that provides a theory about how humans are able to engage with infinitary structures. This is revealed perhaps most clearly when one studies the fruit produced by the part of formal AI devoted to producing discovery systems: such fruit is embarrassingly finitary (e.g., see Shilliday 2009).

Given this context, we are interested in exploring how one might give a machine the ability to reason in infinitary fashion. We are not saying that we in fact have figured out how to give such ability to a computing machine. Our objective here is much more humble and limited: it is to push forward in the *attempt* to engineer a computing machine that has the ability to reason in infinitary fashion. Ultimately, if such an attempt is to succeed, the computing machine in question will presumably be capable of outright hypercomputation. But the fact is that from an engineering perspective, we don't know how to create and harness a hypercomputer. So what we must first try to do, as explained in (Bringsjord & Zenzen 2003), is pursue engineering that initiates the attempt to engineer a hypercomputer, and takes the first few steps. In the present paper, the engineering is aimed specifically at giving a computing machine the ability to, in a limited but well-defined sense, reason in infinitary fashion. Even more specifically, our engineering is aimed at building a machine capable of at least providing a strong case for a result which, in the human sphere, has hitherto required use of infinitary techniques.

## 2 Review of Incompleteness and PA

In this section, we give a very brief review of Gödelian incompleteness and Peano Arithmetic from a purely syntactic point of view. The presentation here parallels that in (Ebbinghaus, Flum & Thomas 1984). First we give a brief overview of first-order logic.

**First-Order Logic (FOL).** The language of FOL includes an infinite collection of variables,  $\mathcal{V} = \{x_1, x_2, \dots, x_\omega\}$ . For convenience, instead of using, for example,  $x_{34}$ , we use  $x, y, z, \dots$  as meta-variables ranging over the actual variables. We also have access to the constant symbols  $\mathcal{C} = \{c_1, c_2, \dots, c_\omega\}$ , the function symbols  $\mathcal{F} = \{f_0, f_1, \dots, f_\omega\}$  and the predicate symbols  $\mathcal{P} = \{P_0, P_1, \dots, P_\omega\}$ . We also have the logical connectives  $\{\forall, \exists, \neg, \Rightarrow, \Leftrightarrow, \wedge, \vee\}$ .

The set of grammatically correct FOL terms  $\mathcal{T}$  are then given by the smallest set that includes the following terms:

1. The atomic terms:  $c_i$  or  $x_j$ .
2. The compound terms  $f_j(t_1, \dots, t_n)$ , where each  $t_i \in \mathcal{T}$ .

The set of grammatically correct FOL formulae  $\mathcal{L}$  are then given by the smallest set that includes the following formulae:

1. the atomic formulae:  $P_k(t_1, \dots, t_n)$ , where each  $t_i \in \mathcal{T}$ ;
2.  $(\neg\phi)$ , for every  $\phi \in \mathcal{L}$ ;
3.  $(\phi \oplus \psi)$ , for every  $\phi, \psi \in \mathcal{L}$  and for  $\oplus$  one of the binary connectives  $\{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$ ;
4.  $(\forall x \phi)$  and  $(\exists x \phi)$  for every  $\phi \in \mathcal{L}$  and  $x$  a variable.

When using FOL, we usually restrict the language to a finite set of constant, function, and predicate symbols  $\sigma$ , the *signature*.

Associated with FOL are many standard proof calculi, such as natural deduction (Pelletier 1999), resolution (Robinson 1965), etc. Given a proof calculus  $\mathcal{P}$ , a formula  $\phi$  being provable from a set of formulae  $\Gamma$  is denoted by  $\Gamma \vdash_{\mathcal{P}} \phi$ . Usually, in metalogical work, if the proof calculus is standard (sound and complete; see (Ebbinghaus et al. 1984)), then the particulars of  $\mathcal{P}$  are often of little importance and the notation used is  $\Gamma \vdash \phi$ .

**Incompleteness of a Theory.** A *theory*  $\Gamma$  in FOL is any set of formulae. A theory  $\Gamma$  is called *negation-complete* if for every formula  $\phi$  in the language of  $\Gamma$ , we have either  $\Gamma \vdash \phi$  or  $\Gamma \vdash \neg\phi$ . A theory is *negation-incomplete* or just *incomplete* if there is at least one such  $\phi$  such that  $\Gamma \not\vdash \phi$  and  $\Gamma \not\vdash \neg\phi$ . Such a statement is said to be *independent* of the theory.

As most readers will recall, Gödel's first incompleteness theorem states that any sufficiently strong theory of arithmetic that has certain desired attributes is incomplete. (See Smith 2007 for a detailed and accessible introduction to Gödel's incompleteness theorems.) Peano Arithmetic (PA) is one of the smallest incomplete theories that covers all of standard number theory.

PA is a first-order theory with the signature  $\{0, +, *, \text{succ}\}$  intended to model arithmetic. PA has six axioms and one axiom schema for induction:

$$\begin{aligned}
 A_1 &: \forall x(0 \neq \text{succ}(x)) \\
 A_2 &: \forall x \forall y(\text{succ}(x) = \text{succ}(y) \Rightarrow x = y) \\
 A_3 &: \forall x(x + 0 = x) \\
 A_4 &: \forall x \forall y(x + \text{succ}(y) = \text{succ}(x + y)) \\
 A_5 &: \forall x(x * 0 = 0) \\
 A_6 &: \forall x \forall y(x * \text{succ}(y) = (x * y) + x) \\
 \text{Induction Schema} &: (\phi(0) \wedge \forall x(\phi(x) \Rightarrow \phi(\text{succ}(x)))) \Rightarrow \forall \phi(x)
 \end{aligned}$$

One natural<sup>2</sup> statement provably independent of PA is Goodstein's Theorem, reviewed in the next section.

<sup>2</sup> In Gödel's original work, which of course doesn't detract from the seminal aspects of the work in the least, we have only unnatural formula schemas the instances of which are proved to be independent of PA, and an ingenious recipe for producing such schemas.

### 3 Review of Goodstein’s Theorem

The numerical examples are borrowed from (Potter 2004, Smith 2007), but the definitions have been slightly condensed.

**Definition 1** ( $b_n(r)$ ). *The base  $n$  representation of a number  $r$ ,  $b_n(r)$ , is a notation in which  $r$  is represented as the sum of powers of  $n$  and where the exponents are also powers of  $n$ , etc. Example:  $266 = 2^{2^{(2^{2^0}+2^0)}} + 2^{(2^{2^0}+2^0)} + 2^{2^0}$*

**Definition 2** ( $Grow_k(n)$  :). *Take the pure base  $k$  representation of  $n$ . Replace all  $k$  by  $k + 1$ . Compute the number obtained. Subtract one from the number.*

*Example:  $b_2(19) = 2^{2^{2^0}} + 2^{2^0} + 2^0$ , therefore*

$$Grow_2(19) = 3^{3^{3^0}} + 3^{3^0} + 3^0 - 1 = 3^{3^3} + 3 = 7625597484990$$

**Definition 3 (The Goodstein Sequence for  $m$ )**. *For any natural number, the Goodstein sequence for  $m$  is*

- $m$ ,
- $Grow_2(m)$ ,
- $Grow_3(Grow_2(m))$ ,
- $Grow_4(Grow_3(Grow_2(m))), \dots$

Some example values are shown in Figure 1

m										
2	2	2	1	0						
3	3	3	3	2	1	0				
4	4	26	41	60	83	109	139	...	11327 <small>(96th term)</small>	...
5	15	$\sim 10^{13}$	$\sim 10^{155}$	$\sim 10^{2185}$	$\sim 10^{36306}$	$10^{695975}$	$10^{15151337}$	...		

**Fig. 1.** Goodstein sequences for  $m$  with  $m \in \{2, 3, 4, 5\}$

**Theorem 1 (Goodstein’s Theorem)**. *For all natural numbers, the Goodstein sequence reaches zero after a finite number of steps.*

**Theorem 2 (Unprovability of Goodstein’s Theorem).** *Goodstein’s theorem is not provable in Peano Arithmetic (PA) (or any equivalent theory of arithmetic).*

All known proofs of Goodstein’s Theorem use infinitary constructs one way or another. The proofs either require infinite sets (beyond finitary arithmetic theories such as PA), or the proofs require non-finitary rules such as the  $\omega$ -rule.

The  $\omega$ -rule is a rule of inference to be used with arithmetic theories. The  $\omega$ -rule is to be added to other systems of inference, either resolution or natural deduction. This infinitary rule is of the following form

$$\frac{\phi(\bar{0}), \phi(\bar{1}), \dots}{\forall x \phi(x)} \quad \omega\text{-rule}$$

The above rule has an infinite number of premises and is unsuitable for implementation in a standard, non-hypercomputational computer. The advantage of the above rule is that Peano Arithmetic in a system which has this rule is *negation-complete*; i.e., for every  $\phi$  in the appropriate language, either  $\text{PA} \vdash_{\omega} \phi$  or  $\text{PA} \vdash_{\omega} \neg\phi$ .

A restricted  $\omega$ -rule is a finite form of the rule which still preserves negation-completeness for PA. The disadvantage is that proof-verification and discovery both fail to be even semi-decidable. Therefore, only approximations of this rule are implementable. Let us assume that we are dealing with a first-order language of arithmetic. This language has only  $\{0, 1, +, *\}$  as non-logical symbols, along with equality. Given this setting, we can invoke *nice* theories: these are consistent, decidable, and allow representations (Ebbinghaus et al. 1984).<sup>3</sup> The first incompleteness theorem states that for such theories there are statements that cannot be proved (and the negation of which also cannot be proved) from the theory in question.

With the restricted  $\omega$ -rule the incompleteness result does not hold. Assume that we have computer programs or machines operating over representations of numerals and proofs. Then if we have a program  $m$  which for all  $n \in \mathbb{N}$ ,  $m : \bar{n} \mapsto \rho(\Gamma, \phi(\bar{n}))$ . That is, for every natural number  $n$  and some formula  $\phi$  with one free variable,  $m$  produces a proof of  $\phi(\bar{n})$  from some set of axioms  $\Gamma$ . Given this, one form of the restricted  $\omega$ -rule is as follows:

$$\frac{\Gamma \quad m}{\forall x \phi(x)}$$

The most accessible reference for the above stated results is (Baker, Ireland & Smaill 1992). All these results are available in (Ebbinghaus et al. 1984, Franzén 2004).

Even the restricted  $\omega$ -rule is beyond trivial machine implementation, as, in the general case, a proof-verification system that handles the rule should be able to check in all possible cases if the program supplied halts with the correct

---

<sup>3</sup> Roughly put, if a theory allows representations, then it can prove facts about the primitive-recursive relations and functions.

proof. This motivates us to consider more cognitively plausible proof-discovery techniques. We feel that analogical reasoning is a general mechanism that is exploited quite frequently in problem-solving in mathematics (see e.g. (Polya 1954)).

## 4 Partial Proof-Sketch Generation in ADR

Before we describe the proof-sketch generation process, we briefly go over the Slate system for proof construction.

### Slate

Slate is a graphical proof-construction environment based on natural deduction, which includes support for constructing proofs in propositional logic, first-order logic, and several modal logics. Slate also has the ability to automatically discover proofs via resolution, by calling ATPs; e.g., SNARK (Stickel 2008). This feature allows one to utilize Slate in a hybrid mode to construct proofs that are semi-automated.<sup>4</sup> Proofs in Slate can be viewed as a directed acyclic graph  $\mathcal{G} = \{\langle \mathcal{F}, \mathcal{I}, E \rangle\}$  with two types of nodes: the formula nodes  $\mathcal{F}$  and the inference rule nodes  $\mathcal{I}$ . Each inference rule node corresponds to an application of the inference rule in the proof and has as parents the premises of the rule, and as children the conclusion of the rule. Each formula node  $f$  has a complex structure comprised of a unique identifier  $\text{id}$ , the formula  $\phi$  corresponding to the node, and a set of identifiers,  $\mathbb{T}$ , corresponding to the *scope* under which the current formula was derived. More concisely, each formula node  $f := \langle \text{id}, \phi, \mathbb{T} \rangle$ . To prove a formula  $\phi$  from a set of premises  $\Gamma$ , one needs to construct a graph that has a formula node  $f = \langle \text{id}, \phi, \mathbb{T} \rangle$  such that the identifiers in  $\mathbb{T}$  correspond to nodes of the following form  $\langle \text{id}_p, \gamma, \{\} \rangle$  where  $\gamma \in \Gamma$  for all  $\text{id}_p \in \mathbb{T}$ . This graph-based approach avoids the usual rigid row-based linearity of formal proofs in favor of more cognitively realistic agent-controlled “workspaces” the formal nature of which generally aligns with Kolmogorov-Uspenskii (= KU) machines.<sup>5</sup>

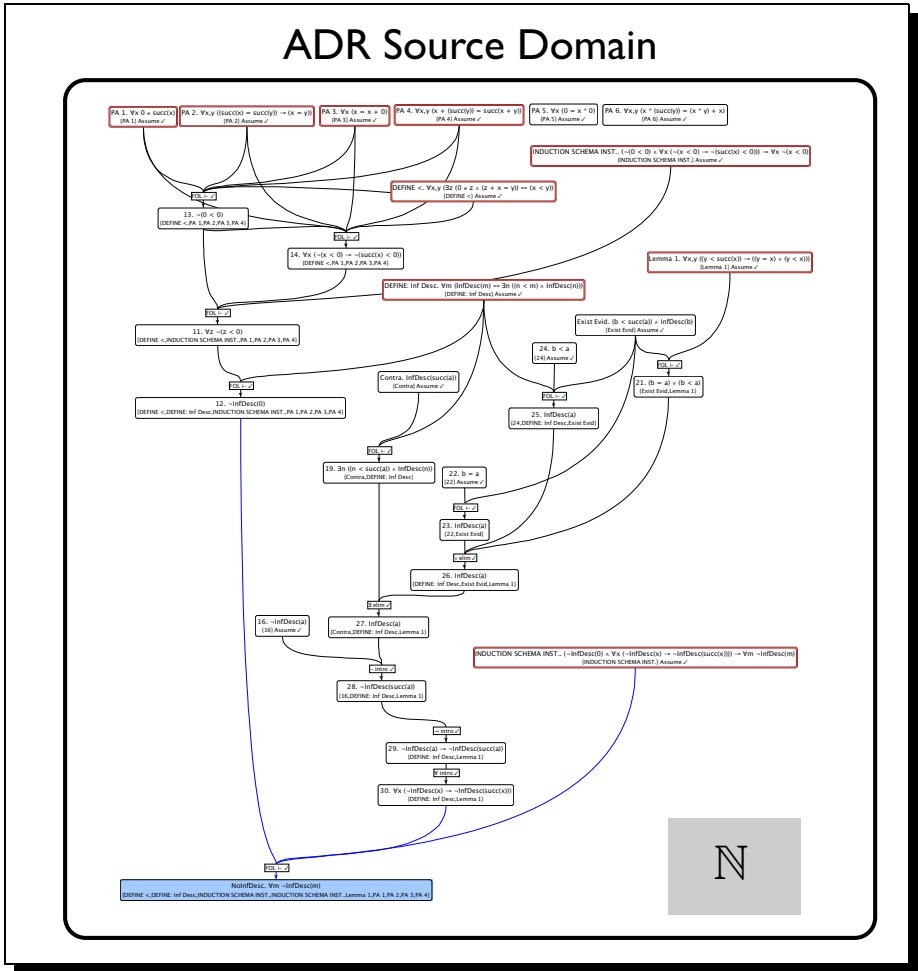
### Analogico-Deductive Reasoning

Proof discovery is often aided by analogical reasoning, particularly when the high-level description of the proof has some analogical similarity to a known proof in a different domain. For example, the proof of Gödel’s First Incompleteness Theorem has a well-known analogical similarity to the proof of the Liar Paradox, one which the present authors have taken steps toward automating and generating computationally. This process, which goes from the initial analogical insight to the desired formal proof, we have named *Analogico-Deductive*

<sup>4</sup> For an overview of an earlier version of Slate, see (Bringsjord, Taylor, Shilliday, Clark & Arkoudas 2008).

<sup>5</sup> These machines introduced in (Kolmogorov & Uspenskii 1958). For a brief discussion of KU machines in connection with the Church-Turing Thesis, see (Bringsjord & Govindarajulu 2011).





**Fig. 2.** Semi-automated proof from PA and a lemma that there is no infinite descent in  $\mathbb{N}$

*Reasoning* (ADR), a combination of analogical and hypothetico-deductive reasoning (Licato, Bringsjord & Hummel 2012, Bringsjord & Licato 2012). ADR works by analogically mapping a known proof in a **source domain** to a partially known proof in a **target domain**. Once this mapping is found, *analogical transfer* is used to fill out as much of the target domain’s proof as possible. Finally, within the target domain, the recently acquired knowledge is subjected to domain-specific verification in order to eliminate contradictions.

Of course, the above is a brutally simplified version of the ADR process. In a fully autonomous ADR system, there must be many other mechanisms in place in order to ensure that the process of analogical mapping and transfer (which is often somewhat imprecise and flexible) does not produce output that is invalid

in a particular domain, such as a non-well-formed-formula in a formal domain. This can occur in cases in which the source domain is less formal than the target domain; for example, a source domain with an informal understanding of the Liar Paradox and a target domain which is a fully formalized proof of Gödel’s First Incompleteness Theorem. This can be at least partially remedied by the use of what we call *patching operators*. These are essentially domain-specific rules meant to emulate the expertise of a reasoner familiar with the domain in question. For more information, see (Licato, Govindarajulu, Bringsjord, Pomeranz & Gittelsohn 2013).

### ADR Used to Move toward Proving Goodstein’s Theorem

We can now give a more in-depth summary of an ADR process that starts with an analogy, and outputs one of the key steps in Goodstein’s Theorem. The key step here is that there is *some* analogical similarity between the natural numbers  $\mathbb{N}$  and the ordinal numbers up to  $\epsilon_0 = \omega^{\omega^{\omega^{\dots}}}$ , the “epsilon-naught” numbers. This is important, as proofs about just the natural numbers can be quite simple. For example, Figure 2 contains a proof that there is no strictly decreasing sequence of natural numbers from PA and the lemma:

$$\forall x \forall y (y < \text{succ}(x)) \Rightarrow (y = x) \vee (y < x)$$

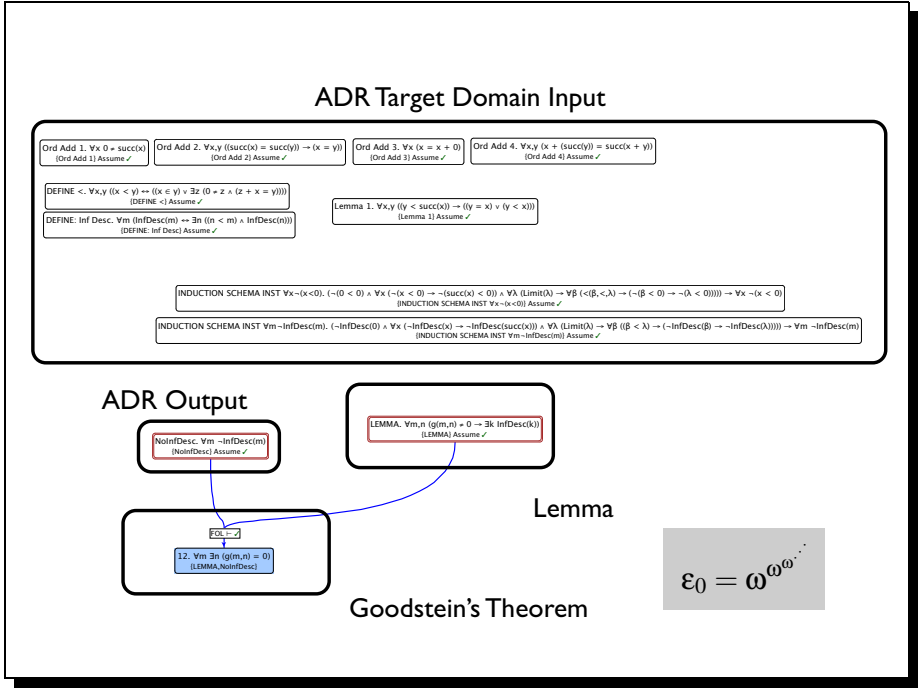
A similar statement holding for  $\epsilon_0$  is a key step in the proof of Goodstein’s Theorem. Given the source domain  $S$  consisting of the Peano-Arithmetic axioms and a proof that there is no sequence of natural numbers that form a strict infinitely descending sequence  $\forall m \neg \text{InfDesc}(m)$  for  $\mathbb{N}$ , and a target domain  $T$  consisting of some similar set-theoretic axioms describing ordinal arithmetic of  $\epsilon_0$ , ADR successfully suggests a corresponding target theorem  $\forall m \neg \text{InfDesc}(m)$  for  $\epsilon_0$  (that there is no infinitely descending sequence of ordinal numbers). The source domain for the ADR process is the Slate workspace in Figure 2; the target is the Slate workspace in Figure 3.

For the analogical mapping step, we use our system Modifiable Engine for Tree-based Analogical Reasoning (META-R), which is based on the matching system originally described by Owen (1990). Our implementation builds on Owen’s system by utilizing newer techniques for efficient graph matching, such as those based on Linear Program Rounding and Bipartite Matching.<sup>6</sup> META-R finds the matching between the axioms in the two domains, and on the basis of this matching transfers some of the intermediate steps in the source domain’s proof, which are then subjected to further analysis by the target domain.

The result of this first step is a high-level proof-sketch, pictured in Figure 3. This proof-sketch can then be transformed into an informal (or large-step) proof using standard automated theorem-proving technologies.

---

<sup>6</sup> We do not elaborate on these here in the interest of staying on topic, but for more information we again point the interested reader to (Licato et al. 2013).



**Fig. 3.** Workspace containing a high-level proof sketch partially filled in by ADR with Figure 2 as input

## 5 Conclusion and Future Work

The ADR process briefly described herein is meant to be an overview of an approach which attempts to simulate the reasoning processes that might allow an experienced mathematician to go from a simple analogical insight to a full formal proof of some theorem that is infinitary in nature. Obviously, much more work is needed to automate such a process, and we do not in this paper claim that full human-level analogico-deductive reasoning ability is achieved by the current version of our system. We simply present a framework that we hope can emphasize the role of ADR in the discovery of complex, infinitary proofs, such as those that establish Goodstein’s Theorem.

Our current and future work involves expanding on much of what was described in this paper. META-R will continue to be augmented and modified using the best-known techniques for solving the computationally difficult problem that analogical matching represents, and we hope to make this tool available for use by other interested researchers. We will also further develop the method of applying ADR to proof generation presented in Licato et al. (2013), and briefly described in the present paper. We suspect that the “Master Argument”

(Smith 2007), which uses Tarski's Theorem and matches the route which Gödel himself regarded to be the most perspicuous one to incompleteness, might be amenable to our approach.

## References

- Arkoudas, K., Bringsjord, S.: Metareasoning for Multi-agent Epistemic Logics. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3487, pp. 111–125. Springer, Heidelberg (2005), <http://kryten.mm.rpi.edu/arkoudas.bringsjord.clima.crc.pdf>
- Baker, S., Ireland, A., Smaill, A.: On the Use of the Constructive Omega-Rule Within Automated Deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 214–225. Springer, Heidelberg (1992)
- Bringsjord, S., Arkoudas, K.: On the Provability, Veracity, and AI-Relevance of the Church-Turing Thesis. In: Olszewski, A., Wolenski, J., Janusz, R. (eds.) Church's Thesis After 70 Years, pp. 66–118. Ontos Verlag, Frankfurt (2006), [http://kryten.mm.rpi.edu/ct\\_bringsjord\\_arkoudas\\_final.pdf](http://kryten.mm.rpi.edu/ct_bringsjord_arkoudas_final.pdf); This book is in the series Mathematical Logic, edited by W. Pohlers, T. Scanlon, E. Schimmerling, R. Schindler, H. Schwichtenberg
- Bringsjord, S., Govindarajulu, N.S.: In Defense of the Unprovability of the Church-Turing Thesis. *Journal of Unconventional Computing* 6, 353–373 (2011); Preprint available at [http://kryten.mm.rpi.edu/SB\\_NSg\\_CTnotprovable\\_091510.pdf](http://kryten.mm.rpi.edu/SB_NSg_CTnotprovable_091510.pdf)
- Bringsjord, S., Licato, J.: Psychometric Artificial General Intelligence: The Piaget-MacGyver Room. In: Wang, P., Goertzel, B. (eds.) Theoretical Foundations of Artificial General Intelligence. Atlantis Press (2012), [http://kryten.mm.rpi.edu/Bringsjord\\_Licato\\_PAGI\\_071512.pdf](http://kryten.mm.rpi.edu/Bringsjord_Licato_PAGI_071512.pdf)
- Bringsjord, S., Taylor, J., Shilliday, A., Clark, M., Arkoudas, K.: Slate: An Argument-Centered Intelligent Assistant to Human Reasoners. In: Grasso, F., Green, N., Kibble, R., Reed, C. (eds.) Proceedings of the 8th International Workshop on Computational Models of Natural Argument, CMNA 2008, Patras, Greece, pp. 1–10 (2008), [http://kryten.mm.rpi.edu/Bringsjord\\_etal\\_Slate\\_cmna\\_crc\\_061708.pdf](http://kryten.mm.rpi.edu/Bringsjord_etal_Slate_cmna_crc_061708.pdf)
- Bringsjord, S., van Heuveln, B.: The Mental Eye Defense of an Infinitized Version of Yablo's Paradox. *Analysis* 63(1), 61–70 (2003)
- Bringsjord, S., Zenzen, M.: Superminds: People Harness Hypercomputation, and More. Kluwer Academic Publishers, Dordrecht (2003)
- Ebbinghaus, H.D., Flum, J., Thomas, W.: *Mathematical Logic*. Springer, New York (1984)
- Franzén, T.: Transfinite Progressions: A Second Look at Completeness. *Bulletin of Symbolic Logic*, 367–389 (2004)
- Kolmogorov, A., Uspenskii, V.: On the Definition of an Algorithm. *Uspekhi Matematicheskikh Nauk* 13(4), 3–28 (1958)
- Licato, J., Bringsjord, S., Hummel, J.E.: Exploring the Role of Analogico-Deductive Reasoning in the Balance-Beam Task. In: Rethinking Cognitive Development: Proceedings of the 42nd Annual Meeting of the Jean Piaget Society, Toronto, Canada (2012), <https://docs.google.com/open?id=0B1S661sacQp6NDJOYzVXajJMwVU>
- Licato, J., Govindarajulu, N.S., Bringsjord, S., Pomeranz, M., Gittelsohn, L.: Analogico-deductive Generation of Gödel's First Incompleteness Theorem from the Liar Paradox. In: Proceedings of the 23rd Annual International Joint Conference on Artificial Intelligence, IJCAI 2013 (2013)

- Owen, S.: *Analogy for Automated Reasoning*. Academic Press (1990)
- Pelletier, J.: A Brief History of Natural Deduction. *History and Philosophy of Logic* 20, 1–31 (1999)
- Polya, G.: *Induction and Analogy in Mathematics*. Princeton University Press, Princeton (1954); This is Volume I of *Mathematics and Plausible Reasoning*. Volume II is *Patterns of Plausible Inference*
- Potter, M.: *Set Theory and its Philosophy: A Critical Introduction*. Oxford University Press, Oxford (2004)
- Robinson, J.: A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM (JACM)* 12(1), 23–41 (1965)
- Shilliday, A.: *Elisa: A New System for AI-assisted Logico-mathematical Scientific Discovery Incorporating Novel Techniques in Infinite Model Finding*. PhD thesis, Rensselaer Polytechnic Institute (2009)
- Smith, P.: *An Introduction to Gödel's Theorems*. Cambridge University Press, Cambridge (2007)
- Stickel, M.E.: *SNARK: SRI's New Automated Reasoning Kit SNARK* (2008), <http://www.ai.sri.com/~stickel/snark.html>

# Secure Information Transmission Based on Physical Principles

Dima Grigoriev<sup>1</sup> and Vladimir Shpilrain<sup>2,\*</sup>

<sup>1</sup> CNRS, Mathématiques, Université de Lille, 59655, Villeneuve d'Ascq, France  
dmitry.grigoryev@math.univ-lille1.fr

<sup>2</sup> Department of Mathematics, The City College of New York, New York, NY 10031  
shpil@groups.sci.ccny.cuny.edu

**Abstract.** We employ physical properties of the real world to design a protocol for secure information transmission where one of the parties is able to transmit secret information to another party over an insecure channel, without any prior secret arrangements between the parties. The distinctive feature of this protocol, compared to all known public-key cryptographic protocols, is that neither party uses a one-way function. In particular, our protocol is secure against (passive) computationally unbounded adversary.

## 1 Introduction

In public-key cryptography, one of the basic functionalities is *encryption*, which is performed over an open channel, without any prior secret arrangements between the parties engaged in a cryptographic protocol.

More specifically, the scenario is as follows. One of the parties, Alice, wants to transmit to another party, Bob, some secret information (say, a secret number  $m$ ). She can use an encryption algorithm to produce an encryption  $E(m)$  of her secret number  $m$ . She then transmits  $E(m)$  to Bob over an open (i.e., insecure) channel, and Bob uses a decryption algorithm to recover  $m$  from  $E(m)$ . All algorithms and all transmissions are supposed to be known to the public and yet the cryptosystem should be secure; this is the so-called *Kerckhoffs's principle*, which is considered mandatory in public-key cryptography. Therefore, in particular, Bob's decryption algorithm should involve a private key that only Bob knows. Also, the encryption function  $m \rightarrow E(m)$  should be *one-way*, which means, informally, that it is efficient to compute but hard to invert (at least, on most inputs). "Efficient" here means that the function of an input can be computed in time polynomial in the size of an input. Thus, "hard to invert" roughly means that there is no algorithm of time complexity polynomial in the size of an input that would find a pre-image of any given input.

It was shown in [4] that for a public-key encryption protocol to be secure, it has to employ at least one one-way function. On the other hand, a necessary

---

\* Research of the second author was partially supported by the NSF grants DMS-0914778 and CNS-1117675.

condition for the existence of one-way functions is that  $\mathbf{P}$  is not equal to  $\mathbf{NP}$ , where  $\mathbf{P}$  is the class of functions computable in deterministic polynomial time, and  $\mathbf{NP}$  is the class of functions computable in non-deterministic polynomial time. It is a central open problem in complexity theory whether or not these two classes are equal, with the prevalent opinion being that they are not. See [2] for a general background on this famous problem.

One immediately obvious thing is that a definition of any complexity class depends on how one formalizes the concept of being “computable”. The universally accepted formalization now is that a function is computable if there is a *Turing machine* that computes this function. Again, see [2] for a general background on Turing machines; here we just say that a Turing machine is a simple mechanism that manipulates symbols on a strip of tape according to a table of rules. This machine was described in 1936 by Alan Turing. A Turing machine that is able to simulate any other Turing machine is called a *universal Turing machine*. A more mathematically oriented definition with a similar “universal” nature was introduced by Alonzo Church, whose work on lambda calculus intertwined with Turing’s in a formal theory of computation known as the Church-Turing thesis. The thesis states that Turing machines are indeed a good formalization of the intuitive notion of computability and provide a precise definition of an algorithm.

It is the main purpose of this paper to show that there are secure encryption protocols that do not employ any one-way functions. On the intuitive level, when combined with the aforementioned results, this yields a feeling that there are “naturally computable” functions that cannot be computed by a Turing machine, but we do not make any formal statements in that direction at this time.

We note that our procedures are rather practical and only use physical properties of objects encountered by people in everyday life.

The structure of the paper is as follows. In Section 2, as a “warm up”, we show how to use a natural algorithm for a special case of a cryptographic primitive called *secret sharing*. More specifically, our special case is: a dealer distributes “shares” of a secret (which could be a secret number) to two parties, the idea being that only when the two parties get together, they can recover the whole secret, but neither alone can. This problem may seem trivial: if the secret number is  $n$ , then the dealer can just split  $n$  as  $n = n_1 + n_2$  and give  $n_1$  to one of the parties and  $n_2$  to the other. What we want to achieve however is that, after the two parties have recovered the whole secret, they end up still not knowing the other party’s share. This is obviously impossible if the secret is  $n = n_1 + n_2$  and the shares are  $n_1$  and  $n_2$ . In our Section 2, we show how to solve this problem by using a natural algorithm that does not, in fact, employ any non-trivial physics; just everyday experience. It is unknown (at least, to us) whether this problem can be solved by a procedure implementable on a Turing machine. What is interesting though is that this question is equivalent to the following one, that looks almost purely mathematical. Alice knows a point  $(x_1, y_1)$  in the plane, while Bob knows a point  $(x_2, y_2)$ . Can they exchange information in such a way that each ends up knowing the slope  $s = \frac{y_2 - y_1}{x_2 - x_1}$  of the line connecting their

points, but neither ends up knowing the other party's point  $(x_i, y_i)$ ? We note that it is fairly easy to arrange secret sharing between more than two parties without any party having to reveal their share to recover the secret, see [3] or [5]. At the end of Section 2, we give an application (pointed out to us by Bren Cavallo) of our method to Yao's "millionaires' problem" [7].

Then, in Section 3, we describe an encryption protocol where a secret number is encrypted as the length of a piece of an elastic rope, which consists of two parts with different moduli of elasticity, one of them being secret. Again, physical properties that are used in this protocol are very simple, and the protocol can be easily implemented in real life, although it is only suitable for transmitting secret information over a short distance. What is important is that this protocol is secure even against (passive) computationally unbounded adversary. Since a computationally unbounded adversary can use "encryption emulation attack" (i.e., reproduce encryption algorithm with all possible randomness and then compare the results to the actual encrypted message), security against such an adversary can only mean one thing: there are many possible combinations of different messages with different random choices that result in the same encryption. This is exactly what happens in our protocol.

We admit that the encryption protocol in Section 3 is rather impractical, but we believe it is helpful in understanding why what we call "nature-based cryptography" provides for what the "usual", complexity-based, cryptography cannot possibly provide: security against (passive) computationally unbounded adversary.

In Section 4, we give a more detailed analysis of the encryption emulation attack in our scenario, and in Section 5, we discuss security against *active* adversary, i.e., an adversary who does not just observe and process information, but can interfere with the protocol itself.

Finally, in our main Section 6, building on the ideas of Section 3, we give a rather simple and practical encryption protocol where instead of a piece of rope we use an electrical circuit to transmit data.

## 2 Secret Sharing between Two Parties

The problem that we address in this section is as follows. The dealer wants to distribute information ("shares") about a secret number  $s$  to two parties, Alice and Bob, in such a way that only when the two parties get together, they can recover the whole secret, but neither alone can.

One possible solution (which is a special case of Shamir's solution [6] of a more general problem) is: the dealer represents the number  $s$  as  $s = \frac{y_2 - y_1}{x_2 - x_1}$  and gives  $(x_1, y_1)$  to Alice and  $(x_2, y_2)$  to Bob. Problem solved.

Now we want to have an additional functionality: after the two parties have recovered the whole secret  $s$ , they end up still not knowing the other party's share. It is an open problem whether or not this can be achieved by "purely mathematical" methods (implementable on a Turing machine), but what we offer here is a solution that uses physical properties of the real world.



We therefore position Alice and Bob in a plane, Alice at the point  $A = (x_1, y_1)$  and Bob at the point  $B = (x_2, y_2)$ , and give them both long pieces of rope. We assume that the scaling is such that Alice cannot see Bob's point from where she is and Bob cannot see Alice's point from where he is. Now here is the protocol:

1. Alice fixes one end of her rope at the point  $A = (x_1, y_1)$  and selects a neighborhood  $U$  of the point  $A$  that cannot be seen by Bob. Bob, too, selects a private neighborhood  $V$  of his point  $B = (x_2, y_2)$ .
2. Alice throws (or brings) the other end of her rope to a random point  $C$  in the plane, far enough so that her neighborhood  $U$  could not be seen from  $C$ . Then she positions the part of the rope inside  $U$  so that this part is not a straight line. She then communicates the coordinates of the point  $C$  to Bob.
3. Bob walks to the point  $C$ , ties one end of his rope to Alice's rope, then walks back to his point  $B$ , unwinding his rope along the way. That is, he is not pulling the rope at this step, just unwinding.
4. When Bob reaches his point  $B$ , he starts pulling the rope until Alice tells him to stop, which is as soon as Alice sees that the part of the rope inside her neighborhood  $U$  is a straight line.
5. To make sure that it is not by accident that the part of the rope inside her neighborhood  $U$  is a straight line, Alice asks Bob whether or not the part of the rope inside his neighborhood  $V$  is a straight line. If it is not, then Alice starts pulling her end of the rope toward her point  $A$  until Bob tells her to stop, which is as soon as Bob sees that the part of the rope inside his neighborhood  $V$  is a straight line.
6. When the parts of the rope inside both neighborhoods  $U$  and  $V$  are straight, Alice and Bob assume that their points  $A$  and  $B$  are connected by a straight rope, and they find the slope  $s$  of the corresponding straight line by selecting any two points on the parts of the line inside their private neighborhoods.

Some parts of this protocol may seem redundant, and indeed, if both parties are honest, the protocol can be simplified. However, we want to make sure that if one of the parties is dishonest, he/she cannot cheat to get a hold of the other party's share  $(x_i, y_i)$ . This is why, in particular, Alice tells Bob to stop as soon as she sees that the part of the rope inside her neighborhood  $U$  is a straight line. Otherwise, Bob could triangulate Alice's point  $A$  by straightening the rope between  $A$  and two different points of his choice.

We note that even though neither party in this scenario can determine the other party's point precisely, the slope of the line connecting the two points gives away *some* information, and this has the following interesting application to Yao's "millionaires' problem".

## 2.1 Application to Yao's "Millionaires' Problem"

The "two millionaires problem" introduced in [7] is: Alice has a number  $x_1$  and Bob has a number  $x_2$ , and the goal of the two parties is to solve the inequality  $x_1 < x_2$ ? without revealing the actual values of  $x_1$  or  $x_2$ .

Here is how we can solve this problem by using the solution of the secret sharing problem above. Alice privately selects a number  $y_1 < 0$ , and Bob privately selects a number  $y_2 > 0$ . Now Alice has a point  $(x_1, y_1)$  in the plane, and Bob has a point  $(x_2, y_2)$  in the plane. By using the method in this section, Alice and Bob can determine the slope  $s$  of the line connecting the two points without revealing the actual points. Then  $x_1 < x_2$  if and only if  $s > 0$ .

Note that Yao’s “millionaires’ problem” in this interpretation is a formally weaker problem than secret sharing between two parties because to solve the “millionaires’ problem”, we do not need to know the actual value of the slope  $s$ ; it is sufficient to know just the sign of  $s$ .

### 3 Encryption without One-Way Functions

**Disclaimer.** *We realize that the encryption protocol in this section is rather impractical, but we believe it is helpful in understanding why what we call “nature-based cryptography” provides for what the “usual”, complexity-based, cryptography cannot possibly provide: security against (passive) computationally unbounded adversary. In particular, we believe that “pulling the rope” relays a helpful visual imagery of how the receiver (Bob) participates in the transmission: he is not just “sitting there” waiting for information from Alice to arrive, but participates in transmission actively by “pulling out” a secret from Alice’s private space. This is something that seems to be impossible to mimic in the “usual” public-key scenario, and it is there where physical properties of the real world play a crucial role.*

*For a simple and practical implementation of the ideas of this section, we address the reader to our Section 6, but we do recommend to read this and the following two sections first.*

In this section, we give an encryption protocol that does not use any one-way functions, so that its security is based on physical properties of real-world objects. Particular real-world objects that we use in this section are elastic ropes that quickly regain their original (“natural”) length after being stretched by a force and then released. We also assume that all ropes mentioned in this section obey Hooke’s law when deformed:

$$\Delta l = \frac{F}{E} \cdot l,$$

where  $l$  is the natural length of a rope,  $\Delta l$  is the rope’s extension (strain),  $E$  is the rope’s modulus of elasticity, and  $F$  is the *normal stress*, i.e., the force straining the rope divided by the area of the rope’s cross-section. We note that in physics, a more popular notation for the normal stress is  $\sigma$ , but we believe that in our situation,  $F$  is a more reader-friendly notation, and we shall often refer to  $F$  simply as “force” because the thickness of the rope does not play any role in our considerations.

The scenario is as follows. Alice wants to send a secret positive number  $x_A$  to Bob. We assume that Alice has a private space  $U$  (e.g. a private room) where

nobody (i.e., neither Bob nor an eavesdropper Eve) can observe her actions. Similarly, Bob has a private space  $V$  where nobody can observe his actions. Alice and Bob share a long elastic rope: Alice holds one end of the rope, and Bob holds the other. We assume that Alice also has a collection of ropes of various modula of elasticity *known to everybody* (this is a tribute to Kerckhoffs's principle) that she can combine privately (by cut and paste) to produce two pieces of rope,  $R$  and  $R'$ , of the same private random length but with different private random modula of elasticity.

We assume that everybody (Alice, Bob, Eve) can observe (and measure) everything that is going on in the "public space", i.e., outside the union of  $U$  and  $V$ . For simplicity, we are assuming that both  $U$  and  $V$  are convex domains in the plane.

Now here is the protocol itself:

1. Alice begins by cutting off a piece of the rope of random length inside her private space  $U$ . Bob does the same inside his private space  $V$ . Now nobody knows the total natural length of the rope connecting Alice and Bob.
2. Alice replaces, inside her private space  $U$ , a random part of the rope by her private piece of rope  $R$  with private random modulus of elasticity. The point of doing this is that now the eavesdropper Eve does not know the modulus of elasticity of the whole rope connecting Alice and Bob.
3. Alice interprets her secret number  $x_A$  as the length of the part of the rope which is inside her private space  $U$  in the beginning of the transmission. (We are assuming that a straight rope of length  $x_A$  can fit inside  $U$ ). Specifically, she randomly splits  $x_A$  as a sum of two positive numbers:  $x_A = x'_A + \varepsilon$ , and pins the rope to the floor so that the part of the rope between the rope's end and the pin has length  $\varepsilon$ , and the part of the rope between the pin and the boundary of  $U$  has length  $x'_A$ . She also marks the point  $P_A$  on the rope where the rope intercepts the boundary of  $U$ .  
Denote by  $x_B$  the natural length of the part of the rope inside Bob's private space  $V$ , and by  $L$  the natural length of the part of the rope outside the union of  $U$  and  $V$ . We assume that everybody (Alice, Bob, Eve) knows  $L$ .
4. Now the transmission begins. Bob, who remains inside his private space  $V$ , pulls the rope on his end with a private force  $F_B$  (he may randomly change its magnitude along the way), until the marked point  $P_A$  is at a random point inside his private space  $V$ . (This prevents the eavesdropper Eve from knowing the natural length of the part of the rope that Bob has pulled inside  $V$  since the beginning of the transmission.) It is also important that during this whole procedure, no part of Alice's private piece of rope  $R$  gets outside of  $U$ , to prevent Eve from computing its modulus of elasticity.
5. Alice replaces the piece of rope  $R$  by  $R'$ , to prevent Eve from computing the modulus of elasticity of the rope used for transmission. She then releases her pin and tells Bob to pull the entire rope inside his private space  $V$ .
6. Having done that, Bob measures the natural length of the entire rope, subtracts  $L$  and  $x_B$ , and gets  $x_A$ , which is Alice's secret number.

Security of this protocol is essentially based on the following claim:

**Main Claim.** If the eavesdropper Eve does not know the modulus of elasticity of the rope inside Alice’s private space  $U$ , then she cannot unambiguously determine the natural length of any part of the rope pulled from the inside of  $U$  to the outside.

*Proof.* The validity of this claim follows from Hooke’s law:  $\Delta l = \frac{F}{E} \cdot l$ , where  $l$  is the natural length of a rope,  $F$  is the force straining the rope,  $\Delta l$  is the rope’s extension (strain), and  $E$  is the rope’s modulus of elasticity.

What Eve wants to know here is  $l$ , the natural length of the rope (or part of it) inside  $U$ . She is observing  $\Delta l$ , but  $E$  is private, so even if Eve is able to measure the force  $F$ , she still has two unknowns in this equation, which we can re-write as:  $\frac{F}{\Delta l} = \frac{E}{l}$ . If Eve knows  $F$  and  $\Delta l$ , then she knows the ratio of  $E$  and  $l$ , which still leaves many possibilities for different pairs  $(E, l)$  with the same ratio. Therefore, Eve cannot determine  $l$  unambiguously by observing the “transmission”.  $\square$

In Section 5, we discuss the scenario where the adversary can actively interfere with the protocol.

## 4 Encryption Emulation Attack

Encryption emulation attack is reproducing encryption algorithm with all possible randomness and then comparing the results to the actual encrypted message. In the classical model, encryption is a (possibly non-deterministic) one-to-one function (otherwise, decryption by the receiver would not be unique), and therefore encryption emulation attack is always effective. This is usually the most powerful attack, but the problem is: this attack typically requires prohibitively vast amount of resources from the attacker. If, however, the attacker is computationally unbounded, she can use this attack, hence no encryption scheme in the classical model can be secure against computationally unbounded adversary.

Since in our scheme, we claim security against computationally unbounded adversary, this can only mean one thing: our encryption is not one-to-one, in contrast with the classical model. This yields a question of how can decryption by the receiver in this case be unique. Let us compare our model to the classical one in more detail.

The main difference is that in our model, the receiver (Bob) actively participates in Alice’s transmission, in contrast with the classical model where the receiver is a passive observer of the transmission, just like Eve, and therefore Eve and Bob get exactly the same information during the transmission phase of the protocol. In our scheme, on the other hand, Bob gets more information during the transmission phase, namely he knows where the point  $P_A$  ends up, while Eve does not. In other words, Bob influences the transmission phase with his secret key. It would be interesting to arrange an encryption scheme with similar properties by purely mathematical means, but at the time of this writing we do not know whether or not this is possible (most likely, it is not).

Now let us see why the encryption in our scheme is not one-to-one as far as Eve is concerned. The point is, again, that encryption is a function not only of Alice's secret number  $x_A$  and her other secret parameters  $\varepsilon, E_A$  (where  $E_A$  is her secret modulus of elasticity), but also of Bob's secret parameters that include his force  $F_B$  and the final position  $B(P_A)$  of the point  $P_A$ . To simplify the notation, denote by  $X$  the set of Alice's parameters, and by  $Y$  the set of Bob's parameters. Now the encryption of Alice's secret number  $x_A$  is a function  $F(X, Y)$ . First of all, we have to ask ourselves: what is the co-domain of this function, i.e., where does it take its values? One possible answer to this question is that a value of  $F$  is a function  $S(t)$  that describes the state of the rope at a moment  $t$  after the beginning of the protocol, where  $t$  changes from 0 to the moment when the whole rope is in Bob's private space  $V$ . Denote by  $\hat{S}(t)$  the "retract" of this function, i.e., the state of the rope observable by Eve at a moment  $t$ .

Since we claim security against computationally unbounded adversary, we do not care whether the function  $F$  is one-way or not (in fact, it is not); what we care about is that this function is not one-to-one as far as Eve is concerned, in the sense that if, for some  $\hat{S}(t)$ , there is a pair  $(X_1, Y_1)$  such that  $F(X_1, Y_1) = \hat{S}(t)$ , then there is also at least one pair  $(X_2, Y_2)$  such that the corresponding secret numbers  $x_A^{(1)} \in X_1$  and  $x_A^{(2)} \in X_2$  are different, and yet

$$F(X_1, Y_1) = F(X_2, Y_2).$$

At the same time, the function  $F$  should be one-to-one as far as decryption by Bob is concerned, i.e., for any  $S(t)$ ,  $(X_1, Y)$ ,  $(X_2, Y)$  such that  $F(X_1, Y) = F(X_2, Y)$ , one should have the corresponding secret numbers  $x_A$  equal. This latter condition is satisfied by the very design of the protocol since we have seen in Section 3 that Bob decrypts without any ambiguity.

To show the validity of the displayed condition, we note that if we change  $\varepsilon$  (which is part of  $x_A$ ), the function  $S(t)$  will not change at least until the moment when Alice releases her pin. For  $\hat{S}(t)$  to remain unchanged after Alice releases her pin, the total natural length of the rope visible to Eve after the pin is released should not change. That is, if, for example, we decrease  $\varepsilon$  by some  $\delta$ , there should be a way to increase the length of the rest of the rope visible to Eve by the same  $\delta$ , by varying other parameters. Indeed, recall Hooke's law again:  $E \cdot \Delta l = F \cdot l$ , where  $l$  is the natural length of the part of the rope visible to Eve after the pin is released, without the part of length  $\varepsilon$  (because the latter part is not being strained). Thus, the total natural length of the rope visible to Eve after the pin is released is  $l + \varepsilon$ .

Now suppose  $\varepsilon' = \varepsilon - \delta$ . Then, to keep the total natural length visible to Eve unchanged, the length  $l$  has to be replaced by  $l' = l + \delta$ . Then, in order to keep  $\Delta l$  (visible to Eve during the transmission) unchanged, Alice's private elasticity modulus  $E$  can be changed to  $E'$  so that  $E' \cdot \Delta l = F \cdot l'$ . We can solve this for  $E'$ :  $E' = F \cdot \frac{l'}{\Delta l}$ , thereby showing that different pairs  $(E, x_A)$  can produce the same state of the rope observable by Eve. This is why Eve cannot decrypt unambiguously by using the encryption emulation attack.

## 5 Active Adversary

Active adversary is an adversary who can actively interfere with the protocol, as opposed to just observing and analyzing. We distinguish two kinds of active adversaries:

1. (not-so-dangerous kind) An adversary of this kind would just mess up communication between Alice and Bob without attempting to retrieve Alice's secret message. The simplest way would be just to cut the rope. There is no defense against such malicious behavior (after all, an Internet cable can be cut, too), but there is no threat to security either, so we shall not be concerned about adversaries of this kind.
2. (dangerous kind) An adversary of this kind would attempt to retrieve Alice's secret message. In our situation, an adversary may try to measure the natural length of a piece of the rope emerging from Alice's private room by applying a force compensating Bob's force to return a selected piece of the rope to its natural state. In this case, Bob will detect a counter-force and abort the protocol.

Thus, even though Alice and Bob may not be able to prevent interference in their protocol, they are able to detect malicious activity and abort the protocol to prevent an adversary from recovering secret information. This has a superficial resemblance to the intruder detection ideas in quantum cryptography [1], although our scenario is much closer to everyday life. Also, in the scenario in [1], the intruder may not be detected (with nonzero probability) if she makes correct guesses.

## 6 A More Practical Implementation: Electrical Circuit

In this section, we describe a more practical implementation of the ideas of Section 3. Obviously, using a piece of rope for transmission of data limits applicability of relevant encryption essentially to communication between two offices on the same floor (at best). Therefore, even though the ideas look nice in theory, if they are to be applied to real-life communication, their implementation has to be much more practical.

Here we suggest an encryption protocol similar to that of Section 3, but instead of a piece of rope we use an electrical circuit to transmit data, and the secret information that we transmit is the electric charge of a capacitor.

Thus, the initial setup is as follows. Alice wants to send a secret positive number  $q_A$  to Bob. We assume that Alice has a private space  $U$  (e.g. a private room) where nobody (i.e., neither Bob nor an eavesdropper Eve) can observe her actions. Similarly, Bob has a private space  $V$  where nobody can observe his actions. In her private space  $U$ , Alice has a capacitor  $C_1$  of the capacitance  $c_A$  with the charge  $q_A$ , while Bob in his private space  $V$  has a capacitor  $C_2$  of the capacitance  $c_B$  with the charge  $q_B$ , selected by Bob randomly. These capacitors are connected to form an electrical circuit (see Figure 1), in such a way that the capacitors' plates holding positive charge are connected by one wire, and

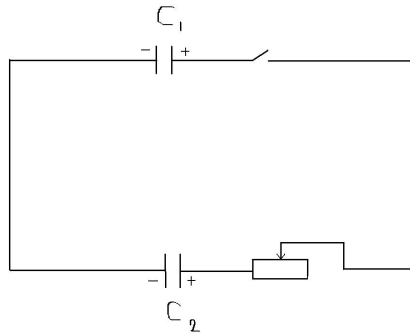
the plates holding negative charge are connected by another wire. Alice also has a switch that keeps the circuit disconnected until the actual transmission begins, and she has an ammeter to monitor the electric current in the circuit. Bob also has (in his private space) a *rheostat* (i.e., a variable resistor) included in the circuit. This allows him to randomly change the resistance of the whole circuit, and therefore also to change parameters of the electric current during the transmission. Now more formally:

**Alice's (sender's) public key:** capacitance  $c_A$

**Alice's secret message:** charge  $q_A$

**Bob's (receiver's) long-term private key:** capacitance  $c_B$

**Bob's session private key:** charge  $q_B$ . This private key is selected by Bob randomly before each transmission from Alice.



**Fig. 1.** Electrical circuit

Now here is the protocol itself:

1. Alice uses her switch to connect the circuit, thereby starting re-distribution of electric charges between the capacitors  $C_1$  and  $C_2$ . When the re-distribution of the charges is complete, Alice's ammeter shows that there is no current in the circuit, so she disconnects the circuit.
2. After re-distribution of the charges is complete, let  $Q_A$  be the new charge of the capacitor  $C_1$ , and  $Q_B$  the new charge of the capacitor  $C_2$ . Then:

$$Q_A = c_A \cdot \frac{q_A + q_B}{c_A + c_B}, \quad Q_B = c_B \cdot \frac{q_A + q_B}{c_A + c_B}, \quad \frac{Q_A}{c_A} = \frac{Q_B}{c_B}.$$

3. Bob, who knows  $Q_B$ ,  $c_B$ ,  $q_B$ , and  $c_A$ , can now recover Alice's secret  $q_A$  from the second formula above:  $q_A = Q_B \cdot \left(1 + \frac{c_A}{c_B}\right) - q_B$ .

Encryption emulation attack on this protocol will not work for the same reason it does not work with the protocol in our Section 3. Namely, in the three equations displayed at Step 2 of our protocol above, there are 5 parameters unknown to the adversary (the only known parameter is  $c_A$ ), which yields many possible

solutions for  $q_A$ . In fact, one of the three equations is redundant (for example, equation 1 follows from the equations 2 and 3), so there are just 2 equations with 5 unknowns to the adversary!

The adversary may attempt to measure the electric current  $I$  in the circuit during the transmission and use other laws of physics to try to recover some of the parameters. Relevant laws of physics include Ohm's law  $I = \frac{U}{R}$ , where  $U$  is the voltage and  $R$  the total resistance in the circuit. Since right after Alice turns her switch on, the initial voltage  $U$  is  $\frac{q_A}{c_A} - \frac{q_B}{c_B}$ , upon combining these two formulas we get  $I = \frac{q_A}{c_A R} - \frac{q_B}{c_B R}$ .

This formula introduces two more parameters, at least one of which, the total resistance  $R$ , cannot be measured by the adversary Eve because some parts of the circuit, including the rheostat with variable resistance, are hidden from Eve.

There are other formulas involving some of the parameters of our electrical circuit, but they all are similar to the formula above in the sense that they introduce new parameters, at least one of which cannot be evaluated by Eve because it is relevant to properties of those circuit elements that are hidden in either Alice's or Bob's private space.

For the same reason, even an active adversary cannot determine  $q_A$  in this situation because even if she measures, say,  $I$ , at whatever moment(s) during the transmission, it will not help her because she does not know how the rheostat's resistance (in Bob's private space) changes. Of course, an active adversary can mess up things in many different ways (the simplest way being just cutting a wire); in particular, she can make sure that Bob will not get the right value of  $q_A$  in the end by connecting her own capacitor to the circuit. Detecting this or other kind of interference with the protocol is one of the issues to be addressed in a real-life implementation of our protocol, but we leave this discussion out of the present paper since it would take us too far into the realm of electrical engineering. What we emphasize here is that even an active adversary cannot get a hold of the actual secret  $q_A$ .

## 6.1 Attempting to Compromise the Receiver's Long-Term Private Key

Suppose Alice transmits several secret messages to Bob. Then, in addition to the three equations (see Step 2 of our protocol), the adversary gets several other triples of equations, with the same  $c_A$  and  $c_B$ , but different other parameters. (As we have seen before, one equation in each triple is redundant, so the adversary actually gets extra *pairs* of equations.) Thus, with each new pair of equations the adversary gets 4 new unknowns ( $q_A$ ,  $q_B$ ,  $Q_A$ ,  $Q_B$ ), which does not help her recover any of the unknowns, including Bob's long-term private key  $c_B$ .

Another attack, called *chosen ciphertext attack*, which is typically considered in cryptography, allows the adversary to encrypt messages of her choice, in an attempt to get information about a long-term private key. What it means for our protocol is that in the system of equations at Step 2 of our protocol there would be 4 unknowns to the adversary rather than 5, but this still is not enough to determine any of the unknowns without ambiguity. Also, if the adversary



encrypts messages of her choice more than once, then each time she will get 2 new equations with 3 new unknowns, which does not help her.

Finally, suppose Alice herself attempts to get Bob's long-term private key. In this case, she knows  $q_A, Q_A, c_A$ , but does not know  $q_B, Q_B, c_B$ , and this still does not allow her to recover Bob's long-term private key  $c_B$  without ambiguity. Encrypting several messages in this scenario will produce more pairs of equations together with new pairs of unknowns  $q_B, Q_B$ , which still does not help her.

## 7 Conclusions

We have employed physical properties of the real world to design public-key encryption protocols secure against computationally unbounded adversary, which is impossible in the "usual" (i.e. complexity-based) cryptography.

What appears to be the main reason why this is possible in nature-based cryptography but impossible in complexity-based cryptography is that physical properties of the real world can be employed to arrange for the receiver (Bob) to be able to influence the transmission of information from the sender (Alice) by using his *private key*, as opposed to the typical scenario in complexity-based cryptography where Bob, after having published his *public key*, is just "sitting there" waiting for information from Alice to arrive.

**Acknowledgements.** Both authors are grateful to the Max Planck Institut für Mathematik, Bonn for its hospitality during their work on this paper. We are also grateful to Igor Monastyrsky for consultations on physical aspects of our schemes, and to Bren Cavallo for pointing out an application of our method to Yao's "millionaires' problem". The first author is also grateful to Labex CEMPI (ANR-11-LABX-0007-01).

## References

1. Bennett, C.H., Brassard, G.: Quantum Cryptography: Public key distribution and coin tossing. In: Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, Bangalore, p. 175 (1984)
2. Garey, M., Johnson, J.: Computers and Intractability, A Guide to NP-Completeness. W. H. Freeman (1979)
3. Grigoriev, D., Shpilrain, V.: Secrecy without one-way functions. Groups, Complexity, Cryptology 5 (2013)
4. Impagliazzo, R., Luby, M.: One-way functions are essential for information based cryptography. In: Proc. 30th IEEE Sympos. on Found. of Comput. Sci., pp. 230–235. IEEE, New York (1989)
5. Kahrobaei, D., Habeeb, M., Shpilrain, V.: A secret sharing scheme based on group presentations and the word problem. Contemp. Math., Amer. Math. Soc. 582, 143–150 (2012)
6. Shamir, A.: How to share a secret. Comm. ACM 22, 612–613 (1979)
7. Yao, A.C.: Protocols for secure computations (Extended Abstract). In: 23rd Annual Symposium on Foundations of Computer Science (Chicago, Ill., 1982), pp. 160–164. IEEE, New York (1982)

# Hypergraph Automata: A Theoretical Model for Patterned Self-assembly

Lila Kari, Steffen Kopecki, and Amirhossein Simjour

Department of Computer Science,  
University of Western Ontario, London, ON, N6A 5B7, Canada  
{lila,steffen,asimjour}@csd.uwo.ca

**Abstract.** Patterned self-assembly is a process whereby coloured tiles self-assemble to build a rectangular coloured pattern. We propose *self-assembly (SA) hypergraph automata* as an automata-theoretic model for patterned self-assembly. We investigate the computational power of SA-hypergraph automata and show that for every recognizable picture language, there exists an SA-hypergraph automaton that accepts this language. Conversely, we prove that for any restricted SA-hypergraph automaton, there exists a Wang Tile System, a model for recognizable picture languages, that accepts the same language. The advantage of SA-hypergraph automata over Wang automata, acceptors for the class of recognizable picture languages, is that they do not rely on an *a priori* defined scanning strategy.

## 1 Introduction

DNA-based self-assembly is an autonomous process whereby a disordered system of DNA sequences forms an organized structure or pattern as a consequence of Watson-Crick complementarity of DNA sequences, without external direction. A DNA-tile-based self-assembly system starts from DNA “tiles”, each of which is formed beforehand from carefully designed single-stranded DNA sequences which bind via Watson-Crick complementarity and ensure the tiles’ shape (square) and structure. In particular, the sides and interior of the square are double-stranded DNA sequence, while the corners have protruding DNA single strands that act as “sticky ends”. Subsequently, the individual tiles are mixed together and interact locally via their sticky-ends to form DNA-based *supertiles* whose structure is dictated by the base-composition of the individual tiles’ sticky ends. Winfree [15] introduced the abstract Tile Assembly Model (aTAM) as a mathematical model for tile-based self-assembly systems. Ma [13] introduced the patterned self-assembly of single patterns, whereby coloured tiles self-assemble to build a particular rectangular *coloured pattern*. Patterned self-assembly models a particular type of application in which tiles may differ from each other by some distinguishable properties, modelled as colours [14,2]. Orponen and Göös [7] and Orponen et al. [10] designed several algorithms to find the minimum tile set required to construct one given coloured pattern. Czeizler and Popa [4] proved that this minimization problem is NP-hard.

In this paper, we propose *self-assembly (SA) hypergraph automata* as a general model for patterned self-assembly and investigate its connections to other models for two-dimensional information and computation, such as 2D (picture) languages and Wang Tile Systems. A 2D (picture) language consists of 2D words (pictures), defined as mappings  $p : [m] \times [n] \rightarrow [k]$  from the points in the two-dimensional space to a finite alphabet of cardinality  $k$ . Here,  $[k]$  denotes the set  $[k] = \{1, 2, \dots, k\}$ . Note that, if we take the alphabet  $[k]$  to be a set of colours, the definition of a picture is analogous to that of a coloured pattern [13].

Early generating/accepting systems for 2D languages comprise  $2 \times 2$  tiles [6], 2D automata [3], two-dimensional on-line tessellation acceptors [8], and 2D grammars. More recently a generating system was introduced by Varricchio [5] that used *Wang tiles*. A *Wang tile system* [5] is a specialized tile-based model that generates the class of *recognizable picture languages*, a subclass of the family of 2D languages. The class of recognizable picture languages is also accepted by *Wang automata*, a model introduced in [11]. Like other automata for 2D languages [1], Wang tile automata use an explicit pre-defined scanning strategy [12] when reading the input picture and the accepted language depends on the scanning strategy that is used. Due to this, Wang automata are a suboptimal model for self-assembly. Indeed, if we consider the final supertile as given, the order in which tiles are read is irrelevant. On the other hand, if we consider the self-assembly process which results in the final supertile, an “order of assembly” cannot be pre-imposed. In contrast to Wang automata, SA-hypergraph automata are scanning-strategy-independent.

SA-hypergraph automata are a modification of the hypergraph automata introduced by Janssens and Rozenberg [9] in 1982. An SA-hypergraph automaton (Section 3) accepts a language of labelled “rectangular grid graphs”, wherein the labels are meant to capture the notion of colours used in patterned self-assembly. An SA-hypergraph automaton consists of an underlying labelled graph (labelled nodes and edges) and a set of *hyperedges*, each of which is a subset of the set of nodes of the underlying graph. Intuitively, the hyperedges are meant to model tiles or supertiles while the underlying graph describes how these can attach to each other, similar to a self-assembly process.

We investigate the computational power of SA-hypergraph automata and prove that for every recognizable picture language  $L$  there is an SA-hypergraph automaton that accepts  $L$  (Thm. 1). Moreover, we prove that for any restricted SA-hypergraph automaton, there exists a Wang tile system that accepts the same language of coloured patterns (Thm. 2). Here, restricted SA-hypergraph automaton means an SA-hypergraph automaton in which certain situations that cannot occur during self-assembly are explicitly excluded.

## 2 Preliminaries

A picture (2D word)  $p$  over the alphabet  $\Sigma$  is a matrix of letters from  $\Sigma$ . Each element of this matrix is called a pixel.  $p_{(i,j)}$  denotes the pixel in the  $i$ th row and  $j$ th column of this matrix. Two pixels  $p_{(i,j)}$  and  $p_{(i',j')}$  are adjacent if  $|i - i'| +$

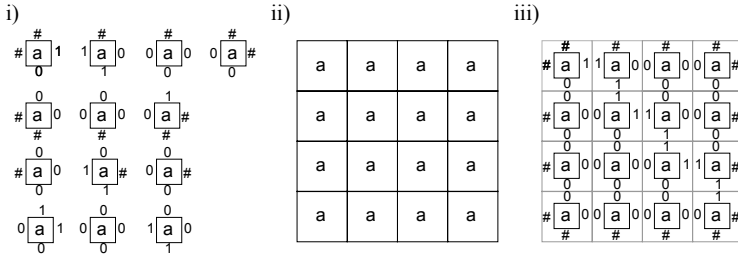
$|j - j'| = 1$ . The function  $w(p)$  denotes the width and  $h(p)$  denotes the height of the picture  $p$ .  $\Sigma^{**}$  is the set of all pictures over the alphabet  $\Sigma$ . A *picture language* (2D language) is a set of pictures over an alphabet  $\Sigma$ . For example,  $L = \{p \in \Sigma^{**} \mid \text{for all } 1 \leq i \leq h(p), p_{(i,1)} = p_{(i,w(p))}\}$  is the language of all rectangles that have the same first and last column.

A *subpicture* over  $\Sigma$  is a matrix of letters from  $\Sigma \cup \{\text{empty}\}$ . A subpicture  $q$  is *connected* if for every pair of pixels  $q_{(i',j')}, q_{(i,j)} \in \Sigma$  there exists a sequence of pixels  $s = \langle s_0, s_1, \dots, s_n \rangle$  from  $q$  such that  $s_0 = q_{(i,j)}$ ,  $s_n = q_{(i',j')}$ , and  $s_k \in \Sigma$  for  $0 \leq k < n$ ; moreover,  $s_k$  and  $s_{k+1}$  must be adjacent. If  $p$  is a picture, then  $q$  is a subpicture of  $p$  if there exists a *translation*  $\delta : \mathbb{N}^2 \rightarrow \mathbb{N}^2$  such that for all  $(i, j) \in [h(q)] \times [w(q)]$  we have either  $q_{(i,j)} = \text{empty}$  or  $q_{(i,j)} = p_{\delta(i,j)}$ . The function  $\delta$  is a translation if  $\delta(i, j) = (i + k, i + l)$  for some  $k, l \in \mathbb{Z}$ .

A definition of *recognizability* was proposed using labelled Wang tiles [12]. A labelled Wang tile, shortly LWT, is a labelled unit square whose edges may be coloured. Formally, a LWT is a 5-tuple  $(c_N, c_E, c_S, c_W, l)$  where  $l$  belongs to a finite set of labels  $\Sigma$  and  $c_N, c_E, c_S,$  and  $c_W$  belong to  $C \cup \{\#\}$  where  $C$  is a finite set of colours and  $\#$  represents an uncoloured edge. Intuitively,  $c_N, c_E, c_S,$  and  $c_W$  represent the colour of the north, east, south, and west edge of the tile, respectively. Labelled Wang tiles cannot rotate. The colours on the north, south, east, and west edges of an LWT  $t$  are denoted by  $\sigma_N(t), \sigma_S(t), \sigma_E(t),$  and  $\sigma_W(t)$ , respectively;  $\lambda(t)$  denotes the label of  $t$ .

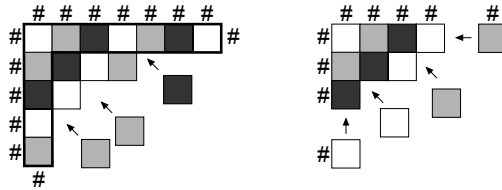
A Wang Tile System (WTS) [5] is a triple  $W = (\Sigma, C, \Theta)$  where  $\Sigma$  and  $C$  are two finite alphabets (the alphabet of tile labels and the alphabet of colours, respectively) with  $\# \notin C$ , and  $\Theta$  is a finite set of labelled Wang tiles with labels from  $\Sigma$  and colours from  $C$ . The WTS  $W$  recognizes the picture language  $\mathcal{L}(W)$  where the picture  $p \in \Sigma^{**}$  belongs to  $\mathcal{L}(W)$  if and only if there exists a mapping  $m : [h(p)] \times [w(p)] \rightarrow \Theta$  from the pixels of  $p$  to tiles from  $\Theta$  such that the label of the tile  $m(i, j)$  is equal to  $p_{(i,j)}$ ; moreover, this mapping must be *mismatch free*. The mapping  $m$  is mismatch free if for two adjacent pixels  $p_{(i,j)}$  and  $p_{(i+1,j)}$  in  $p$  the south edge of  $m(i, j)$  and the north edge of  $m(i + 1, j)$  are coloured by the same colour from  $C$ ; for two adjacent pixels  $p_{(i,j)}$  and  $p_{(i,j+1)}$  in  $p$  the east edge of  $m(i, j)$  and the west edge of  $m(i, j + 1)$  are coloured by the same colour from  $C$ ; and for every border pixel  $p_{(i,j)}$  with  $i = 1, j = 1, i = h(p),$  or  $j = w(p)$  we require that the north, west, south, or east edge, respectively, of  $m(i, j)$  is uncoloured. For a pixel in a corner, e.g.  $p_{(1,1)}$ , this implies that two edges are uncoloured. Let  $\bar{p}$  be a matrix of labelled Wang tiles from  $\Theta$ . We call  $\bar{p}$  a Wang tiled version of the picture  $p$  if the width and the height of  $p$  and  $\bar{p}$  are equal, and there exists a mismatch free mapping  $m$  such that for any  $i$  and  $j$  we have  $\bar{p}_{(i,j)} = m(i, j)$ . Two tiles  $\bar{p}_{(i,j)}$  and  $\bar{p}_{(i',j')}$  are adjacent if the pixels  $p_{(i,j)}$  and  $p_{(i',j')}$  are adjacent. A language  $L$  is *recognizable* if there exists a Wang tile system  $W$  such that  $W$  recognizes  $L$ . Fig. 1 shows an example.

A coloured pattern, as defined in [13], is the end result of a self-assembly process that starts with a fixed-size  $L$ -shaped seed supertile and proceeds as in Fig. 2 i) until one coloured rectangle is formed. Note that Wang Tile Systems can be seen as generators for (potentially infinite) languages of such coloured



**Fig. 1.** Let  $W = (\Sigma, C, \Theta)$  be the Wang Tile System where  $\Sigma = \{a\}$ ,  $C = \{0, 1\}$  and  $\Theta$  consists of the 13 LWTs shown in i). This Wang tile system recognizes the picture language containing all square pictures  $p$  with  $h(p) = w(p) \geq 3$  and where every pixel is labelled by  $a$ . Part ii) is an example picture and iii) shows the Wang tiled version of the picture in part ii).

patterns where the  $L$ -shaped seed-structure of an arbitrary size is generated starting from a single-tiled seed with uncoloured north and west edges and is extended by tiles with uncoloured north or west edges, as shown in Fig. 2 ii).



**Fig. 2.** i) The self-assembly of a single *coloured pattern*, starting with a fixed-size  $L$ -shaped seed. ii) The process of generating a *picture* in the language of a *Wang Tile System*.

### 3 Hypergraph Automata

Let  $f: A \rightarrow B$  be a function and let  $A' \subseteq A$ . The *restriction* of  $f$  to  $A'$  is  $f|_{A'}: A' \rightarrow B$  such that  $f|_{A'}(x) = f(x)$  for all  $x \in A'$ . For any set  $A$  we let  $id: A \rightarrow A$  denote the *identity*.

Let  $\Sigma$  be an alphabet. A *pseudo-picture graph* is a directed labelled graph  $G = (N, E_v \cup E_h, \pi)$  where  $N$  is a finite set of nodes,  $E_v, E_h \subseteq N \times N$  are two sets of edges such that  $E_v \cap E_h = \emptyset$ , and  $\pi: N \rightarrow \Sigma$  is the label function. Edges from  $E_v$  and  $E_h$  will frequently be denoted by  $\xrightarrow{v}$  and  $\xrightarrow{h}$ , respectively. The *node-induced subgraph* of  $G$  by a subset  $N' \subseteq N$  is defined as the graph  $(N', E'_v \cup E'_h, \pi|_{N'})$  where  $E'_v = \{(x, y) \in E_v \mid x, y \in N'\}$  and  $E'_h = \{(x, y) \in E_h \mid x, y \in N'\}$ . A graph  $G'$  is called a *full subgraph* of  $G$  if for some  $N' \subseteq N$  it is the node-induced subgraph of  $G$  by  $N'$ .

A pseudo-picture graph  $G = (N, E_v \cup E_h, \pi)$  is an  $n \times m$ -*picture graph* (for  $n, m \in \mathbb{N}$ ) if there is a bijection  $f_G: N \rightarrow [n] \times [m]$  such that for  $x, y \in N$ ,

we have  $(x, y) \in E_v$  if and only if  $f_G(x) + (1, 0) = f_G(y)$ , and  $(x, y) \in E_h$  if and only if  $f_G(x) + (0, 1) = f_G(y)$ . We want to stress that we do not use Cartesian coordinates: our pictures are defined as matrices; incrementing the first coordinate corresponds to a step downwards, and incrementing the second coordinate corresponds to a step rightwards. In other words, the nodes of a picture graph  $G$  can be embedded in  $\mathbb{N}^2$  such that every edge in  $E_v$  has length 1 and points downwards, every edge in  $E_h$  has length 1 and points rightwards, and two nodes with Euclidean distance 1 are connected by an edge. N.B. if a pseudo-picture graph is an  $n \times m$ -picture graph, it cannot be an  $n' \times m'$ -picture graph with  $n \neq n'$  or  $m \neq m'$ , and the function  $f_G$  is unique. If  $G$  is a picture graph, we call  $e \in E_v$  a *vertical edge* and  $e \in E_h$  a *horizontal edge*. The set of all picture graphs is denoted by  $\mathcal{G}$ . Every  $n \times m$ -picture graph  $G = (N, E_v \cup E_h, \pi)$  represents a picture  $p(G) \in \Sigma^{**}$  with  $h(p(G)) = n$  and  $w(p(G)) = m$ . More precisely, for all  $(i, j) \in [n] \times [m]$  we let  $p(G)_{(i,j)} = \pi(f_G^{-1}(i, j))$ . Hence,  $p: \mathcal{G} \rightarrow \Sigma^{**}$  can be seen as a function. A connected pseudo-picture graph  $G'$  is called a *subgrid* if it is a full subgraph of a picture graph  $G$ . We also say  $G'$  is a subgrid of  $G$ .

A *hypergraph* [9] is a triple  $H = (N, E, f)$  where  $N$  is the finite set of nodes,  $E$  is the finite set of *hyperedges*, and  $f: E \rightarrow \mathcal{P}(N)$  is a function assigning to each hyperedge a set of nodes; the same set of nodes may be assigned to two distinct hyperedges. For every hyperedge  $e \in E$ , we let

$$I_H(e) = \{x \in N \mid \exists e' \in E \setminus \{e\}: x \in f(e) \cap f(e')\}$$

be the set of *intersecting nodes* in  $f(e)$ . Janssens and Rozenberg [9] introduced *hypergraph automata* to describe graph languages. Here, we modified their definition in order to study pseudo-picture graphs.

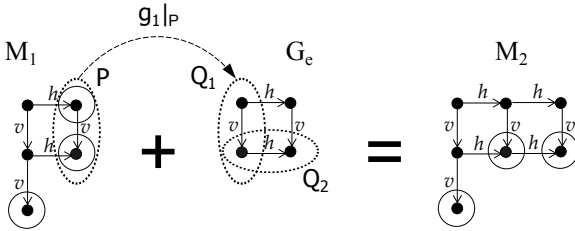
**Definition 1.** A self-assembly (SA) hypergraph automaton is a tuple  $A = (N, E, f, d, G, E_0)$  where  $H = (N, E, f)$  is a hypergraph, called the underlying hypergraph,  $d: E \rightarrow I_H(e) \times I_H(e)$  is the transition function assigning to each hyperedge  $e \in E$  a transition  $Q_1 \rightarrow Q_2$  with  $Q_1, Q_2 \subseteq I_H(e)$ ,  $G$  is a pseudo-picture graph with node set  $N$  called the underlying graph, and  $E_0 \subseteq E$  is the set of initial hyperedges.

Every hyperedge  $e \in E$  defines a graph  $G_e$  which is the subgraph of  $G$  induced by  $f(e)$ . For  $d(e) = Q_1 \rightarrow Q_2$  we call  $Q_1$  and  $Q_2$  the *incoming active nodes* and *outgoing active nodes* of  $G_e$ , respectively. In order for the hypergraph automaton to be well-defined, we require that  $G_e$  is connected and that the subgraph of  $G_e$  induced by its incoming active nodes is connected, too, for all  $e \in E$ . If  $e \in E_0$ , then  $G_e$  is also called an *initial graph*.

A *configuration* of the hypergraph automaton  $A$  is a triple  $(M, O, g)$  where  $M = (N_M, E_{M,v} \cup E_{M,h}, \pi_M)$  is a subgrid,  $O \subseteq N_M$  is the set of *active nodes*, and  $g: N_M \rightarrow N$  is a function such that  $\pi_M(x) = \pi(g(x))$  for all  $x \in N_M$ . The set  $N_M$  consists of (possibly multiple) copies of nodes from  $N$  and the function  $g$  assigns to each node in  $N_M$  its original node in  $N$ . An edge  $(x, y) \in E_{M,h}$  is a copy of the edge  $(g(x), g(y)) \in E_h$  and  $(x, y) \in E_{M,v}$  is a copy of the edge  $(g(x), g(y)) \in E_v$ . However, for two nodes  $x$  and  $y$  in  $M$ , if their originals  $g(x)$

and  $g(y)$  are connected by a horizontal (or vertical) edge, this does not imply that  $x$  and  $y$  are connected by a horizontal (or vertical) edge.

Let  $(M_1, O_1, g_1)$  be a configuration with  $M_1 = (N_1, E_{1,v} \cup E_{1,h}, \pi_1)$  and let  $e \in E$  be a hyperedge with  $d(e) = Q_1 \rightarrow Q_2$ . If there exists a non-empty subset  $P \subseteq O_1$  such that  $g_1|_P$  forms a graph-isomorphism from the subgraph of  $M_1$  induced by  $P$  to the subgraph of  $G_e$  induced by the incoming active nodes  $Q_1$ , then the hyperedge  $e$  defines a *transition* or *derivation step*  $(M_1, O_1, g_1) \xrightarrow[A]{*} (M_2, O_2, g_2)$ . Informally speaking, the resulting graph  $M_2$  consists of joining together the graphs  $M_1$  and  $G_e$  by identifying every node  $x \in P$  with the corresponding node  $g_1(x) \in Q_1$ . The active nodes  $O_2$  in  $M_2$  are the active nodes  $O_1 \setminus P$  in  $M_1$  plus the outgoing active nodes  $Q_2$  in  $G_e$ , see Fig. 3. We also say that  $(M_2, O_2, g_2)$  is the result of *gluing* the hyperedge  $e$  to  $(M_1, O_1, g_1)$ . Formally, the configuration  $(M_2, O_2, g_2)$  where  $M_2 = (N_2, E_{2,v} \cup E_{2,h}, \pi_2)$  is constructed as follows. Let  $N' = \{x' \mid x \in f(e) \setminus Q_1\}$  be a set containing a copy of each node from  $G_e$  except for the incoming active nodes such that  $N' \cap N_1 = \emptyset$ . Let  $N_2 = N_1 \cup N'$  and let  $g_2: N_2 \rightarrow N$  such that  $g_2(x) = g_1(x)$  for  $x \in N_1$  and  $g_2(x') = x$  for  $x' \in N'$ . An edge  $(x, y)$  belongs to  $E_{2,v}$  if  $(x, y) \in E_{1,v}$  or  $x, y \in P \cup N'$  and  $(g_2(x), g_2(y)) \in E_v$ ; an edge  $(x, y)$  belongs to  $E_{2,h}$  if  $(x, y) \in E_{1,h}$  or  $x, y \in P \cup N'$  and  $(g_2(x), g_2(y)) \in E_h$ . Naturally,  $\pi_2(x) = \pi(g_2(x))$  for all  $x \in N_2$  and  $O_2 = (O_1 \setminus P) \cup \{x' \in N' \mid x \in Q_2\}$ . The reflexive and transitive closure of  $\xrightarrow[A]{*}$  is denoted by  $\xrightarrow[A]{*}$  and called a *derivation*.



**Fig. 3.** A transition  $(M_1, O_1, g_1) \xrightarrow[A]{*} (M_2, O_2, g_2)$  joins together the graphs  $M_1$  and  $G_e$  by identifying every node  $x \in P$  with the corresponding node  $g_1(x) \in Q_1$ . The set  $O_2$  of the active nodes of the new configuration  $M_2$  consists of the nodes of the union of the active nodes in  $O_1 \setminus P$  with the outgoing active nodes  $Q_2$  of  $G_e$ . The active nodes of  $M_1$  and  $M_2$  are represented as circled nodes.

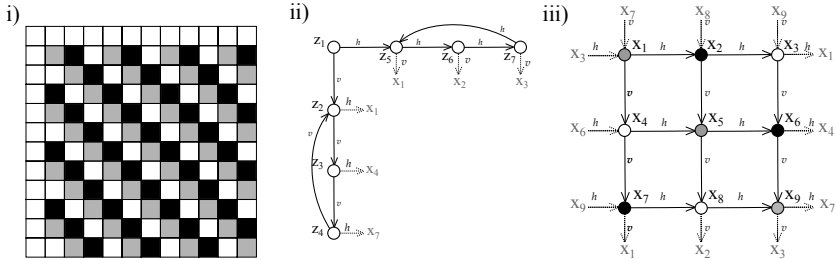
For  $e \in E_0$  we let  $O_e$  such that  $d(e) = Q_1 \rightarrow O_e$  and we call the configuration  $(G_e, O_e, id)$  an *initial configuration* of  $A$ . A *final configuration* is a configuration  $(M, \emptyset, g)$  without active nodes. The graph language accepted by the SA-hypergraph automaton  $A$  is

$$\mathcal{L}(A) = \left\{ M \in \mathcal{G} \mid \exists e \in E_0 : (G_e, O_e, id) \xrightarrow[A]{*} (M, \emptyset, g) \right\}.$$

Note that  $\mathcal{L}(A)$  contains picture graphs only. The *picture language associated to the graph language*  $\mathcal{L}(A)$  is the language  $p(\mathcal{L}(A))$ .

*Remark 1.* Since we only investigate picture graphs, we assume that for every hyperedge  $e \in E$  the underlying graph  $G_e$  is a subgrid.

*Example 1.* Fig. 4 shows an example of an SA-hypergraph automaton  $A$  which is defined as follows. The SA-hypergraph automaton is  $A = (N, E, f, d, G, E_0)$ , where  $N = \{x_1, x_2, \dots, x_9, z_1, z_2, \dots, z_7\}$ ,  $E = \{e_1, e_2, \dots, e_{16}\}$ , and  $E_0 = \{e_{10}\}$ . The function  $f$  is defined such that each hyperedge consists of four nodes which build a  $2 \times 2$ -subgrid of the grid graph in Fig. 4. For example we have,  $f(e_1) = \{x_1, x_2, x_4, x_5\}$ ,  $f(e_2) = \{x_2, x_3, x_5, x_6\}$ ,  $\dots$ ,  $f(e_9) = \{x_9, x_7, x_3, x_1\}$ ,  $f(e_{10}) = \{z_1, z_5, z_2, x_1\}$ , and  $f(e_{11}) = \{z_5, z_6, x_1, x_2\}$ . For each hyperedge in ii), the function  $d$ , which describes the active areas where we can glue new hyperedges, is defined to build a horizontal (vertical) chain of nodes that models the top row (left column) of tiles. For example,  $d(e_{11}) = \{z_5, x_1\} \rightarrow \{z_6, x_1, x_2\}$ . The “backward edges”, e.g.,  $(x_3, x_1)$ ,  $(x_6, x_4)$ ,  $(x_9, x_7)$ , and  $(z_7, z_5)$ , enable the reuse of hyperedges to build a periodic pattern. For each hyperedge in iii), the function  $d$  changes the active input nodes (top-left, bottom-left, and top-right) to the new set of active nodes (top-right, bottom-left, and bottom-right), signifying the change of the places where the new hyperedges can be glued. For example,  $d(e_1) = \{x_1, x_2, x_4\} \rightarrow \{x_2, x_4, x_5\}$ ,  $d(e_2) = \{x_2, x_3, x_5\} \rightarrow \{x_3, x_5, x_6\}$ , and  $d(e_3) = \{x_3, x_1, x_4\} \rightarrow \{x_1, x_6, x_4\}$ .

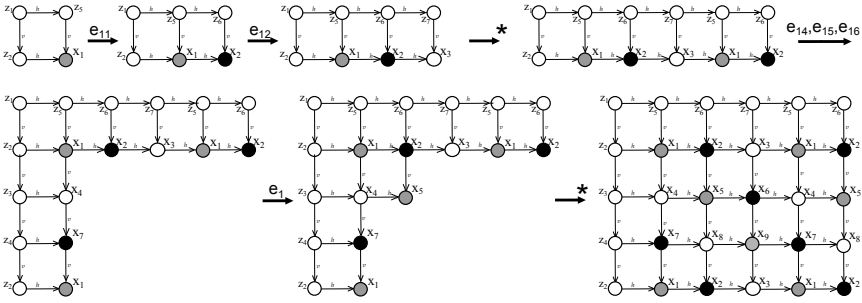


**Fig. 4.** Part i) shows an example of coloured self-assembled pattern. Parts ii) and iii) together depict the underlying graph of the SA-hypergraph automaton that constructs the same pattern. Part ii) constructs the white top row and white left column, and part iii) constructs the coloured pattern.

The SA-hypergraph automaton  $A$  starts from the top-left white tile, corresponding to  $E_0 = \{e_{10}\}$ . Afterwards, the automaton continues the construction with the hyperedges in the top row or the left column. The construction of the white-grey-black part starts after the construction of the white top row and left column. Fig. 5 shows an example of possible transitions of the SA-hypergraph automaton  $A$ .

The concept of hypergraph automata was introduced by Janssens and Rozenberg [9] in 1982. Our definition of SA-hypergraph automata is a variant of the original definition with the following modifications: Firstly, we start from a set of initial graphs whereas the original definition used a single initial graph. For unlabelled





**Fig. 5.** In this example, the construction of a picture graph from Fig. 4 is explained. At each step, one hyperedge or a sequence of hyperedges is glued.

graphs both models are capable of accepting the same class of graph languages, as long as one makes an exception for the empty graph. However, for labelled graphs a single initial graph is not sufficient; e. g., if a language  $L$  of labelled graphs contains one graph  $A$  where every node is labelled by  $a$  and one graph  $B$  where every node is labelled by  $b$ , then  $A$  and  $B$  cannot be generated from the same initial graph as  $A$  and  $B$  do not have a common non-empty isomorphic subgraph. Secondly, we use final configurations in order to accept only some of the graphs that can be generated by rules from the initial graph. In the original definition, for simplicity, final configurations were omitted and every graph which can be generated from the initial graph belonged to the accepted language. Thirdly, it seemed more convenient to us to use the notion of active nodes rather than active intersections.

### 4 Hypergraph Automata for Picture Languages

In this section, we establish a strong connection between recognizable picture languages and picture graph languages that can be accepted by SA-hypergraph automata. We prove that the self-assembly of a Wang Tile System can be simulated by an SA-hypergraph automaton, see Thm. 1. The main idea is to start the tiling in the top left corner of a tiled picture and then extend the tiled picture downwards and rightwards, just as in Fig. 2 ii). Our converse result is slightly weaker: the picture language  $L = p(\mathcal{L}(A))$ , associated to the graph language accepted by an SA-hypergraph automaton  $A$ , is recognizable if  $A$  does not contain a *strong loop*, see Thm. 2. The restriction for  $A$  not to contain a strong loop is a natural assumption as strong loops cannot be used in any derivation that accepts a picture graph.

**Theorem 1.** *For any recognizable picture language  $L$  there is a SA-hypergraph automaton  $A$  such that the picture language associated to the graph language  $\mathcal{L}(A)$  is  $L$ .*

*Proof.* Let  $V = (\Sigma, C', \Theta')$  be a Wang Tile System that recognizes the picture language  $L$ , that is  $L = \mathcal{L}(V)$ . We will slightly modify the WTS  $V$  such that

it fulfils a certain property as described in the following. We define a WTS  $W = (\Sigma, C, \Theta)$  which recognizes  $L$  and such that any two copies of a tile  $t \in \Theta$  in a tiling of  $W$  must have a row- and a column-distance which is a multiple of 3. The modification of  $V$  will become of importance later in the proof: We need to ensure that for a  $2 \times 2$  square of matching tiles  $t_1, t_2, t_3, t_4$ , it is not possible to directly attach another copy of any of  $t_1, t_2, t_3, t_4$  to this square.

We will define a SA-hypergraph automaton  $A = (N, E, f, d, G, E_0)$  which simulates the assembly of a tiled picture from  $L = \mathcal{L}(W)$  as described in Fig. 2 ii). Let  $N$  be a set of nodes such that  $|N| = |\Theta|$  and let  $\vartheta: N \rightarrow \Theta$  be a bijection. For each node  $x \in N$  there is a *corresponding tile*  $\vartheta(x)$  and vice versa. Let  $N_T, N_R, N_B, N_L$  be the set of nodes which correspond to tiles on the top, right, bottom, left border of a tiled picture, respectively:

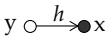

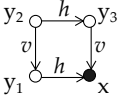
$$\begin{aligned} N_T &= \{x \in N \mid \sigma_N(\vartheta(x)) = \#\}, & N_R &= \{x \in N \mid \sigma_E(\vartheta(x)) = \#\}, \\ N_B &= \{x \in N \mid \sigma_S(\vartheta(x)) = \#\}, & N_L &= \{x \in N \mid \sigma_W(\vartheta(x)) = \#\}. \end{aligned}$$

Let  $G = (N, E_v \cup E_h, \pi)$  be the underlying graph of  $A$ . The label function  $\pi$  is naturally defined as  $\pi(x) = \lambda(\vartheta(x))$  for  $x \in N$ . For all nodes  $x, y \in N$  there is an edge  $(x, y) \in E_h$  if and only if  $\sigma_E(\vartheta(x)) = \sigma_W(\vartheta(y)) \neq \#$  and either  $x, y \in N \setminus (N_T \cup N_B)$  or  $x, y \in N_T$  or  $x, y \in N_B$ ; there is an edge  $(x, y) \in E_v$  if and only if  $\sigma_S(\vartheta(x)) = \sigma_N(\vartheta(y)) \neq \#$  and either  $x, y \in N \setminus (N_L \cup N_R)$  or  $x, y \in N_L$  or  $x, y \in N_R$ . This means if the east edge of a tile  $t$  can attach to the west edge of tile  $s$ , then their corresponding nodes  $x = \vartheta^{-1}(t)$  and  $y = \vartheta^{-1}(s)$  are connected by an  $h$ -edge  $(x, y) \in E_h$ . Analogously, if the south edge of a tile  $t$  can attach to the north edge of tile  $s$ , then their corresponding nodes  $x = \vartheta^{-1}(t)$  and  $y = \vartheta^{-1}(s)$  are connected by an  $v$ -edge  $(x, y) \in E_v$ .

If  $N_T \cap N_B \neq \emptyset$  or  $N_R \cap N_L \neq \emptyset$ , the language  $\mathcal{L}(W)$  possibly contains pictures  $p$  with  $h(p) = 1$  or  $w(p) = 1$ , respectively, which can be seen as one-dimensional pictures. These pictures have to be treated separately. For now we assume that  $N_T \cap N_B = N_R \cap N_L = \emptyset$ .

The hyperedges  $E$  and the transition function  $d$  define the possible transitions of  $A$ . In every transition we add exactly one node to the graph of a configuration of  $A$ . Our naming convention is that  $x$  is the node which is attached in the derivation step and  $y, y_1, y_2, y_3$  are incoming active nodes of the hyperedge. Every graph containing only one node which corresponds to a tile in the top left corner is an initial graph. In order to construct a picture graph which represents a picture in  $\mathcal{L}(W)$  we introduce three types of transitions, see Fig. 6. The transitions of type I generate the top row of the graph and transitions of type II generate the left column of the graph; both transition types keep every generated node active. Transitions of type III generate the rest of the graph: A node is attached if it has a matching east neighbour ( $y_1$ ), a matching north neighbour ( $y_3$ ), and these two nodes are connected by another node ( $y_2$ ); unless we reach the right or bottom border of the graph the nodes  $x, y_1$ , and  $y_3$  are active after using the transition.  $\square$

Next, we prove that a picture language  $L = p(\mathcal{L}(A))$ , associated to the graph language  $\mathcal{L}(A)$ , is recognizable if  $A$  does not contain a strong loop. Let  $A$  be an

Type	I	II	III
Hyperedges			

**Fig. 6.** The hyperedges in the SA-hypergraph automaton  $A$  induce three different types of graphs. White nodes represent incoming active nodes of the hyperedges.

SA-hypergraph automaton. A series of hyperedges  $s = \langle e_0, e_1, \dots, e_n \rangle$  from  $A$  is a (*derivation*) *loop* if  $e_0 = e_n$  and  $Q_{2,i} \cap Q_{1,i+1} \neq \emptyset$  where  $d(e_i) = Q_{1,i} \rightarrow Q_{2,i}$  for  $0 \leq i < n$ . Loops in an SA-hypergraph automaton are a prerequisite for using a hyperedge several times in one derivation. Therefore, an SA-hypergraph automaton without any loops can only accept a finite graph language. Let  $G_i = G_{e_i}$  be the graph induced by  $e_i$ , let  $x$  be a node in  $G_0 = G_n$ , and let  $O_i = Q_{2,i} \cap Q_{1,i+1}$  be set overlapping incoming/outgoing active nodes of  $G_i$  and  $G_{i+1}$ . There is a path in the underlying graph of  $A$  from  $x$  to  $x$  which only visits the subgraphs  $G_0, \dots, G_n$ , in the given order, and passes through at least one node of each  $O_i$  (the path may use incoming and outgoing edges). The loop  $s$  is a *strong loop* if, on this path, the number of incoming horizontal edges equals the number of outgoing horizontal edges and the number of incoming vertical edges equals the number of outgoing vertical edges. In other words, when starting from a configuration  $M$  and successively gluing the hyperedges from  $s$  to  $M$ , then the subgraph added by the hyperedge  $e_0$  and the subgraph added by the hyperedge  $e_n$  fully overlap when naturally embedded in  $\mathbb{Z}^2$ . Note that, by Remark 1, all graphs  $G_i$  are subgrids which implies that the choice of the path from  $x$  to  $x$  does not matter in this definition.

**Theorem 2.** *Let  $A$  be a SA-hypergraph automaton without any strong loops. The picture language  $L = p(\mathcal{L}(A))$ , associated to the graph language  $\mathcal{L}(A)$ , is recognizable (by a Wang Tile System).*

*Proof.* Let  $A = (N, E, f, d, G, E_0)$  and let  $G = (N, E_v \cup E_h, \pi)$ . We may assume that  $e \in E_0$  if and only if  $d(e) = \emptyset \rightarrow O_e$ . Therefore, none of the initial hyperedges can be used in a transition. This assumption is justified by the fact that we can duplicate all hyperedges in  $E_0$  such that one copy can be used in a transition but does not belong to  $E_0$  and the other copy which belongs to  $E_0$  cannot be used in a transition. Furthermore, any hyperedge without incoming active nodes which does not belong to  $E_0$  is useless and can be removed from  $E$ .

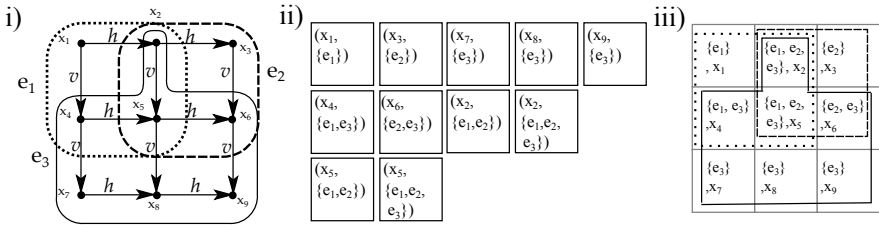
For a node  $x \in N$  we define the list of *related hyperedges* to  $x$ ,  $H_x = \{e \in E \mid x \in f(e)\}$ . Let  $x$  be a node and  $\psi \subseteq H_x$ . We call a hyperedge  $g \in \psi$  a *generator* of  $(x, \psi)$  if  $x \notin Q_1$  with  $d(g) = Q_1 \rightarrow Q_2$ . Note that if  $g \in E_0$ , then  $g$  must be a generator. We call a hyperedge  $c \in \psi$  a *consumer* of  $(x, \psi)$  if  $x \notin Q_2$  with  $d(c) = Q_1 \rightarrow Q_2$ . The pair  $(x, \psi)$  is a *tile candidate* if  $\psi$  contains exactly one generator  $g_{(x,\psi)}$  and exactly one consumer  $c_{(x,\psi)}$ ; furthermore, if  $g_{(x,\psi)} = c_{(x,\psi)}$ , we require that  $\psi = \{g_{(x,\psi)}\}$ . Note that if  $g_{(x,\psi)} \neq c_{(x,\psi)}$ , then for all  $e \in \psi$  with

$d(e) = Q_1 \rightarrow Q_2$ , we have that  $x \in Q_1$  unless  $e$  is the generator and  $x \in Q_2$  unless  $e$  is the consumer. The tile candidate  $(x, \psi)$  describes the attachment of a copy of the node  $x$  to the output graph by the generator; afterwards,  $x$  is used as active node by all hyperedges in  $\psi \setminus \{g(x, \psi), c(x, \psi)\}$ ; finally,  $x$  is deactivated by the consumer. Let  $G_\psi$  be the node-induced subgraph of  $G$  by  $\bigcup_{e \in \psi} f(e)$ . If  $G_\psi$  is not a subgrid (a subgraph of some picture graph), we remove  $(x, \psi)$  from the set of tile candidates. Let  $\Psi$  denote the set of all remaining tile candidates.

The Wang tile system  $W = (\Sigma, C, \Theta)$  which recognizes  $L$  is constructed based on the list  $\Psi$ . In order to recognize the picture language associated to  $\mathcal{L}(A)$ , we have to define the attachments of tile candidates. We use unordered pairs  $\{(x, \psi), (y, \varphi)\} \in \Psi^2$  of tile candidates for the colours on the edges. For a tile candidate  $(x, \psi) \in \Psi$  we define the set of labelled Wang tiles

$$\Theta_{(x, \psi)} = \mathcal{S}_{N,(x, \psi)} \times \mathcal{S}_{E,(x, \psi)} \times \mathcal{S}_{S,(x, \psi)} \times \mathcal{S}_{W,(x, \psi)} \times \{l_x\}$$

where  $l_x$  is the label  $\pi(x)$  and  $\mathcal{S}_{N,(x, \psi)}$ ,  $\mathcal{S}_{E,(x, \psi)}$ ,  $\mathcal{S}_{S,(x, \psi)}$ ,  $\mathcal{S}_{W,(x, \psi)}$  are sets of colours which are defined below. The tile set is the union  $\Theta = \bigcup_{(x, \psi) \in \Psi} \Theta_{(x, \psi)}$ . Fig. 7 shows an example of this construction.



**Fig. 7.** Let  $A = (N, E, f, d, G, E_0)$  be a SA-hypergraph automaton where  $N, E, f$ , and  $G$  are defined in part i). Function  $d$  is defined such that  $d(e_1) = \{x_1\} \rightarrow \{x_2, x_4, x_5\}$ ,  $d(e_2) = \{x_2, x_5\} \rightarrow \{x_2, x_5, x_6\}$  and  $d(e_3) = \{x_2, x_4, x_5, x_6\} \rightarrow \{\}$ . SA-hypergraph automaton starts from  $e_1$ . Part ii) shows the set of all the possible tile candidates. On each tile related node and the set of  $\psi$  are written. The tiling on part iii) is the result of overlapping of three hyperedges  $e_1, e_2$ , and  $e_3$ .

For  $(x, \psi), (y, \varphi) \in \Psi$ , we let  $\{(x, \psi), (y, \varphi)\} \in \mathcal{S}_{E,(x, \psi)}$  and  $\{(x, \psi), (y, \varphi)\} \in \mathcal{S}_{W,(y, \varphi)}$  if and only if 1.)  $(x, y) \in E_h$ ; 2.)  $H_x \cap \varphi \subseteq \psi$ ; 3.)  $\psi \cap H_y \subseteq \varphi$ ; and 4.)  $g(x, \psi) = g(y, \varphi)$  or  $y \in Q_1$  for  $d(g(x, \psi)) = Q_1 \rightarrow Q_2$  or  $x \in Q'_1$  for  $d(g(y, \varphi)) = Q'_1 \rightarrow Q_2$ . For  $(x, \psi), (y, \varphi) \in \Psi$ , we let  $\{(x, \psi), (y, \varphi)\} \in \mathcal{S}_{S,(x, \psi)}$  and  $\{(x, \psi), (y, \varphi)\} \in \mathcal{S}_{N,(y, \varphi)}$  if and only if  $(x, y) \in E_v$  and conditions 2 to 4 are satisfied. For  $(x, \psi) \in \Psi$ , we let  $\mathcal{S}_{E,(x, \psi)} = \{\#\}$  if  $x$  does not have an incoming vertical edges in the graph  $G_\psi$ . By symmetric condition we let  $\mathcal{S}_{N,(x, \psi)} = \{\#\}$ ,  $\mathcal{S}_{S,(x, \psi)} = \{\#\}$ , or  $\mathcal{S}_{W,(x, \psi)} = \{\#\}$ .

Every picture  $p \in \mathcal{L}(W)$ , generated by the suggested tiling system, is in  $p(\mathcal{L}(A))$  and vice versa.  $\square$

## 5 Conclusion

We introduced SA hypergraph automata, a language/automata theoretic model for patterned self-assembly systems. SA hypergraph automata accept all recognizable picture languages but, unlike other models, (e.g., Wang Tile Automata) SA-hypergraph automata do not rely on an *a priori* given scanning strategy of a picture. This property makes the SA hypergraph automata better suited to model DNA-tile-based self-assembly systems.

SA-hypergraph automata provide a natural automata-theoretic model for patterned self-assemblies that will enable us to analyse self-assembly in an automata-theoretic framework. This framework lends itself easily to, e.g., descriptive and computational complexity analysis, and such studies may ultimately lead to classifications and hierarchies of patterned self-assembly systems based on the properties of their corresponding SA-hypergraph automata. An additional feature is that each SA-hypergraph automaton accepts an entire class of “supertiles” as opposed to a singleton set, which may also be of interest for some applications or analyses.

**Acknowledgements.** We thank Professor Grzegorz Rozenberg for extended discussions and his suggestion of applying hypergraph automata to the DNA self-assembly setting.

## References

1. Anselmo, M., Giammarresi, D., Madonia, M.: Tiling automaton: A computational model for recognizable two-dimensional languages. In: Holub, J., Žďárek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 290–302. Springer, Heidelberg (2007)
2. Barish, R.D., Schulman, R., Rothmund, P.W.K., Winfree, E.: An information-bearing seed for nucleating algorithmic self-assembly. Proceedings of the National Academy of Sciences (2009)
3. Blum, M., Hewitt, C.: Automata on a 2-dimensional tape. In: SWAT (FOCS), pp. 155–160 (1967)
4. Czeizler, E., Popa, A.: Synthesizing minimal tile sets for complex patterns in the framework of patterned DNA self-assembly. In: Stefanovic, D., Turberfield, A. (eds.) DNA 2012. LNCS, vol. 7433, pp. 58–72. Springer, Heidelberg (2012)
5. de Prophetis, L., Varricchio, S.: Recognizability of rectangular pictures by Wang systems. Journal of Automata, Languages and Combinatorics 2(4), 269 (1997)
6. Giammarresi, D., Restivo, A.: Two-dimensional languages, pp. 215–267. Springer (1997)
7. Göös, M., Orponen, P.: Synthesizing minimal tile sets for patterned DNA self-assembly. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16. LNCS, vol. 6518, pp. 71–82. Springer, Heidelberg (2011)
8. Inoue, K., Nakamura, A.: Some properties of two-dimensional on-line tessellation acceptors. Inf. Sci. 13(2), 95–121 (1977)
9. Janssens, D., Rozenberg, G.: Hypergraph systems generating graph languages. In: Ehrig, H., Nagl, M., Rozenberg, G. (eds.) Graph Grammars 1982. LNCS, vol. 153, pp. 172–185. Springer, Heidelberg (1983)

10. Lempiäinen, T., Czeizler, E., Orponen, P.: Synthesizing small and reliable tile sets for patterned DNA self-assembly. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 145–159. Springer, Heidelberg (2011)
11. Lonati, V., Pradella, M.: Picture recognizability with automata based on Wang tiles. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 576–587. Springer, Heidelberg (2010)
12. Lonati, V., Pradella, M.: Strategies to scan pictures with automata based on Wang tiles. *RAIRO - Theor. Inf. and Applic.* 45(1), 163–180 (2011)
13. Ma, X., Lombardi, F.: Synthesis of tile sets for DNA self-assembly. *IEEE Trans. on CAD of Integrated Circuits and Systems* 27(5), 963–967 (2008)
14. Rothmund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology* (2004)
15. Winfree, E.: Algorithmic self-assembly of DNA. PhD thesis (1998)

# Modeling Heart Pacemaker Tissue by a Network of Stochastic Oscillatory Cellular Automata

Danuta Makowiec

Institute of Theoretical Physics and Astrophysics, University of Gdańsk, Poland

fizdm@univ.gda.pl

<http://www.strony.fizdm.ug.edu.pl/ME/>

**Abstract.** Computations performed by a system of cyclic cellular automata, designed to model the principal organization rules known for the tissue of the first cardiac pacemaker — the sinus node, are investigated in terms of Kuramoto order parameters of synchronization. We show that such description provides consistent quantification of stationary states in the model. Finally, the model is used to give possible explanations for changes observed in the sinus node rhythmicity caused by age.

**Keywords:** cellular automata, synchronization, multiscale modeling.

## 1 Introduction

Discoveries appearing day by day from biochemical and physiological labs are challenging for computational methods because they need novel conceptions for computation. The natural computations are different from methods to which we got used to in many aspects. It is even said that specifying the problem is demanding [1].

The natural computations are parallel. The natural systems are space distributed, constituted from many different subsystems. The computations are run in parallel in all parts of these systems. The natural computations are dynamical. The present state of each system unit: organizing (as a protein or cell, for example) or functional unit (like a tissue or organ), are recorded in its environment by initiating a sequence, often a cascade, of biochemical processes. Effects of these processes are read back by the same unit to modify, adapt its next state. Since the effects of states are emitted to the environment, the results are dynamically used by other neighboring units. The natural computations are multi-layered. The local environment of any unit consists of elements of other computational systems. They can be different from each other because of, for example, the way of communication. One system passes the information via diffusion of transmitters (neuronal systems) while the other by cell-to-cell direct injection of ions (cardiac cells). So they act at different time and space scales. But these different computational systems penetrate tightly one another providing the robust solution.

In the following we will observe computations performed by a system made of cyclic cellular automata, designed to model the principal organization known for

the tissue of the sinus node — the first cardiac pacemaker [2–8]. By computations in this system we understand emerging of the collective state — synchronization of oscillations of individual cells to produce the strong pacemaker signal. To achieve such computational system, the network of coupled discrete-state units is proposed. The elementary units — the basic layer of computations, reveal the electrochemical properties of pacemaker cells. The network of intercell connections establishes the next layer. The sparsity and stochastic heterogeneity, known organizing principles of the pacemaker tissue structure, are taken into account. The adaptive and/or controlling layer is introduced in the so-called tonic way. The regulatory role of the autonomic nervous system is revealed by parameters describing sensitivity of cells for external stimulation.

The model has its roots in Greenberg-Hastings cellular automata modeling the excitable medium [9, 10]. Discrete modeling has become attractive because it provides the opportunity to adjust both assumptions and results in models with natural cellular automata where the natural computations are performed. By natural cellular automata we mean the technics of in vitro models where cultured cardiac cells are placed on special matrices in order to observe emergence and development of cell-to-cell interactions [11, 12].

Therefore, the basic objective of our modeling is to provide a tool which is enable to test intuitions about general mechanisms responsible for the multiscale effects. The work is a continuation of our investigations [13, 14]. In the following the model is rewritten in a compact way, Sec. 2. Then results are presented which were obtained in simulations aimed on effects of heterogeneity among intercellular connections on collective properties of stationary states, Sec.3. The results are expressed in terms of Kuramoto order parameters — the popular tools for quantifying the synchrony among coupled oscillators [19, 20]. We show that such description consistently quantifies collective properties in the studied systems. In last Section we discuss the model properties in order to explain some changes in the sinus node with age.

## 2 From Natural Computation to Computer Computation

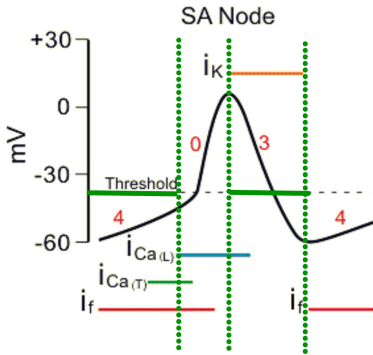
The heart automaticity is originated in the sinoatrial node — a flat tissue located on the right atrium. The node produces sustained oscillations of electrochemical signals. These signals initiate the sequence of events which eventually lead to the whole heart contraction. It is commonly believed that the source of sinoatrial node function is closely related to electrophysiology of each individual cell.

### 2.1 The Model of a Cell

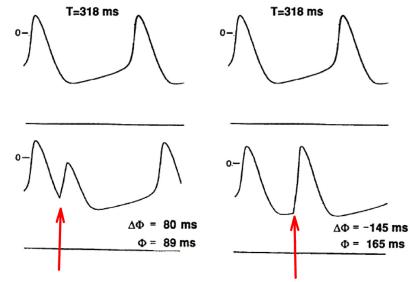
**Physiology of a Pacemaker Cell.** There is a sequence of biochemical processes which change the electrical potential of a myocyte — the cardiac cell, membrane. The phenomenon of ions transmission through the membrane the cell: inward (potassium  $K^+$ ) and outward (sodium  $Na^+$  and calcium  $Ca^{2+}$ ), is called the action potential. The time interval when the myocyte membrane is



depolarized takes about 400ms what is the half of the time interval between subsequent heart contractions, approximately. In the case of pacemaker cells, the course of the action potential is substantially different from other myocardium cells. The depolarization process is slow, see Fig.1, because the increase in the membrane potential is not caused by sodium  $\text{Na}^+$  ions but by calcium  $\text{Ca}^{2+}$  ions only. Furthermore, after completing the action potential, the membrane potential does not stay resting but rises continuously due to activity of  $i_f$  current and calcium current  $i_{\text{Ca}(T)}$ , and reaches the threshold value. Then the next action potential develops due to  $i_{\text{Ca}(L)}$  [2, 3]. The dotted lines in Fig.1 divide the oscillation interval into three physiologically justified stages: 0, 3 and 4, called in the following: *firing*, *refractory* and *activity*, respectively.



**Fig. 1.** Membrane potential of a nodal cell in time together with basic currents:  $i_f$ ,  $i_{\text{Ca}(T)}$ ,  $i_{\text{Ca}(L)}$ ,  $i_K$  that drive the membrane potential change. Diagram is a modified scheme from [3].



**Fig. 2.** The effects of depolarizing of pulses (denoted as red arrows) on the spontaneous cycle of a single enzymatically dissociated pacemaker cell from the sinoatrial node of the rabbit. Diagram is a modified scheme from [16, 2]

The, so-called, phase sensitivity [15, 16, 2] describes an important aspect of the pacemaker cell electrophysiology, see Fig.2. Experiments with rabbit and other mammalian heart cells showed that an external stimulus can influence the intrinsic cycle. It could shorten the cycle if it arrives when a cell is after completing of the action potential, so after stage 3. The perturbation could elongate the period if it comes when the action potential process has not been completed, so during stage 3. In Fig.2 upper plots describe unperturbed membrane potential with the mean period ( $T$ ) = 318 msec. Lower plots show the membrane potential when the pulse appears before the end of the repolarization process, and when the pulse comes later.

**Assumptions.** We propose to consider an automaton, called *FRA-cell*, to model the sequence of nodal cell cycle. The model is based on the following assumptions:

- (i) The cell cycle consists of three successive stages along which the cell progresses: *firing*, *refractory* and *activity*. Upon completion of *activity* stage, the cell immediately enters a new cycle in *firing* stage.
- (ii) Each stage is characterized by its maximal duration which is represented by consecutive stage states. The number of states for each stage is fixed. As soon as the duration of a given stage is reached, the transition to the first state of next stage of the cell cycle occurs.
- (iii) The time at which the transition takes place can vary in a random manner.
- (iv) If a cell is in *activity* stage and receives a stimulus then it switches to *firing* stage.
- (v) If a cell is in *refractory* stage and receives a stimulus then it stays longer in this stage.

### The *FRA*-Cell Model

- (a) Let  $\Sigma = \{F_i, R_j, A_k\}$  be the state space of a *FRA*-cell,  $i, j, k \in \{1, 2, \dots, n_\sigma\}$ , with  $n_\sigma \in \{f, r, a\}$  denoting the duration of stages *firing*, *refractory*, and *activity*, respectively.  
Let  $\phi(t) = \sigma_l$  is the phase of a *FRA* cell at time  $t$ ,  $\sigma_l \in \Sigma$ .
- (b) The cellular phase in the next time step  $\phi(t+1)$  is

$$\begin{aligned} \phi(t+1) &= \text{next}(\sigma)_1, & \text{with probability } & \left(\frac{l}{n_\sigma}\right)^\xi \text{ with } l \leq n_\sigma, \\ \phi(t+1) &= \sigma_{l+1}, & \text{otherwise} & \end{aligned} \quad (1)$$

where  $\text{next}(\textit{firing})=\textit{refractory}$ ,  $\text{next}(\textit{refractory})=\textit{activity}$ ,  $\text{next}(\textit{activity})=\textit{firing}$  forces the transition to the subsequent cellular stage, and  $\xi > 1$ .

- (c) If a *FRA* cell receives a stimulus then

$$\begin{aligned} \phi(t+1) &= F_1, & \text{if } \phi(t) &= A_l \text{ with } l = 1, \dots, a, & (2) \\ \phi(t+1) &= R_{\max\{1, g(l)\}}, & \text{if } \phi(t) &= R_l \text{ with } l = 1, \dots, r, & (3) \\ & \text{where } g(l) &= \lfloor l \setminus 2 \rfloor \end{aligned}$$

The rule (1) is probabilistic. It allows to shorten duration of each stage. But when  $\xi$  is large enough then the dynamics becomes deterministic. The rules (1) and (2) together are adaptation of rules used in models of excitable medium [9, 11, 10]. The rule (3) was considered in simple models of two interacting cells only, and, up to our knowledge, it has not been investigated in the network systems.

## 2.2 The Tissue Model

**Physiology of Pacemaker Tissue.** The sinus node tissue is flat without any fixed structure. Its contents is usually described as heterogeneous populations of small myocytes [4–8]. The nodal cells form clusters and bundles surrounded by abundant collagen. The node border is a relatively discrete boundary seen between the margins of the node and the adjacent atrial tissues. The transmission of action potentials from cell to cell occurs via large-conductance ion channels, closely packed in large arrays named gap junctions.

**Assumptions.** Following [11] we assume that the pacemaker tissue is well approximated by a square lattice with open boundary conditions. Each vertex is occupied by a *FRA*-cell, but only some of lattice neighboring cellular connections are established for inter-cellular interactions. Additionally, we inject heterogeneity to the network by local and intentional wrinkling of inter-cellular connections. The procedure of wrinkling is based on Watts-Strogatz [17] rewiring rule. The intentionality of wrinkling is realized by preferential unlinking from the rare connected neighbors. The locality means that only cells from the close neighborhood can be linked instead. The network of interactions does not evolve. Details of the wrinkling algorithm are given in [18]. Moreover, we assume that cells in *firing* stage are the only source of stimuli to nearest neighbors.

### The Model of Network Interactions

- (A) A *FRA*-network of density  $d$  consists of  $N = L \times L$  *FRA*-cells located in vertices of square lattice, where any two cells of Moore neighborhood are connected to interact with probability  $d$ . The boundary conditions of a lattice are open.
- (B) Let  $a$  be a *FRA*-cell and  $\mathcal{N}(a)$  be the set of cells interacting with cell  $a$ . Let  $b \in \mathcal{N}(a)$ . For a given  $p \in [0, 1]$ , probability  $p_{break}$  to unlink cell  $b$  from the cell  $a$  is as follows

$$p_{break} = \frac{p}{\deg(b)}$$

where  $\deg(b) = \text{card } \mathcal{N}(b)$  is the vertex degree of cell  $b$ . A randomly chosen cell  $b' \in \mathcal{N}(b)$  is linked to cell  $a$  in place of cell  $b$ . So, finally  $b' \in \mathcal{N}(a)$ .

The procedure is repeated  $J$  Monte Carlo time steps.

Unlinking from a leaf is forbidden. In total, the probability for rewiring of each link is  $p * J$ . Let us recall that Moore neighborhood on a square lattice comprises the eight cells surrounding the central cell.

## 2.3 The Control System

**The Physiology of the Cardiac Heart Contraction Control.** Depolarization of the sinus node cells results in the heart contractions about 100 times per minute. This high rate is constantly modified by the activity of sympathetic and parasympathetic nerve fibers. Parasympathetic activation basically decreases the pacemaker rate by decreasing  $i^f$  current. The activation of the sympathetic part of the autonomic regulation acts contrary.

**Assumptions.** We assume that autonomic regulation acts tonically what reveals in the sensitivity of a *FRA*-cell to interact with its neighbors.

### The Model

Let  $N(a, t) = \text{card } \{a' \in \mathcal{N}(a) | \phi_{a'}(t) = F_i, i = 1, \dots, f\}$  be number of cells interacting with  $a$  which at  $t$  are in *firing* stage.

Then for a given  $F = 0, 1, \dots$  and  $R = 0, 1, \dots$

If  $N(a, t) > F$  then rule (2) applies.

If  $N(a, t) > R$  then rule (3) applies.

So,  $F$  is the sensitivity threshold for interaction of cell  $a$  in *activity* stage, and  $R$  is the sensitivity threshold for interaction of cell  $a$  in *refractory* stage.

### 3 Results

#### 3.1 Kuramoto Order Parameters to Quantify Collective States

If there is some variation among  $N$  interacting oscillators, namely, when an oscillator is isolated then oscillates with own intrinsic frequency, then macroscopic synchronization means that the quantity:

$$\mathcal{K}_f = \frac{M}{N}, \quad (4)$$

has non-zero value. Here  $M$  is the size of the largest group of oscillators that attain the same mean frequency. This parameter of synchrony perfectly fits to neuronal interactions. However, in the case of short-range and pulse-like interactions, the more accurate measure of the level of synchrony in a collections of  $N$  phase oscillators  $\phi_l(t)$  is given by the following quantity:

$$\mathcal{K}_\phi = \frac{1}{N} \left| \sum_{l=1}^N e^{i\phi_l} \right|, \quad (5)$$

If all oscillators have the same phase then  $\mathcal{K}_\phi = 1$ . If phases are scattered at random then  $\mathcal{K}_\phi$  is close to zero. Hence, this order parameter quantifies the degree of phase synchronization, whereas  $\mathcal{K}_f$  measures the degree of frequency synchronization. A non-zero  $\mathcal{K}_\phi$  implies a non-zero  $\mathcal{K}_f$ , but the opposite is not true.

It turns out [13] that if *FRA*-network is homogeneous (no wrinkling) and *FRA*-cell dynamics is deterministic ( $\xi \gg 1$ ) then system, evolving under rules (1) and (2) only, is led to states with the perfect adjustment of frequency (hence,  $\mathcal{K}_f = 1$ ) and with the fixed arrangement of cellular phases. The phases of neighboring cells differ by  $\pm 1$ . Such organization of phases denotes emergence of spiral-wave patterns. The stationary state oscillates either with the natural cellular period  $T = f + r + a$  or with the shortest possible:  $T^* = f + r + 1$ . Rarely, a period occurs which length is between  $T^*$  and  $T$ . In all these cases, the order parameter  $\mathcal{K}_\phi$  is significantly greater than 0 and its value depends on periodicity of the wave. But the spiral patterns are observed in the real cardiac tissue only in the pathological cases [6–8].

Entering rule (3) results in that any two neighboring cells always gain the phase difference equal to zero, asymptotically [14]. It means that in the network of *FRA*-cells we observe the perfect phase synchronization with  $\mathcal{K}_\phi = 1$ .

This state is called *marching cells*. Moreover at certain model parameters waves collapsing inward occur. Again, all such states in the real sinus node tissue are a manifestation of pathological changes.

Notice that because of (3) the periods of states with stationary wave-patterns can be elongated. For example, at wave phase adjustment in a line of *FRA*-cells, the following pattern is stable:

$$\begin{array}{rccccccc}
 t - 1 : & F_{f-2} & F_{f-1} & F_f & R_1 & R_1 & R_2 \\
 t : & F_{f-1} & F_f & R_1 & R_1 & R_2 & R_3 \\
 t + 1 : & F_f & R_1 & R_1 & R_2 & R_2 & R_4
 \end{array} \tag{6}$$

what means that the period of the state oscillations is greater than a natural cell period by 1.

### 3.2 Results from Simulations

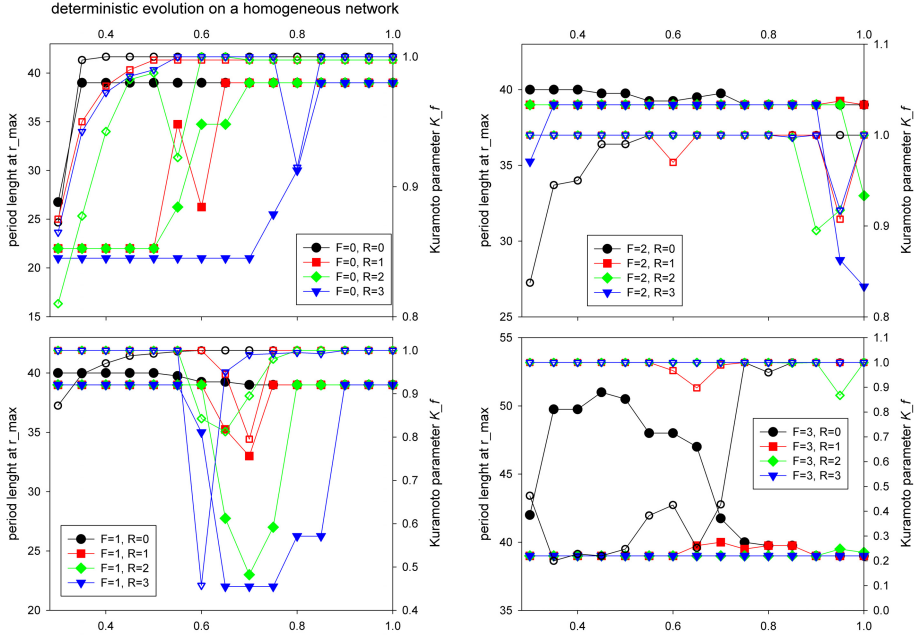
In Figs 3 – 6 the results are shown in terms of order parameters  $\mathcal{K}_f$  and  $\mathcal{K}_\phi$  for densities  $d$  large enough to observe the collective properties.

From Fig. 3, describing systems with deterministically evolving *FRA* cells located on homogeneous networks, we see that the solutions are firmly determined. The parameter  $\mathcal{K}_f$  equals to 1 in large intervals of densities  $d$  and for many values of  $F$  and  $R$ . This means that all cells oscillate with the same period. Only if  $F = 3$  and  $R = 0$  the value of  $\mathcal{K}_f$  drops down. Furthermore, when  $d$  is changing, the switch is observed in the state periodicity from oscillations other than  $T$  (shorter than  $T$  in case  $F = 0, 1$ , or longer than  $T$  in case  $F = 2, 3$ ,) to the oscillation with  $T$ .

States oscillating with period different from  $T$  demand permanent entrainment between cellular states. The transitions are observed for densities in  $\Delta_c = \{d : 0.5 < d < 0.8\}$ . From Fig. 4 we see that there are essential changes in  $\mathcal{K}_\phi$  value for densities in  $\Delta_c$  interval. Outside this interval, if  $d > \Delta_c$ , then  $\mathcal{K}_\phi$  often reaches 1, while in the case of small density the phase synchronization is, in general, small.

Combining the values of order parameters for models with  $F = 0$  and  $R = 1$  or  $R = 2$  with the dominant oscillations presented in Fig. 3, we can give the direct interpretation for  $\mathcal{K}_\phi$  values as follows. If a state is a spiral wave with the shortest period  $T^*$  then  $\mathcal{K}_\phi \approx 0.5$ . If a state oscillates with the cellular period  $T$  then  $\mathcal{K}_f \approx 0.2$ . These estimates agree with the rough assessment based on the mean field calculations.

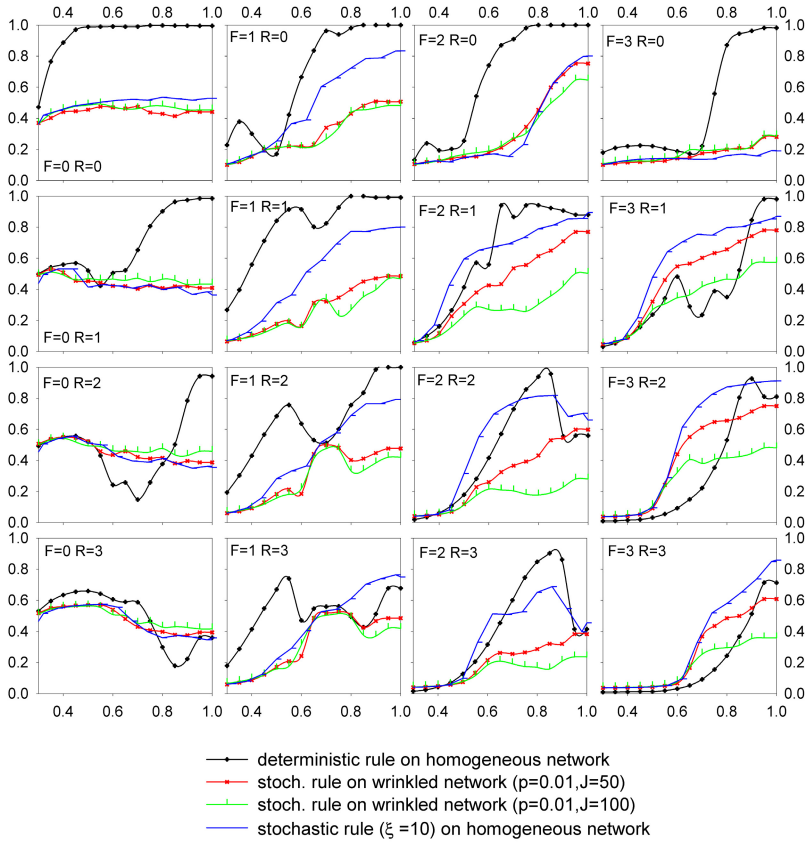
Comparing the listed results to the corresponding properties shown in Fig. 5 we can learn which properties survive inclusion of stochastic dynamics and heterogeneity into the intercellular connections. The rough observation indicates that dependence of dominant oscillation on density  $d$  is similar to that observed in the rigid systems. However, now the participation of the dominant period is much smaller. Notice that for all  $F$  and  $R$  considered by us,  $\mathcal{K}_f < 0.3$ . Moreover, from Fig. 4 we see that  $\mathcal{K}_\phi$  never attains 1. But, on the other hand,  $\mathcal{K}_\phi$  is also significantly distinct from 0. Hence the high level of phase synchronization is



**Fig. 3.** Kuramoto order parameter  $\mathcal{K}_f$  (empty marks) and the most frequent period (filled marks) for different densities  $d$ , and for different sensitivity  $F$  and  $R$  in stationary states obtained from system performing deterministic dynamics on not wrinkled networks. *Simulation condition:* the lattice size  $N = 10^4$  and  $f = 9, r = 11, a = 19$ ; plots represent mean values from  $10^4$  time steps and 50 independent simulation experiments. The first  $10^4$  time steps were skipped to let the system stabilize.

achieved in stationary states. The phase synchronization emerges due to the fact that distributions of periods are concentrated around the dominant period. In Fig. 6 we show distributions of period lengths for some model parameters. We see that in most cases the distribution is a modal one. However, also bimodal distributions appear for densities  $d \in \Delta_c$ .

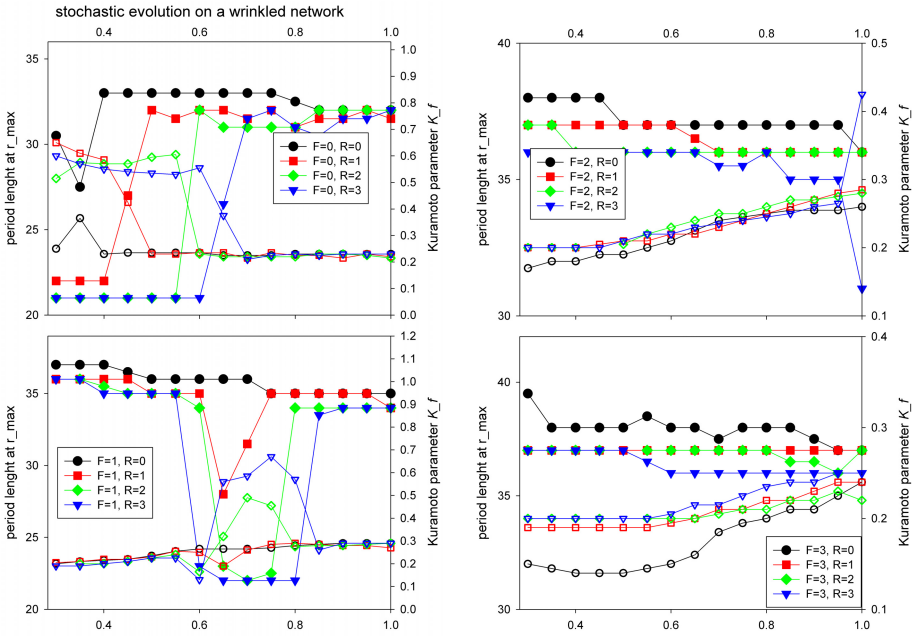
It is worth noting, that for the model parameters  $F = 1, R = 2, 3$  and  $d \in (0.6, 0.7) \subset \Delta_c$ , independently of the cellular dynamics (deterministic or stochastic) and of network structure (homogeneous or intentionally wrinkled), the value of  $\mathcal{K}_\phi$  is the same. So the state of the system is firmly kept with the common oscillations though the value of the period can change from the shortest oscillations to the natural cellular oscillations. Since the natural systems often work at the edge of criticality then we hypothesize that these  $FRA$ -systems correspond to the natural pacemaker in the best way. The strong support for our hypothesis comes from the fact that  $\Delta_c$  falls into densities which are observed in the real sinus node tissue [4].



**Fig. 4.** Kuramoto order parameter  $\mathcal{K}_\phi$  of stationary states for different density of stochastic network connections, sensitivity  $F$  and  $R$  of interactions, cellular dynamics (deterministic versus stochastic with  $\xi = 10$ ), and network structure (homogeneous versus heterogeneous due to wrinkling). Simulations conditions are described in Fig.3

## 4 Discussion

Cardiac cell cultures are becoming important experimental systems of minimal complexity that capture many of the salient features of myocardial tissue function and are simple enough that the tissue parameters can be controlled systematically [11, 12]. Between the two pathological network states of strongly entrained spiral waves and marching cells which are observed in both computer and biological tissue models, there are many states with physiologically justified properties. Hence, by discrete modeling we provide tools to formalize the biological observations. Here, especially Kuramoto order parameters  $\mathcal{K}_f$ ,  $\mathcal{K}_\phi$  justified their ability to qualify and quantify the collective features in the multi parameter model. However, the next challenge is to send the model results back to physiologists, so they can use them iteratively as the input in their further experiments.



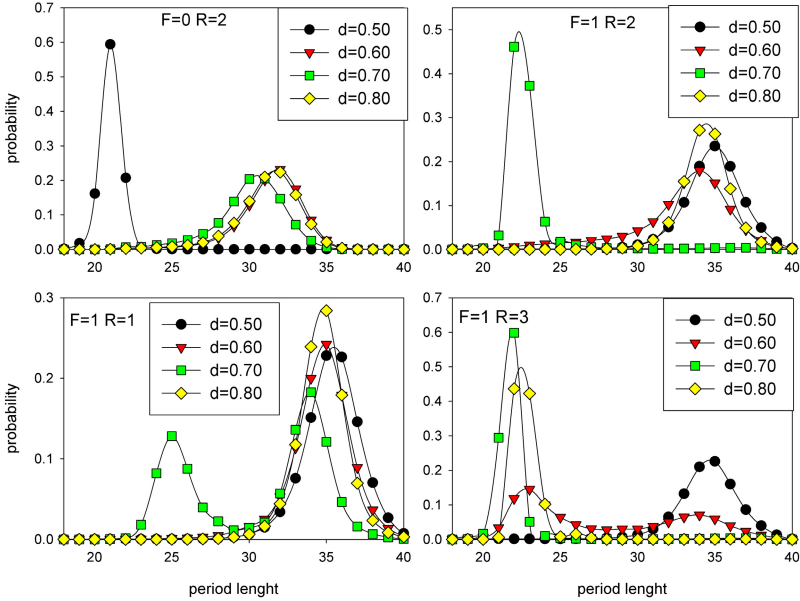
**Fig. 5.** Kuramoto order parameter  $\mathcal{K}_f$  (empty marks) and the most frequent period (filled marks) for different densities  $d$ , and for different sensitivity  $F$  and  $R$  in stationary states obtained from systems driven by stochastic dynamics with  $\xi = 10$ , and on wrinkled networks when  $p = 0.01$ ,  $J = 50$ . Simulations conditions are described in Fig.3.

Therefore, when closing we concentrate on clarification if the model results are able to reveal some changes in the sinus node with age.

The function of the sinus node declines with age, leading to a condition called sick sinus syndrome [5, 8, 21]. Aging causes a decrease in the overall intrinsic heart rate, and an increase in the nodal conduction time. These changes are preceded by a period of tissue remodeling — significant structural changes in the intracellular matrix caused by increase of the collagen tissue. The collagen deposition can be thought as limiting the plasticity of the network connections what translates to *FRA* system as less wrinkled underlying networks of interactions. In consequence, if  $F = 1$ , then tendency to the solution with marching cells appears, see blue lines in Fig. 4. Moreover, together with collagen increase, the density  $d$  could be thought as decreased, what leads, according to our results, to states where strongly entrained spirals emerge.

Age dependent alternations in ion channels (by perturbations in expression and/or function of genes) influence both the intrinsic cellular cycle and sensitivity of a cell for interactions. For example, there is observed a decrease in potassium  $\mathbf{i}_K$  current with aging in the rat sinus node which could be linked to the observed increase in action potential duration with aging [22]. Such properties can be easily coded in *FRA* systems. By varying with durations of particular stages of *FRA*-cell cycle one reconstructs effects of changes in particular parts





**Fig. 6.** Distribution of periods observed in some stationary state at given model parameters: stochastic dynamics,  $\xi = 10$ , wrinkling at  $p = 0.01$ ,  $J = 0.01$ . Other parameters are the same as described under Fig.3.

of the intrinsic cell period. Modifications in values of  $F$  and  $R$  parameters allow to reveal effects of the impairment of sensitivity. It appears that if  $F$  changes from 1 to 2 or 3 then only heterogeneity in the network connection which is strong enough, protects against appearance of entrained spirals. If additionally  $R$  decreases then desynchronization occurs - there is no source of the leading oscillation independently of the level of heterogeneity.

## References

1. Stumpf, M., Balding, D.J., Gorolami, M.: Handbook of Statistical Systems Biology. Wiley (2011)
2. Jalife, J., Delmar, M., Davidenko, J., Anumonwo, J., Kalifa, J.: Basic Cardiac Electrophysiology for the Clinician. Wiley-Blackwell (2009)
3. Klabunde, R.E.: Cardiovascular Physiology Concepts. accessible via, <http://www.cvphysiology.com/Arrhythmias/A005.html>
4. Saffitz, J.E., Lerner, D.L., Yamada, K.A.: Gap Junctions Distribution and regulation in the Heart. In: Zipes, D.P., Jalife, J. (eds.) Cardiac Electrophysiology. From Cell to Bedside, pp. 181–191. Saunders Co., Philadelphia (2004)
5. Dobrzynski, H., Boyett, M.R., Anderson, R.H.: New Insights Into Pacemaker Activity: Promoting Understanding of Sick Sinus. *Circulation* 115, 1921 (2007)
6. Mangoni, M.E., Nargeot, J.: Genesis and Regulation of the Heart Automaticity. *Physiol. Rev.* 89, 919 (2008)

7. Aslanidi, O.V., Boyett, M.R., Dobrzynski, H., Zhang, H.: Mechanisms of transition from normal to reentrant electrical activity in a model of rabbit atrial tissue: interaction of tissue heterogeneity and anisotropy. *Biophysical J.* 96, 7989 (2009)
8. Boyett, M.R.: 'And the beat goes on' The cardiac conduction system: the wiring system of the heart. *Experimental Physiol.* 94, 1035 (2009)
9. Greenberg, J.M., Hastings, S.P.: Spatial patterns for discrete models of diffusion in excitable media. *SIAM J. Appl. Math.* 34, 515 (1978)
10. Berry, H., Fatés, N.: Robustness of the critical behaviour in the stochastic Greenberg-Hastings cellular automaton model. *IJUC* 7, 65 (2011)
11. Bub, G., Shrier, A., Glass, L.: Global Organization of Dynamics in Oscillatory Heterogeneous Excitable Media. *Phys. Rev. Lett.* 94, 028105 (2005)
12. Chang, M.G., Zhang, Y., Chang, C.Y., Xu, L., Emokpae, R., Tung, L., Marban, E., Abraham, M.R.: Spiral waves and reentry dynamics in an in vitro model of the healed infarct border. *Circ. Res.* 105, 1062 (2009)
13. Makowiec, D.: Modeling the sinoatrial node by cellular automata with irregular topology. *Int. J. Mod. Phys. C* 21, 107 (2010)
14. Makowiec, D.: Phase-sensitive cellular automata on stochastic network as a model for cardiac pacemaker rhythmicity. *Acta Phys. Pol. B Proc. Supp.* 5, 85 (2012)
15. Michaels, D.C., Matyas, E.P., Jalife, J.: Dynamic interactions and mutual synchronization of sinoatrial node pacemaker cells. *Circ. Res.* 58, 706 (1986)
16. Anumonvo, J.M., Delmar, M., Vinet, A., Michaels, D.C., Jalife, J.: Phase resetting and entrainment of pacemaker activity in single sinus nodal cells. *Circ. Res.* 68, 1138 (1991)
17. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. *Nature* 393, 409 (1998)
18. Makowiec, D.: Evolving network - simulation study. From a regular lattice to scale free network. *EPJ B* 48, 547 (2005)
19. Kuramoto, Y.: *Chemical Oscillations, Waves and Turbulence*. Springer, Berlin (1984)
20. Acebron, J.A., Bonilla, L.L., Vicente, C.J.P., Ritort, F., Spigler, R.: The Kuramoto model: A simple paradigm for synchronization phenomena. *Rev. Mod. Phys.* 77, 137 (2005)
21. Rose, A.: Keeping the clock ticking as we age: changes in sinoatrial node gene expression and function in the aging heart. *Exp. Physiol.* 96, 1114 (2011)
22. Alings, A.M., Bouman, L.M.: Electrophysiology of the ageing rabbit and cat sinoatrial node-a comparative study. *Eur. Heart J.* 14(9), 1278 (1993)

# Reaction Systems Made Simple\*

## A Normal Form and a Classification Theorem

Luca Manzoni and Antonio E. Porreca

Dipartimento di Informatica, Sistemistica e Comunicazione  
Università degli Studi di Milano-Bicocca  
Viale Sarca 336/14, 20126 Milano, Italy  
`{luca.manzoni,porreca}@disco.unimib.it`

**Abstract.** Reaction systems are models of computation inspired by the interactions between biochemical reactions. We define a notion of multi-step simulation among reaction systems and derive a classification with respect to the amount of resources (reactants and inhibitors) involved in the reactions. We prove that one reactant and one inhibitor per reaction are sufficient to simulate arbitrary systems. Finally, we show that the equivalence relation of mutual simulation induces exactly five linearly ordered classes of reaction systems.

## 1 Introduction

Reaction systems, introduced by Ehrenfeucht and Rozenberg [3,4], are a formalised abstraction of biochemical processes in which the dynamics are discrete in both space and time and are described in terms of *reactions*. A reaction is modelled as a set of *reactants*, necessary for the reaction to take place, a set of *inhibitors*, whose presence blocks the reaction from occurring, and a set of *products*.

Reaction systems may be considered a qualitative model, as opposed to a quantitative one, as we only focus on the presence or absence of chemical species, and not on the precise amounts. In particular, multiple reactions having common reactants do not interfere; indeed, *all* reactions that are enabled at a certain time step happen simultaneously. Another feature of reaction systems which differentiates them from other biologically inspired computational models is the lack of permanency: the state of the system only consists of the products of the reactions that took place in the last time step, without preserving the entities that were not involved in any reaction.

Mathematically, a reaction systems defines a transition function (the *result* function) between states, i.e., sets of entities (chemical species), which completely describes the dynamics of the system. In many cases, the study of the properties of reaction systems involves the comparison of the result functions of different systems or classes of systems. A natural way to understand the modelling power of reaction systems is to consider their behaviour when the amount of resources

---

\* This research was partially funded by Lombardy Region under project NEDD.

(reactants and inhibitors per reactions) is limited. It was proved [2,5] that there exist infinite proper hierarchies of classes of result functions: by allowing more resources, more functions become definable by reaction systems.

While the analysis of result functions is a direct way to compare reaction systems, the classification it provides has a very high granularity. Requiring the equality of the whole dynamics can be restrictive for certain applications where we are interested in a higher-level view of the behaviour of the systems. As an analogy, consider a simulation between Turing machines: we are often not interested in a step-by-step correspondence of configurations, and we allow the simulation to be slower than the original machine. In a similar fashion, in this paper we define a notion of simulation in which the simulating system is allowed to use several steps to simulate a single step of the other system; auxiliary entities (analogous to an alphabet extension) may also be involved in the simulation. The resulting equivalence relation of mutual simulability is coarser than equality of result functions, but still captures the intuitive idea of “having the same behaviour”.

This paper is structured as follows. In Section 2 we recall the definitions and notation related to reaction systems. In Section 3, we introduce the notion of  $k$ -simulation and prove that any reaction system can be  $k$ -simulated by using only one reactant and one inhibitor per reaction. In Section 4 we study reaction systems with no reactants or no inhibitors, and prove that exactly five linearly ordered equivalence classes exist. Finally, in Section 5 we discuss the results and provide some possible directions for further research.

## 2 Basic Notions

In this paper we denote sets by upper-case letters, reactions and atomic elements by lower-case letters, and reaction systems by calligraphic letters. Given a set  $X$ , we denote by  $2^X$  the power set of  $X$ .

A reaction is formally defined as follows.

**Definition 1.** *Given a finite set  $S$  (the background set), a reaction over  $S$  is a triple of sets  $a = (R_a, I_a, P_a) \in 2^S \times 2^S \times 2^S$ . We call  $R_a$  the set of reactants,  $I_a$  the set of inhibitors, and  $P_a$  the set of products.*

Since we will show that one reactant and one inhibitor suffice to simulate any reaction system (see Theorem 1), in this paper we also admit empty reactant and inhibitor sets, as in the original definition [4], in order to investigate the expressivity of the resulting reactions and to prove that they are strictly weaker than reactions involving both kinds of resources.

**Definition 2.** *A reaction system is a pair  $\mathcal{A} = (S, A)$  where  $S$  is a finite set and  $A$  a set of reactions over  $S$ .<sup>1</sup>*

---

<sup>1</sup> We may assume, without loss of generality, the existence of a countably infinite universe including every background set.

A *state* of a reaction system  $\mathcal{A} = (S, A)$  is any subset of  $S$ . The dynamics of a reaction systems are defined as follows.

**Definition 3.** Let  $\mathcal{A} = (S, A)$  be a reaction system,  $a = (R_a, I_a, P_a) \in A$ , and  $T \subseteq S$ . We say that  $a$  is enabled by  $T$  iff  $R_a \subseteq T$  and  $I_a \cap T = \emptyset$ .

The result of  $a$  on  $T$  is defined as

$$\text{res}_a(T) = \begin{cases} P_a & \text{if } a \text{ is enabled by } T \\ \emptyset & \text{otherwise.} \end{cases}$$

The result of  $\mathcal{A}$  on  $T$  is defined as

$$\text{res}_{\mathcal{A}}(T) = \bigcup_{a \in A} \text{res}_a(T).$$

The *state sequence* of a reaction system  $\mathcal{A}$  with initial state  $T$  is given by successive iterations of the result function:

$$(\text{res}_{\mathcal{A}}^n(T))_{n \in \mathbb{N}} = (T, \text{res}_{\mathcal{A}}(T), \text{res}_{\mathcal{A}}^2(T), \dots).$$

Since the background set of a reaction system is finite, the state space is also finite; hence, every state sequence is ultimately periodic.

### 3 A Normal Form for Reaction Systems

We begin by observing that, for each reaction system, there exists another reaction system having the same result function (hence the same behaviour) but using only one product per reaction.

**Proposition 1 (Brijder, Ehrenfeucht, Rozenberg [1]).** For each reaction system  $\mathcal{A} = (S, A)$  there exists a reaction system  $\mathcal{A}' = (S, A')$  over the same background set having reactions with at most one product per reaction and such that  $\text{res}_{\mathcal{A}}(T) = \text{res}_{\mathcal{A}'}(T)$  for all  $T \subseteq S$ .  $\square$

We classify reaction systems according to the maximum amount of reactants and inhibitors appearing in their reactions; the number of products is not used as a parameter due to the proposition above.

**Definition 4.** For all  $i, r \in \mathbb{N}$ , we denote by  $\mathcal{RS}(r, i)$  the class of reaction systems  $\mathcal{A} = (S, A)$  such that, for all  $(R_a, I_a, P_a) \in A$ , we have  $|R_a| \leq r$  and  $|I_a| \leq i$ . We also define the classes  $\mathcal{RS}(\infty, i) = \bigcup_{r \in \mathbb{N}} \mathcal{RS}(r, i)$ ,  $\mathcal{RS}(r, \infty) = \bigcup_{i \in \mathbb{N}} \mathcal{RS}(r, i)$ , and  $\mathcal{RS}(\infty, \infty) = \bigcup_{r, i \in \mathbb{N}} \mathcal{RS}(r, i)$ .

The classification into classes of the form  $\mathcal{RS}(r, i)$  is exhaustive, and the class  $\mathcal{RS}(\infty, \infty)$  contains all reaction systems.

In order to compare reaction systems with respect to their ability to generate state sequences, we define a notion of simulation less restrictive than equality of result functions: here, the simulating system may use several steps to simulate a

single step of the original system. This is consistent with notions of simulation employed for many computational models (e.g., Turing machines), when we are not interested in the strict correspondence of every pair of configurations, but only in the overall behaviour of the two systems.

**Definition 5 (*k*-simulation).** Let  $\mathcal{A} = (S, A)$  and  $\mathcal{A}' = (S', A')$ , with  $S \subseteq S'$ , be reaction systems, and let  $k \in \mathbb{N}$ . We say that  $\mathcal{A}'$  *k*-simulates  $\mathcal{A}$  iff, for all  $T \subseteq S$  and all  $n \in \mathbb{N}$ , we have

$$\text{res}_{\mathcal{A}}^n(T) = \text{res}_{\mathcal{A}'}^{kn}(T) \cap S.$$

In other words, when considering the sequences of states of  $\mathcal{A}$  and  $\mathcal{A}'$  starting from  $T$ , the  $n$ -th state of  $\mathcal{A}$  coincides with the  $(kn)$ -th state of  $\mathcal{A}'$  with respect to the elements of  $S$  (some auxiliary elements of  $S' - S$  may also occur).

We use the notion of *k*-simulation to define a relation on classes of reaction system.

**Definition 6.** Let  $X$  and  $Y$  be classes of reaction systems, and let  $k \in \mathbb{N}$ . We define the binary relation  $\preceq_k$  as follows:  $X \preceq_k Y$  iff for all  $\mathcal{A} \in X$  there exists a reaction system in  $Y$  that  $\ell$ -simulates  $\mathcal{A}$  for some  $\ell \leq k$ .

We say that  $X \preceq Y$  iff  $X \preceq_k Y$  for some  $k \in \mathbb{N}$ . We write  $X \approx_k Y$  if  $X \preceq_k Y$  and  $Y \preceq_k X$ , and  $X \approx Y$  for  $X \preceq Y \wedge Y \preceq X$ . Finally, the notation  $X \prec Y$  is shorthand for  $X \preceq Y \wedge Y \not\preceq X$ .

Notice that  $X \subseteq Y$  implies  $X \preceq_1 Y$ , since any reaction system is trivially 1-simulated by itself.

A *k*-simulation and an  $\ell$ -simulation can be composed into a  $(k\ell)$ -simulation.

**Lemma 1.**  $X \preceq_k Y$  and  $Y \preceq_\ell Z$  implies  $X \preceq_{k\ell} Z$ . □

From this lemma, we immediately get the following result:

**Proposition 2.** The relation  $\preceq$  is a preorder. Hence, the relation  $\approx$  is an equivalence relation. □

We now begin analysing the relationships among the classes of reaction systems defined above. The first step is to show that a reaction system can always be simulated using only one reactant.

**Lemma 2.**  $\mathcal{RS}(r, i) \preceq_3 \mathcal{RS}(1, r)$  for all  $r \geq 1, i \geq 1$ .

*Proof.* Let  $\mathcal{A} = (S, A) \in \mathcal{RS}(r, i)$ . Let  $\mathcal{A}' = (S', A')$  be a reaction system with  $S' = S \cup \hat{S} \cup \bar{S} \cup 2^S$ , where  $\hat{S} = \{\hat{s} : s \in S\}$  and  $\bar{S} = \{\bar{s} : s \in S\}$ . The power set of  $S$  is included in  $S'$  in order to encode subsets of  $S$  as atomic elements in  $\mathcal{A}'$ . The set  $A'$  contains, for each  $s \in S$ , the reactions

$$(\emptyset, \{s\}, \{\bar{s}\}) \quad (\{s\}, \emptyset, \{\hat{s}\}). \quad (1)$$

Furthermore, for each reaction  $a = (R_a, I_a, P_a) \in A$ , with  $R_a = \{x_1, \dots, x_p\}$  and  $I_a = \{y_1, \dots, y_q\}$ , the set  $A'$  contains the following reactions:

$$(\emptyset, \{\bar{x}_1, \dots, \bar{x}_p\}, \{R_a\}) \tag{2}$$

$$(\{\hat{y}_1\}, \emptyset, \{I_a\}), \dots, (\{\hat{y}_q\}, \emptyset, \{I_a\}) \tag{3}$$

$$(\{R_a\}, \{I_a\}, P_a) \tag{4}$$

In order to prove that  $\mathcal{A}'$  3-simulates  $\mathcal{A}$ , we show that the following statement holds: if  $n$  is a multiple of 3 (i.e.,  $n = 3m$  for some  $m$ ), then

$$\text{res}_{\mathcal{A}'}^n(T) \cap S = \text{res}_{\mathcal{A}}^{n/3}(T); \tag{5}$$

if  $n = 3m + 1$ , then

$$\text{res}_{\mathcal{A}'}^n(T) \cap (\hat{S} \cup \bar{S}) = \{\hat{y} : y \in \text{res}_{\mathcal{A}}^{(n-1)/3}(T)\} \cup \{\bar{x} : x \notin \text{res}_{\mathcal{A}}^{(n-1)/3}(T)\}, \tag{6}$$

and if  $n = 3m + 2$ , then

$$\begin{aligned} \text{res}_{\mathcal{A}'}^n(T) \cap 2^S &= \{R_a : R_a \subseteq \text{res}_{\mathcal{A}}^{(n-2)/3}(T)\} \cup \\ &\quad \{I_a : I_a \cap \text{res}_{\mathcal{A}}^{(n-2)/3}(T) \neq \emptyset\}. \end{aligned} \tag{7}$$

By induction on  $n$ : if  $n = 0$ , then (5) holds by definition.

If  $n > 0$  has the form  $3m + 1$ , then by induction hypothesis we have

$$\text{res}_{\mathcal{A}'}^{n-1}(T) \cap S = \text{res}_{\mathcal{A}}^{(n-1)/3}(T).$$

Notice that the only reactions producing elements of  $\hat{S}$  or  $\bar{S}$  are those in (1), which, for every  $s \in S$ , produce  $\hat{s}$  if  $s \in \text{res}_{\mathcal{A}'}^{n-1}(T)$ , and  $\bar{s}$  otherwise. As a consequence, statement (6) holds.

If  $n > 0$  has the form  $3m + 2$ , then by induction hypothesis we have

$$\text{res}_{\mathcal{A}'}^{n-1}(T) \cap (\hat{S} \cup \bar{S}) = \{\hat{y} : y \in \text{res}_{\mathcal{A}}^{(n-2)/3}(T)\} \cup \{\bar{x} : x \notin \text{res}_{\mathcal{A}}^{(n-2)/3}(T)\}.$$

The only reactions having elements of  $2^S$  as products are (2) and (3): for every reaction  $a \in A$ , the set  $\{R_a\}$  is produced iff for all  $x \in R_a$  we have  $\bar{x} \notin \text{res}_{\mathcal{A}'}^{n-1}(T)$ , which is equivalent (by induction hypothesis) to  $x \in \text{res}_{\mathcal{A}}^{(n-2)/3}(T)$ . Furthermore, for every  $a \in A$ , the set  $\{I_a\}$  is produced iff there exists at least one  $y \in I_a$  such that  $\hat{y} \in \text{res}_{\mathcal{A}'}^{n-1}(T)$ , which in turn means that  $y \in \text{res}_{\mathcal{A}}^{(n-2)/3}(T)$ . Hence, statement (7) holds.

Finally, if  $n > 0$  has the form  $3m$ , by induction hypothesis we have

$$\begin{aligned} \text{res}_{\mathcal{A}'}^{n-1}(T) \cap 2^S &= \{R_a : R_a \subseteq \text{res}_{\mathcal{A}}^{(n-3)/3}(T)\} \cup \\ &\quad \{I_a : I_a \cap \text{res}_{\mathcal{A}}^{(n-3)/3}(T) \neq \emptyset\}. \end{aligned}$$

The only reactions having products in  $S$  are of the form (4). For every reaction  $a = (R_a, I_a, P_a) \in A$ , the corresponding reaction  $(\{R_a\}, \{I_a\}, P_a) \in A'$  is enabled in  $\mathcal{A}'$  at time  $n - 1$  iff  $a$  is enabled in  $\mathcal{A}$  at time  $\frac{n-3}{3} = \frac{n}{3} - 1$ . Hence, the two reaction systems  $\mathcal{A}$  and  $\mathcal{A}'$  have the same state (ignoring symbols in  $S' - S$ ) at time  $\frac{n}{3}$  and  $n$  respectively, as required.

In particular, statement (5) holds for all  $n$ , i.e.,  $\mathcal{A}'$  3-simulates  $\mathcal{A}$ .  $\square$

Now we show that the number of inhibitors can also be reduced to one.

**Lemma 3.**  $\mathcal{RS}(r, i) \preceq_2 \mathcal{RS}(\max\{r, 1\}, 1)$ .

*Proof.* Let  $\mathcal{A} = (S, A) \in \mathcal{RS}(r, i)$ . Consider the reaction system  $\mathcal{A}' = (S', A')$  with  $S' = S \cup 2^S$  and  $A'$  containing, for each reaction  $a = (R_a, I_a, P_a) \in A$  with  $I_a = \{s_1, \dots, s_n\}$ , the following reactions:

$$(\{s_1\}, \emptyset, \{I_a\}), \dots, (\{s_n\}, \emptyset, \{I_a\}) \quad (R_a, \emptyset, \{R_a\}) \quad (\{R_a\}, \{I_a\}, P_a).$$

In order to prove that  $\mathcal{A}'$  2-simulates  $\mathcal{A}$ , we can show by induction that

$$\text{res}_{\mathcal{A}'}^n(T) = \text{res}_{\mathcal{A}}^{n/2}(T) \quad (8)$$

if  $n$  is even, and

$$\text{res}_{\mathcal{A}'}^n(T) = \{R_a : R_a \subseteq \text{res}_{\mathcal{A}}^{(n-1)/2}(T)\} \cup \{I_a : I_a \cap \text{res}_{\mathcal{A}}^{(n-1)/2}(T) \neq \emptyset\}$$

if  $n$  is odd; the conditions are easily seen to hold by considering the form of the reactions of  $\mathcal{A}'$ . The result then follows immediately from (8).  $\square$

By combining the previous two lemmata, we are finally able to show that one reactant and one inhibitor can simulate any reaction systems, thus providing a normal form into which every reaction system can be reduced.

**Theorem 1 (Normal form).**  $\mathcal{RS}(\infty, \infty) \approx_6 \mathcal{RS}(1, 1)$ .

*Proof.* By definition we have  $\mathcal{RS}(1, 1) \subseteq \mathcal{RS}(\infty, \infty)$ , thus it follows immediately that  $\mathcal{RS}(1, 1) \preceq_6 \mathcal{RS}(\infty, \infty)$ . Conversely, if  $r \geq 1$  and  $i \geq 1$  we have  $\mathcal{RS}(r, i) \preceq_3 \mathcal{RS}(1, r)$  by Lemma 2 and  $\mathcal{RS}(1, r) \preceq_2 \mathcal{RS}(\max\{1, 1\}, 1) = \mathcal{RS}(1, 1)$  by Lemma 3, hence  $\mathcal{RS}(i, r) \preceq_6 \mathcal{RS}(1, 1)$  by transitivity (Lemma 1). Since  $\mathcal{RS}(r, i) \preceq_1 \mathcal{RS}(r+1, i+1)$ , the result holds even if  $i = 0$  or  $r = 0$ .  $\square$

## 4 Classification of Reaction Systems

Having established a minimum amount of resources needed to simulate general reaction systems, we are interested in analysing the behaviour of weaker systems, i.e., with reactions involving no reactants or no inhibitors.

First of all, we show that the number of reactants in a reaction with no inhibitors can be halved, provided that their initial number is greater than two.

**Lemma 4.**  $\mathcal{RS}(r, 0) \preceq_2 \mathcal{RS}(\lceil \frac{r}{2} \rceil, 0)$  for all  $r > 2$ .

*Proof.* Let  $\mathcal{A} = (S, A) \in \mathcal{RS}(r, 0)$ . We simulate  $\mathcal{A}$  with another reaction system  $\mathcal{A}' = (S', A')$ , where  $S' = S \cup 2^S$ .

Each reaction  $a = (R_a, \emptyset, P_a) \in A$  is simulated by at most three reactions in  $\mathcal{A}'$ . We can write  $R_a$  as the union of (non necessarily distinct) sets  $R_1$  and  $R_2$  consisting of at most  $\lceil \frac{r}{2} \rceil$  elements each. The reaction  $a$  is then simulated by

$$(R_1, \emptyset, \{R_1\}) \quad (R_2, \emptyset, \{R_2\}) \quad (\{R_1, R_2\}, \emptyset, P_a).$$

Notice that  $\mathcal{A}' \in \mathcal{RS}(\lceil \frac{r}{2} \rceil, 0)$ . In order to show that  $\mathcal{A}'$  2-simulates  $\mathcal{A}$ , we prove by induction on  $n$  that



- if  $n$  is even, then  $\text{res}_{\mathcal{A}'}^n(T) \cap S = \text{res}_{\mathcal{A}}^{n/2}(T)$ ;
- if  $n$  is odd, then for all  $a = (R_a, \emptyset, P_a) \in A$  we have  $R_a \subseteq \text{res}_{\mathcal{A}}^{(n-1)/2}(T)$  iff there exist (non necessarily distinct) sets  $Q_1, Q_2 \subseteq S$  such that  $R_a = Q_1 \cup Q_2$  and  $Q_1, Q_2 \in \text{res}_{\mathcal{A}'}^n(T)$ .

If  $n = 0$ , then the condition clearly holds, since  $\text{res}_{\mathcal{A}'}^0(T) \cap S = T = \text{res}_{\mathcal{A}}^0(T)$ .

If  $n > 0$  is odd, by induction hypothesis  $\text{res}_{\mathcal{A}'}^{n-1}(T) \cap S = \text{res}_{\mathcal{A}}^{(n-1)/2}(T)$ . Hence, for all  $a \in A$ , we have  $R_a \subseteq \text{res}_{\mathcal{A}}^{(n-1)/2}(T)$  iff  $R_a \subseteq \text{res}_{\mathcal{A}'}^{n-1}(T) \cap S$ ; this is equivalent to the existence of  $Q_1, Q_2 \subseteq \text{res}_{\mathcal{A}'}^{n-1}(T)$  such that  $Q_1 \cup Q_2 = R_a$ ; in particular, by letting  $Q_1 = R_1$  and  $Q_2 = R_2$  as described above, we get  $Q_1, Q_2 \in \text{res}_{\mathcal{A}'}^n(T)$  by applying the reactions. Conversely, if there exist sets  $Q_1, Q_2 \subseteq S$  such that  $R_a = Q_1 \cup Q_2$  and  $Q_1, Q_2 \in \text{res}_{\mathcal{A}'}^n(T)$ , these are necessarily produced by the two reactions  $(Q_1, \emptyset, \{Q_1\})$  and  $(Q_2, \emptyset, \{Q_2\})$ , implying  $Q_1, Q_2 \subseteq \text{res}_{\mathcal{A}'}^{n-1}(T)$ , that is  $R_a \subseteq \text{res}_{\mathcal{A}'}^{n-1}(T) \cap S = \text{res}_{\mathcal{A}}^{(n-1)/2}(T)$ .

Now assume  $n > 0$  and even. Let  $x \in \text{res}_{\mathcal{A}'}^n(T) \cap S$ . Then  $x \in P_a$  for some reaction  $a' = (\{R_1, R_2\}, \emptyset, P_a) \in A'$  enabled at time  $n-1$ , hence  $R_1, R_2 \in \text{res}_{\mathcal{A}'}^{n-1}(T)$ . The reaction  $a'$  has a corresponding reaction  $a = (R_a, \emptyset, P_a) \in A$  with  $R_a = R_1 \cup R_2$ . Then, by induction hypothesis, we have  $R_a \subseteq \text{res}_{\mathcal{A}}^{(n-2)/2}(T)$ : reaction  $a$  is enabled in  $\mathcal{A}$  at time  $\frac{n-2}{2}$ , producing  $x$  at time  $\frac{n}{2}$ . Conversely, let  $x \in \text{res}_{\mathcal{A}}^{n/2}(T)$ . Then  $x \in P_a$  for some reaction  $a = (R_a, \emptyset, P_a) \in A$  and  $R_a \subseteq \text{res}_{\mathcal{A}}^{(n-2)/2}(T)$ ; by induction hypothesis then  $R_a \subseteq \text{res}_{\mathcal{A}'}^{n-2}(T) \cap S$ . Since  $A'$  contains reactions of the forms  $(R_1, \emptyset, \{R_1\})$ ,  $(R_2, \emptyset, \{R_2\})$ , and  $(\{R_1, R_2\}, \emptyset, P_a)$  with  $R_1 \cup R_2 = R_a$ , the element  $x$  is produced in two steps, i.e.,  $x \in \text{res}_{\mathcal{A}'}^n(T) \cap S$ .

In particular, we have  $\text{res}_{\mathcal{A}'}^{2n}(T) \cap S = \text{res}_{\mathcal{A}}^n(T)$  for all  $n \in \mathbb{N}$ .  $\square$

By iterating Lemma 4 the number of reactants can be reduced to two.

**Proposition 3.**  $\mathcal{RS}(\infty, 0) \approx \mathcal{RS}(2, 0)$ .

*Proof.* By definition we have  $\mathcal{RS}(2, 0) \preceq_1 \mathcal{RS}(\infty, 0)$ . By applying Lemma 4 repeatedly we obtain  $\mathcal{RS}(r, 0) \preceq_r \mathcal{RS}(2, 0)$ , implying  $\mathcal{RS}(\infty, 0) \preceq \mathcal{RS}(2, 0)$ .  $\square$

In a reaction system where only one reactant per reaction is allowed, each element appearing at a given time in the state of the system can be either traced back to a single element of the initial state, or it is always generated, independently of the initial state.

**Lemma 5.** *Let  $\mathcal{A} = (S, A) \in \mathcal{RS}(1, 0)$ . Then, for all  $T \subseteq S$ , for all  $n \in \mathbb{N}$ , for all  $x \in \text{res}_{\mathcal{A}}^n(T)$  either there exists  $y \in T$  such that  $x \in \text{res}_{\mathcal{A}}^n(\{y\})$ , or we have  $x \in \text{res}_{\mathcal{A}}^n(\emptyset)$ .*

*Proof.* By induction on  $n$ . When  $n = 0$  we have  $\text{res}_{\mathcal{A}}^0(T) = T$  and  $x \in T$ . Hence, it suffices to choose  $y = x$ .

Now assume  $n > 0$  and let  $x \in \text{res}_{\mathcal{A}}^n(T)$ . There are two cases: either  $x$  is generated by a reaction  $(\emptyset, \emptyset, \{x\}) \in A$ , or by a reaction  $(\{z\}, \emptyset, \{x\}) \in A$  for some  $z \in S$ . In the first case, we have  $x \in \text{res}_{\mathcal{A}}^n(\emptyset)$ . Otherwise, by induction hypothesis, there are two sub-cases:

- either  $z \in \text{res}_{\mathcal{A}}^{n-1}(\emptyset)$ , and then  $x \in \text{res}_{\mathcal{A}}^n(\emptyset)$ , or
- there exists  $y \in T$  such that  $z \in \text{res}_{\mathcal{A}}^{n-1}(\{y\})$ , and then  $x \in \text{res}_{\mathcal{A}}^n(\{y\})$ .  $\square$

As a consequence, reaction systems where two reactants per reaction are allowed can produce more complex state sequences than those with only one reactant, since the generation of products may depend on the simultaneous presence of several reactants.

**Proposition 4.**  $\mathcal{RS}(1, 0) \prec \mathcal{RS}(2, 0)$ .

*Proof.* Clearly  $\mathcal{RS}(1, 0) \preceq_1 \mathcal{RS}(2, 0)$ .

Let  $\mathcal{A} = (S, A) \in \mathcal{RS}(2, 0)$  be defined by  $S = \{x, y, z\}$  and the reaction  $(\{x, y\}, \emptyset, \{z\})$ . Suppose that  $\mathcal{A}' \in \mathcal{RS}(1, 0)$   $k$ -simulates  $\mathcal{A}$  for some  $k$ . We have  $\text{res}_{\mathcal{A}}(\{x, y\}) = \{z\}$ , hence  $\text{res}_{\mathcal{A}'}^k(\{x, y\}) \cap S = \{z\}$ . By Lemma 5, one of the two following conditions holds:

$$z \in \text{res}_{\mathcal{A}'}^k(\emptyset) \tag{9}$$

$$z \in \text{res}_{\mathcal{A}'}^k(\{w\}) \text{ for some } w \in \{x, y\}. \tag{10}$$

If (9) holds, then we have  $z \in \text{res}_{\mathcal{A}'}^k(\emptyset) \cap S \neq \text{res}_{\mathcal{A}}(\emptyset) = \emptyset$ , contradicting the fact that  $\mathcal{A}'$   $k$ -simulates  $\mathcal{A}$ . On the other hand, if (10) holds, we have  $z \in \text{res}_{\mathcal{A}'}^k(\{w\}) \cap S \neq \text{res}_{\mathcal{A}}(\{w\}) = \emptyset$ , once again a contradiction.

Therefore  $\mathcal{A}$  cannot be  $k$ -simulated by any reaction system in  $\mathcal{RS}(1, 0)$ .  $\square$

In the absence of inhibitors, at least one reactant is needed in order to obtain state sequences that actually depend on the initial state.

**Proposition 5.**  $\mathcal{RS}(0, 0) \prec \mathcal{RS}(1, 0)$ .

*Proof.* Clearly  $\mathcal{RS}(0, 0) \preceq_1 \mathcal{RS}(1, 0)$ .

We prove that there exists  $\mathcal{A} = (A, S) \in \mathcal{RS}(1, 0)$  such that no  $\mathcal{A}' = (A', S') \in \mathcal{RS}(0, 0)$  simulates  $\mathcal{A}$ . Observe that  $\text{res}_{\mathcal{A}'}^n(T) = \text{res}_{\mathcal{A}'}(\emptyset)$  for all  $T \subseteq S'$  and  $n \geq 1$ , i.e, the evolution of  $\mathcal{A}'$  reaches a fixed point immediately after the first step, irrespective of the initial state. On the other hand, if  $\mathcal{A}$  is defined by  $S = \{x\}$  with the reaction  $(\{x\}, \emptyset, \{x\})$ , we have

$$\text{res}_{\mathcal{A}}(\emptyset) = \emptyset \neq \{x\} = \text{res}_{\mathcal{A}}(\{x\}).$$

Hence  $\mathcal{RS}(1, 0) \not\preceq \mathcal{RS}(0, 0)$ .  $\square$

Unlike reactants, any number of inhibitors can be simulated by a single one.

**Proposition 6.**  $\mathcal{RS}(0, \infty) \approx_3 \mathcal{RS}(0, 1)$ .

*Proof.* Trivially,  $\mathcal{RS}(0, 1) \preceq_3 \mathcal{RS}(0, \infty)$  holds.

Let  $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$ , and let  $\mathcal{A}' = (S', A') \in \mathcal{RS}(0, 1)$  with  $S' = S \cup \bar{S} \cup 2^S$ , where  $\bar{S} = \{\bar{x} : x \in S\}$ . For each  $x \in S$ ,  $A'$  contains the reaction

$$(\emptyset, \{x\}, \{\bar{x}\}) \tag{11}$$

and, for each  $a = (\emptyset, I_a, P_a)$  with  $I_a = \{x_1, \dots, x_p\}$ , the reactions

$$(\emptyset, \{\bar{x}_1\}, \{I_a\}), \dots, (\emptyset, \{\bar{x}_p\}, \{I_a\}) \tag{12}$$

$$(\emptyset, \{I_a\}, P_a). \tag{13}$$

We prove, by induction on  $n$ , that for all  $T \subseteq S$  we have

$$\text{res}_{\mathcal{A}'}^n(T) \cap S = \text{res}_{\mathcal{A}}^{n/3}(T) \quad \text{if } n = 3m; \tag{14}$$

$$\bar{x} \in \text{res}_{\mathcal{A}'}^n(T) \iff x \notin \text{res}_{\mathcal{A}}^{(n-1)/3}(T) \quad \text{if } n = 3m + 1; \tag{15}$$

$$I_a \in \text{res}_{\mathcal{A}'}^n(T) \cap 2^S \iff I_a \cap \text{res}_{\mathcal{A}}^{(n-2)/3}(T) \neq \emptyset \quad \text{if } n = 3m + 2. \tag{16}$$

For  $n = 0$ , we have  $\text{res}_{\mathcal{A}'}^0(T) \cap S = T = \text{res}_{\mathcal{A}}^0(T)$ .

If  $n > 0$  is a multiple of 3, then by induction hypothesis

$$I_a \in \text{res}_{\mathcal{A}'}^{n-1}(T) \cap 2^S \iff I_a \cap \text{res}_{\mathcal{A}}^{(n-3)/3}(T) \neq \emptyset.$$

Notice that, if  $X \in \text{res}_{\mathcal{A}'}^{n-1}(T) \cap 2^S$ , then necessarily  $X = I_a$  for some  $a \in A$ , as the only reactions producing elements of  $2^S$  have the form (12). For each reaction  $a \in A$  we have a corresponding reaction  $a'$  of type (13), and  $a$  is inhibited at time  $\frac{n-3}{3}$  in  $\mathcal{A}$  iff  $a'$  is inhibited at time  $n-1$  in  $\mathcal{A}'$ : statement (14) follows.

If  $n > 0$  with  $n = 3m + 1$ , by induction hypothesis we have

$$\text{res}_{\mathcal{A}'}^{n-1}(T) \cap S = \text{res}_{\mathcal{A}}^{(n-1)/3}(T).$$

We have  $\bar{x} \in \text{res}_{\mathcal{A}'}^n(T)$  iff the reaction  $(\emptyset, \{x\}, \{\bar{x}\})$  was enabled at time  $n-1$ , that is  $x \notin \text{res}_{\mathcal{A}}^{n-1}(T) \cap S = \text{res}_{\mathcal{A}}^{(n-1)/3}(T)$  as required.

Finally, if  $n > 0$  with  $n = 3m + 2$ , by induction hypothesis

$$\bar{x} \in \text{res}_{\mathcal{A}'}^{n-1}(T) \iff x \notin \text{res}_{\mathcal{A}}^{(n-2)/3}(T).$$

Let  $a \in A$ . We have  $I_a \in \text{res}_{\mathcal{A}'}^n(T) \cap 2^S$  iff at least one of the reactions of the form (12) was enabled at time  $n-1$ . This means that there exists  $x \in I_a$  such that  $\bar{x} \notin \text{res}_{\mathcal{A}}^{n-1}(T)$  and  $x \in \text{res}_{\mathcal{A}}^{(n-2)/3}(T)$ . Equivalently,  $I_a \cap \text{res}_{\mathcal{A}}^{(n-2)/3}(T) \neq \emptyset$ . This proves (16).

The statement of the proposition immediately follows from (14). □

Perhaps surprisingly, reactants can also be simulated by a single inhibitor.

**Lemma 6.**  $\mathcal{RS}(\infty, 0) \preceq_2 \mathcal{RS}(0, 1)$ .

*Proof.* Let  $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ . Let  $\mathcal{A}' = (S', A') \in \mathcal{RS}(0, 1)$  with  $S' = S \cup 2^S$  and having, for each reaction  $a = (R_a, \emptyset, P_a)$  with  $R_a = x_1, \dots, x_p$ , the following set of reactions:

$$(\emptyset, \{x_1\}, \{R_a\}), \dots, (\emptyset, \{x_p\}, \{R_a\}) \tag{17}$$

$$(\emptyset, \{R_a\}, P_a). \tag{18}$$

Let  $T \subseteq S$ . We prove, by induction on  $n$ , that

$$\text{res}_{\mathcal{A}'}^n(T) \cap S = \text{res}_{\mathcal{A}}^{n/2}(T) \quad \text{if } n \text{ is even} \quad (19)$$

$$R_a \in \text{res}_{\mathcal{A}'}^n(T) \iff R_a \not\subseteq \text{res}_{\mathcal{A}}^{(n-1)/2}(T) \quad \text{if } n \text{ is odd.} \quad (20)$$

For  $n = 0$  we have  $\text{res}_{\mathcal{A}'}^0(T) \cap S = T = \text{res}_{\mathcal{A}}^0(T)$ .

For even  $n > 0$  we have, by induction hypothesis,

$$R_a \in \text{res}_{\mathcal{A}'}^{n-1}(T) \iff R_a \not\subseteq \text{res}_{\mathcal{A}}^{(n-2)/2}(T).$$

Notice that the only reactions in  $\mathcal{A}'$  with products in  $S$  have the form (18), and they are enabled at time  $n-1$  iff  $R_a \subseteq \text{res}_{\mathcal{A}}^{(n-2)/2}(T)$ , i.e., iff reaction  $a$  is enabled in  $\mathcal{A}$  at time  $\frac{n-2}{2}$ . Condition (19) follows.

For odd  $n > 0$ , by induction hypothesis we have

$$\text{res}_{\mathcal{A}'}^{n-1}(T) \cap S = \text{res}_{\mathcal{A}}^{(n-1)/2}(T)$$

The only reactions of  $\mathcal{A}'$  having products in  $2^S$  have the form (17). The element  $R_a$  is produced iff there exists  $x \in R_a$  with  $x \notin \text{res}_{\mathcal{A}'}^{n-1}(T) \cap S$ , i.e., iff reaction  $a$  is *not* enabled in  $\mathcal{A}$  at time  $\frac{n-1}{2}$ , as in (20).

The statement of the lemma follows from (19).  $\square$

The following two lemmata provide a property of the result function of reaction systems without inhibitors and without reactants, respectively.

**Lemma 7.** *If  $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ , then the function  $\text{res}_{\mathcal{A}}$  is monotone, i.e.,  $T_1 \subseteq T_2$  implies  $\text{res}_{\mathcal{A}}(T_1) \subseteq \text{res}_{\mathcal{A}}(T_2)$  for all  $T_1, T_2 \subseteq S$ . As a consequence, the function  $\text{res}_{\mathcal{A}}^n$  is monotone for all  $n \in \mathbb{N}$ .*

*Proof.* For each reaction  $a \in A$ , if  $a$  is enabled by  $T_1$  then it is also enabled by  $T_2 \supseteq T_1$ , as  $a$  has no inhibitors. Thus,  $\text{res}_a(T_1) \subseteq \text{res}_a(T_2)$ , and

$$\text{res}_{\mathcal{A}}(T_1) = \bigcup_{a \in A} \text{res}_a(T_1) \subseteq \bigcup_{a \in A} \text{res}_a(T_2) = \text{res}_{\mathcal{A}}(T_2).$$

The function  $\text{res}_{\mathcal{A}}^n$  is monotone by induction on  $n$ .  $\square$

In a similar way, the next result can be proved.

**Lemma 8.** *If  $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$ , then the function  $\text{res}_{\mathcal{A}}$  is anti-monotone, i.e.,  $T_1 \subseteq T_2$  implies  $\text{res}_{\mathcal{A}}(T_1) \supseteq \text{res}_{\mathcal{A}}(T_2)$  for all  $T_1, T_2 \subseteq S$ . As a consequence, the function  $\text{res}_{\mathcal{A}}^n$  is anti-monotone for odd  $n$ , and monotone for even  $n$ .  $\square$*

These properties imply that reaction systems using one inhibitor exclusively can produce state sequences that no reaction system using only two reactants (or, by Proposition 3, any number of reactants) can generate.

**Proposition 7.**  $\mathcal{RS}(2, 0) \prec \mathcal{RS}(0, 1)$ .

*Proof.* By Lemma 6 we have  $\mathcal{RS}(2, 0) \preceq_2 \mathcal{RS}(0, 1)$ .

Let  $\mathcal{A} = (S, A) \in \mathcal{RS}(0, 1)$  be defined by  $S = \{x\}$  and  $(\emptyset, \{x\}, \{x\})$  as the only reaction. By Lemma 8, the function  $\text{res}_{\mathcal{A}}$  is anti-monotone (furthermore, it is not monotone as it is not the identity function). By Lemma 7, for any  $\mathcal{A}' \in \mathcal{RS}(2, 0)$  the function  $\text{res}_{\mathcal{A}'}^k$  is monotone for all  $k \in \mathbb{N}$ . Therefore,  $\mathcal{A}'$  cannot  $k$ -simulate  $\mathcal{A}$ .  $\square$

Finally, we show that both reactants and inhibitors are needed in order to simulate arbitrary state sequences, thus proving the minimality of the normal form of Theorem 1.

**Proposition 8.**  $\mathcal{RS}(0, 1) \prec \mathcal{RS}(1, 1)$ .

*Proof.* Trivially, we have  $\mathcal{RS}(0, 1) \preceq_1 \mathcal{RS}(1, 1)$ .

Consider the reaction system  $\mathcal{A} = (S, A) \in \mathcal{RS}(1, 1)$  defined by  $S = \{x, y\}$  and the reaction  $(\{x\}, \{y\}, \{x, y\})$ . We have

$$\text{res}_{\mathcal{A}}(\emptyset) = \emptyset \qquad \text{res}_{\mathcal{A}}(\{x\}) = \{x, y\} \qquad \text{res}_{\mathcal{A}}(\{x, y\}) = \emptyset.$$

Hence,  $\text{res}_{\mathcal{A}}$  is neither monotone nor anti-monotone. No  $\mathcal{A}' \in \mathcal{RS}(0, 1)$  can simulate  $\mathcal{A}$ , since (by Lemma 8) the function  $\text{res}_{\mathcal{A}'}^k$  is monotone for even  $k$  and anti-monotone for odd  $k$ .  $\square$

All the results proved in this paper can be summarised by the following theorem, which provides a complete classification of reaction systems with respect to the number of reactants and inhibitors per reaction.

**Theorem 2.** *The relation  $\preceq$  is a total preorder on the set of classes of reaction systems of the form  $\mathcal{RS}(r, i)$ . The classes are comparable according to the following diagram for all  $r, i \geq 2$ :*

$$\begin{array}{ccccccc} \mathcal{RS}(0, 0) & \prec & \mathcal{RS}(1, 0) & \prec & \mathcal{RS}(2, 0) & \prec & \mathcal{RS}(0, 1) & \prec & \mathcal{RS}(1, 1) \\ & & \Downarrow & & \Downarrow & & \Downarrow & & \\ & & \mathcal{RS}(r, 0) & \prec & \mathcal{RS}(0, i) & \prec & \mathcal{RS}(r, i) & & \\ & & \Downarrow & & \Downarrow & & \Downarrow & & \\ & & \mathcal{RS}(\infty, 0) & \prec & \mathcal{RS}(0, \infty) & \prec & \mathcal{RS}(\infty, \infty) & & \end{array}$$

*In particular, the relation  $\approx$  induces exactly five equivalence classes.*  $\square$

## 5 Further Remarks

After having introduced the notion of  $k$ -simulation, we have proved that every reaction system  $\mathcal{A} = (S, A) \in \mathcal{RS}(r, i)$  can be simulated by using one reactant

and one inhibitor per reaction. We have then analysed reaction systems with no reactants or no inhibitors, showing that there exist a finite, linear hierarchy of non-equivalent classes of reaction systems.

The simulating reaction system  $\mathcal{A}'$  has a linear slowdown, that is, simulating  $n$  steps of the original system is performed in  $kn$  steps, and  $k$  is usually independent of  $|S|$ ,  $|A|$ ,  $r$ , and  $i$ . The only exception is the simulation in  $O(rn)$  steps of an  $\mathcal{RS}(r, 0)$  reaction system by means of an  $\mathcal{RS}(2, 0)$  reaction system. Furthermore, the size of  $\mathcal{A}'$  can be always made polynomial with respect to the size of  $\mathcal{A}$ : even if we often include, for the sake of simplicity, the whole power set of  $S$  in the background set of  $\mathcal{A}'$ , only a polynomial number of elements (depending on  $|S|$  and on the number of reactions of  $\mathcal{A}$ ) actually appear as reactants, inhibitors, or products: hence, the remaining ones can be simply removed. The number of reactions of  $\mathcal{A}'$  is also polynomial with respect to the size of  $\mathcal{A}$ .

Although in this paper we focused only on system where the input is all given in the initial state, the original definition [4] allows the system to receive further input (i.e., new elements to be inserted in the state of the system) at every step. It is possible to extend the definition of  $k$ -simulation to this case, and prove all the results above in the new setting.

## 5.1 Open Problems

An open problem involves the minimality of  $k$  in certain  $k$ -simulations described here. For instance, the  $k$ -simulation of Proposition 6 is provably minimal by anti-monotonicity (Lemma 8); similarly, in Lemma 6 the simulation cannot be performed in one step (by Lemmata 7 and 8). In the  $r$ -simulation of Proposition 3 the dependency on  $r$  is probably unavoidable. On the other hand, it is unknown whether there exist reaction systems necessarily requiring a 6-simulation in order to reduce reactants and inhibitors to one. Furthermore, most  $k$ -simulations in this paper employ auxiliary elements, and it seems unlikely that they can always be eliminated. Can we at least ensure that every  $(kn)$ -th state of the simulating system is identical to the  $n$ -th state of the simulated one?

## References

1. Brijder, R., Ehrenfeucht, A., Rozenberg, G.: Reaction systems with duration. In: Kelemen, J., Kelemenová, A. (eds.) *Computation, Cooperation, and Life*. LNCS, vol. 6610, pp. 191–202. Springer, Heidelberg (2011)
2. Ehrenfeucht, A., Main, M., Rozenberg, G.: Functions defined by reaction systems. *International Journal of Foundations of Computer Science* 22(1), 167–168 (2011)
3. Ehrenfeucht, A., Rozenberg, G.: Basic notions of reaction systems. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) *DLT 2004*. LNCS, vol. 3340, pp. 27–29. Springer, Heidelberg (2004)
4. Ehrenfeucht, A., Rozenberg, G.: Reaction systems. *Fundamenta Informaticae* 75, 263–280 (2007)
5. Salomaa, A.: Functions and sequences generated by reaction systems. *Theoretical Computer Science* 466, 87–96 (2012)

# Voting with a Logarithmic Number of Cards

Takaaki Mizuki, Isaac Kobina Asiedu, and Hideaki Sone




Cyberscience Center, Tohoku University,  
6-3 Aramaki-Aza-Aoba, Aoba, Sendai 980-8578, Japan  
tm-paper+ucnc2013@g-mail.tohoku-university.jp

**Abstract.** Consider an election where there are two candidates and several voters. Such an election usually requires the same number of ballot papers as the number of voters. In this paper, we show that such an election can be conducted using only a logarithmic number of cards with two suits—black and red—with identical backs. That is, we can securely compute the summation of a number of inputs (0s and 1s) using a logarithmic number of cards with respect to the number of inputs.

## 1 Introduction

Assume that there are  $n$  players  $P_1, P_2, \dots, P_n$  wishing to evaluate a function  $f(x_1, x_2, \dots, x_n)$ , where  $x_i$  is a secret value provided by player  $P_i$ . The goal of *secure computation* is to ensure the privacy of the players' inputs while guaranteeing the correctness of the computation. In the cryptography community, a considerable amount of research has been devoted to the problem of secure computation since the seminal research of Yao [13]; comprehensive surveys appear in [5,11].

The above-mentioned problem typically arises during elections. For example, assume that there are two candidates, and that each of  $n$  players chooses one of the candidates in a vote. Then, player  $P_i$ 's ballot can be regarded as input  $x_i \in \{0, 1\}$ , i.e.,  $x_i = 0$  and  $x_i = 1$  mean that the player selects the first and second candidate, respectively. Based on this interpretation, it suffices to compute the value of  $f(x_1, x_2, \dots, x_n) = \sum_{i=1}^n x_i$  without revealing the secret inputs. This can be done through the use of some cryptographic protocol. However, even in the digital era, such elections are often conducted with physical ballot papers, without any cryptographic protocols implemented in computers. Thus, the importance of physical implementations of voting can not be dismissed (e.g., [1,9,10]).

An election using ballot papers usually requires  $n$  sheets of paper (where  $n$  is the number of voters). In contrast, in this paper we show that such elections can be conducted using a logarithmic number of cards with two suits—black () and red ()—with identical backs (). That is, we can securely compute the summation of  $n$  inputs of 0s and 1s with  $O(\log n)$  cards.

### 1.1 Computation Using a Deck of Cards

As mentioned above, this paper deals with secure computation using a deck of cards. Here, we present some notations and introduce the properties of cards to be used.

The alphabet of cards is  $\clubsuit$ ,  $\heartsuit$ , and  $?$ , where  $?$  represents a card with its face down; both  $\clubsuit$  and  $\heartsuit$  are represented with  $?$  when facing down. The binary coding used for computation in this paper is as follows:

$$\clubsuit\heartsuit = 0, \heartsuit\clubsuit = 1. \tag{1}$$

Given a bit  $x \in \{0, 1\}$ , a pair of face-down cards  $??$  whose value is equal to  $x$  (according to the encoding rule (1) above) is called a *commitment to  $x$* , and is expressed as

$$\underbrace{??}_x.$$

The cards can also be manipulated as follows.

- Face up:

$$? \rightarrow \clubsuit, ? \rightarrow \heartsuit.$$

- Face down:

$$\clubsuit \rightarrow ?, \heartsuit \rightarrow ?.$$

- Swap:

$$\underbrace{??}_x \xrightarrow{\equiv} \overbrace{??}^{\equiv} \rightarrow \underbrace{??}_{\bar{x}}.$$

Note that swapping the two cards constituting a commitment to a bit  $x$  results in a commitment to negation  $\bar{x}$ . As seen later, some shuffling operations are used in addition to these manipulations, in the literature as well as in our protocols constructed in this paper.

### 1.2 History of Card-Based Protocols

Several *card-based protocols* for secure computation have been proposed (Table 1). All of these protocols take two commitments to bits  $a, b \in \{0, 1\}$  as input, and produce output  $a \wedge b$  of the AND function, or  $a \oplus b$  of the XOR function.

There are two types of protocols with regard to output format. While the first two protocols in Table 1 produce their output (the value of  $a \wedge b$ ) publicly, the remaining seven protocols produce their output in a *committed format*, i.e., their output is described as a sequence such as

$$\underbrace{??}_{a \wedge b}$$

that follows the encoding rule (1).

In addition to the protocols listed in Table 1, there are copy protocols, one of which is introduced later in Section 2.4.



**Table 1.** Known card-based protocols for secure computation

◦ *Secure AND in a non-committed format*

	# of colors	# of cards	Avg. # of trials
den Boer [2]	2	5	1
Mizuki-Kumamoto-Sone [7]	2	4	1

◦ *Secure AND in a committed format*

	# of colors	# of cards	Avg. # of trials
Crépeau-Kilian [4]	4	10	6
Niemi-Renvall [10]	2	12	2.5
Stiglic [12]	2	8	2
Mizuki-Sone [6] (§2.2)	2	6	1

◦ *Secure XOR in a committed format*

	# of colors	# of cards	Avg. # of trials
Crépeau-Kilian [4]	4	14	6
Mizuki-Uchiike-Sone [8]	2	10	2
Mizuki-Sone [6] (§2.3)	2	4	1

### 1.3 Our Results

As mentioned before, we wish to implement voting in the case of two candidates. If we distribute two cards of different suits to each voter  $P_i$ , then the voters can privately commit their ballots  $x_i$  to commitments in the form of

$$\underbrace{\boxed{?} \boxed{?}}_{x_1} \quad \underbrace{\boxed{?} \boxed{?}}_{x_2} \quad \dots \quad \underbrace{\boxed{?} \boxed{?}}_{x_n}.$$

Given such a sequence of  $n$  secret ballots, the simplest way to achieve the goal is to reveal the  $n$  cards after collecting and shuffling all left cards from each commitment, so that we can obtain only the number of ballots evaluating to 1 (and consequently the number of ballots evaluating to 0) by counting the number of red cards. In this case, because each voter is given a pair of cards to choose one to vote with,  $2n$  cards are required in total.

In this paper, we present card-based cryptographic protocols which can conduct such elections using fewer than  $2n$  cards. Specifically, we first review the most efficient protocols currently known in Section 2; then in Section 3, combining the existing protocols, we construct a summation protocol with  $2\lceil \log n \rceil + 8$  cards (where the logarithm base is 2 throughout this paper); in Section 4, we devise a new efficient half adder protocol, which improves summation computation in a way that it can be conducted with  $2\lceil \log n \rceil + 6$  cards; and finally, this paper is concluded in Section 5.

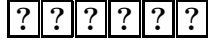
## 2 Known Protocols

In this section, after explaining the concept of a “random bisection cut,” which is a kind of shuffling operation, we introduce the existing AND, XOR and copy protocols [6] that are the most efficient ones currently known. (See Table 1 again.)

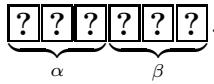
### 2.1 Random Bisection Cuts

A random bisection cut [6] is a type of shuffle operation, where a deck of cards is bisected and shifted randomly. We demonstrate the operation by taking six cards as an example.

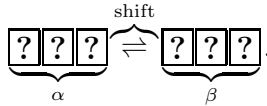
1. Assume that there are six cards as follows:



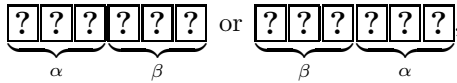
2. Bisect the deck of cards, and let the two sections be  $\alpha$  and  $\beta$ :



3. Shift  $\alpha$  and  $\beta$  randomly:

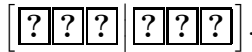


4. After applying such a random shift, the cards are either in their initial state or in a shifted state, as follows:





each of which occurs with probability of exactly 1/2.

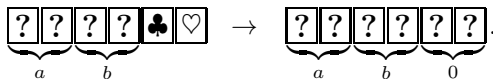
This kind of shuffling is referred to as a *random bisection cut* denoted by  $[\cdot|\cdot]$ . Below is the expression of a random bisection cut for six cards:



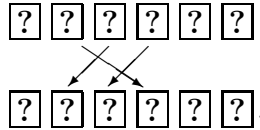
### 2.2 Six-Card AND Protocol

Using the random bisection cut, we can construct a six-card AND protocol [6], which can securely compute the function  $f(a,b) = a \wedge b$  with a total of six cards, namely three s and three s. The procedure of the AND protocol is as follows.

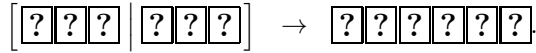
1. Arrange the six cards as below, and then turn over the two face-up cards:



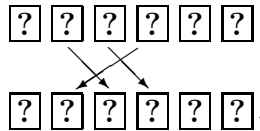
2. Rearrange the sequence of six cards as follows:



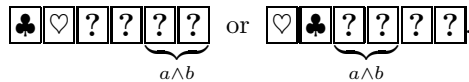
3. Apply a random bisection cut:



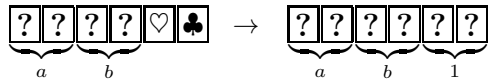
4. Rearrange the sequence of six cards as follows:



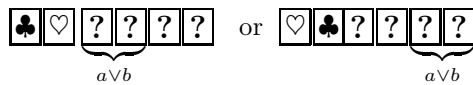
5. Reveal the first and second cards. Then, a commitment to  $a \wedge b$  is obtained as follows:



A six-card OR protocol can be easily constructed in a similar manner, that is, starting from



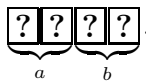
we can similarly obtain



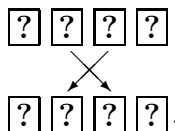
### 2.3 Four-Card XOR Protocol

We can also construct a four-card XOR protocol [6].

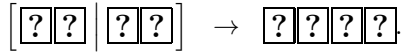
1. Arrange two commitments:



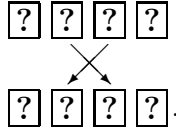
2. Rearrange the sequence of four cards as follows:



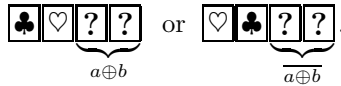
3. Apply a random bisection cut:



4. Rearrange the sequence of four cards as follows:



5. Reveal the first and second cards. Then, a commitment to  $a \oplus b$  (or  $\overline{a \oplus b}$ ) is obtained as follows:

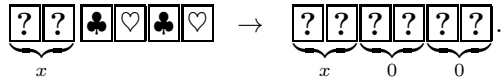


Recall that the NOT operation can be accomplished fairly easily by simply swapping the two face-down cards.

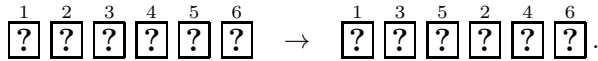
### 2.4 Copy Protocol with a Random Bisection Cut

Given a commitment to a bit  $x$ , four additional cards are sufficient to make a copy of the commitment  $[6]$ , as follows.

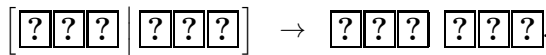
1. Arrange the four additional cards to the right of the given commitment:



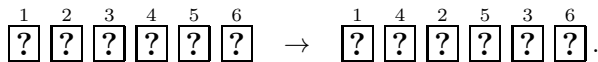
2. Rearrange the sequence of six cards as follows:



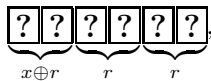
3. Apply a random bisection cut:



4. Rearrange the sequence of six cards as follows:

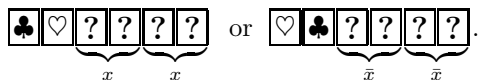


Then, we have



where  $r$  is a random bit because of the random bisection cut.

5. Reveal the first and second cards. Then, we have



Carrying this idea further,  $2k + 2$  additional cards are sufficient to make  $k$  copies of a commitment to  $x$   $[6]$ .

### 3 Voting with a Logarithmic Number of Cards

In this section, we show that by utilizing a half adder, the number of cards used for voting can be reduced from  $2n$  to  $2\lceil \log n \rceil + 8$ .

A half adder is a logical circuit that performs an addition operation on two binary numbers (of length 1). The half adder produces a sum and a carry value which are both binary numbers. The sum  $s$  is an XOR function of inputs  $a$  and  $b$ :

$$s = a \oplus b.$$

The carry  $c$  is an AND function of inputs  $a$  and  $b$ :

$$c = a \wedge b.$$

By combining the existing AND, XOR, and copy protocols introduced in the previous section, we can easily construct a half adder protocol, which is shown in Section 3.1. Based on the half adder, we can perform secure voting with  $2\lceil \log n \rceil + 8$  cards as shown in Section 3.2.

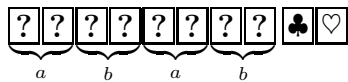
#### 3.1 Computing the Half Adder Using the Existing Protocols

Given two commitments to  $a$  and  $b$ , the following procedure securely computes the half adder with six additional cards.

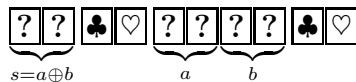
1. Place six cards next to the two given commitments as follows:



2. Use the six additional cards to copy inputs  $a$  and  $b$  with the copy protocol shown in Section 2.4. After the copy operation, the state of all 10 cards is as follows:



3. Use the first two commitments (to  $a$  and  $b$ ) to produce a commitment to  $a \oplus b$  with the XOR protocol mentioned in Section 2.3. Then, we obtain the sum  $s = a \oplus b$  of the half adder:



4. Use the rightmost three commitments to produce a commitment to  $a \wedge b$  with the AND protocol mentioned in Section 2.2. Then, we obtain the carry  $c = a \wedge b$  of the half adder (after some rearrangements):



### 3.2 Voting with the Half Adder

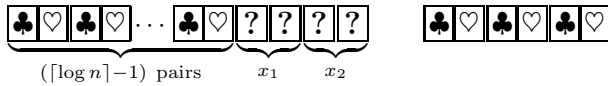
By using the half adder protocol presented above, the summation  $\sum_{i=1}^n x_i$  can be securely computed with  $2\lceil \log n \rceil + 8$  cards.

- Place  $\lceil \log n \rceil + 1$  pairs of different cards together with six additional cards as follows:

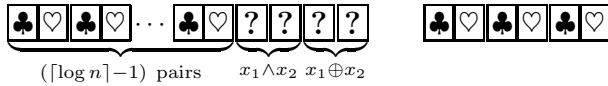


where we use the left deck to represent the outcome of the summation (more specifically,  $\lceil \log n \rceil$  pairs are sufficient to represent any integer between 0 and  $n - 1$ , and one more pair is needed for the  $n$ -th commitment to  $x_n$ ).

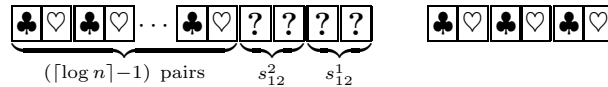
- Let the first two voters  $P_1$  and  $P_2$  make commitments to  $x_1$  and  $x_2$  using four cards from the left deck:



- Apply the half adder (explained in the previous subsection) to the rightmost 10 cards. Then, we have

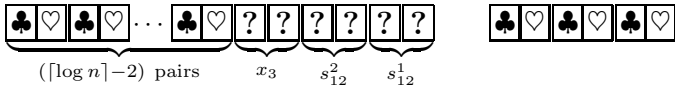


which can be rewritten as

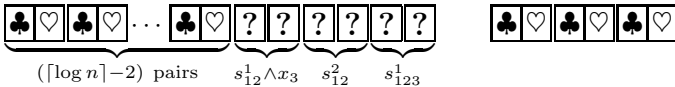


where  $s_{12}^1$  represents the first digit in  $x_1 + x_2$ , and  $s_{12}^2$  represents the second digit in  $x_1 + x_2$ .

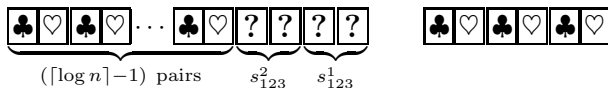
- Next, after the third player  $P_3$  makes a commitment:



apply the half adder to  $s_{12}^1$  and  $x_3$  to arrive at



To obtain  $s_{123}^2$ , apply the XOR protocol to  $(s_{12}^1 \wedge x_3)$  and  $s_{12}^2$ :



- Repeat in this way until we obtain  $s_{12\dots n}^{\lceil \log(n+1) \rceil}$ .

By doing this, we can obtain a sequence of commitments to the binary representation of the value of  $\sum_{i=1}^n x_i$ , namely the total number of votes for the candidate represented by  $\boxed{\heartsuit} \boxed{\clubsuit} = 1$ . The total number of cards with the use of this naive half adder protocol amounts to  $2\lceil \log n \rceil + 8$ .

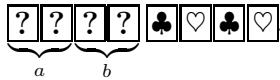
## 4 New Adder Protocols

In this section, we propose an efficient half adder protocol that requires two fewer cards than the naive protocol presented above. By using this improved half adder protocol, the number of cards for voting can be reduced to  $2\lceil \log n \rceil + 6$ . Furthermore, we also design an efficient full adder protocol.

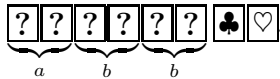
### 4.1 An Improved Half Adder Protocol

Recall that the naive half adder protocol presented in Section 3.1 requires six additional cards (other than the two input commitments). In contrast, our new half adder protocol requires only four additional cards. Moreover, it needs to copy only one input commitment. The procedure for computation is given below.

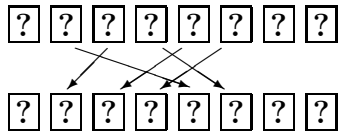
1. Arrange the input commitments and the four additional cards as follows:



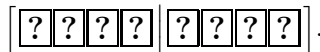
2. Use the four additional cards to copy input  $b$  with the copy protocol mentioned in Section 2.4. After the copy operation, the state of the cards is as follows:



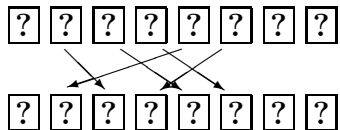
3. Let all cards face down and rearrange their order:



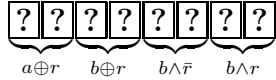
4. Apply a random bisection cut to the cards:



5. Rearrange the order again:



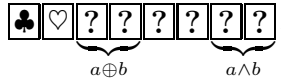
Then, we have



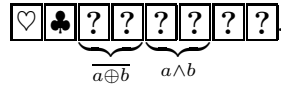
where  $r$  is a random bit because of the random bisection cut.

6. Reveal the first and second cards.

(a) If they are  $\heartsuit \spadesuit$ , then  $a \oplus r = 0$ , that is,  $a = r$ , and hence the output of the half adder is:



(b) If they are  $\heartsuit \clubsuit$ , then  $a \oplus r = 1$ , that is,  $a = \bar{r}$ , and hence the output of the half adder is:



Note that revealing the two cards in step 6 does not leak any information about  $a$  and  $b$  because  $r$  is a random bit.

Obviously, by replacing the naive half adder protocol with the improved one in the summation protocol given in Section 3.2, we can securely compute  $\sum_{i=1}^n x_i$  using only  $2\lceil \log n \rceil + 6$  cards.

### 4.2 Computing the Full Adder

Here, we propose a full adder protocol, which performs an addition operation on three bits. Given two input bits  $a$  and  $b$  and a carry  $c$ , the full adder produces the sum

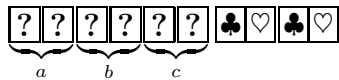
$$s' = (a \oplus b) \oplus c$$

and the carry value

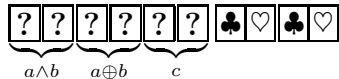
$$c' = (a \wedge b) \vee ((a \oplus b) \wedge c).$$

Using four additional cards, the full adder can be securely computed as follows.

1. Place three commitments and four additional cards as follows:

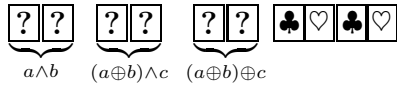


2. Use the four additional cards to apply the improved half adder protocol (presented in Section 4.1) to  $a$  and  $b$ : after that, the state of the cards is as follows:

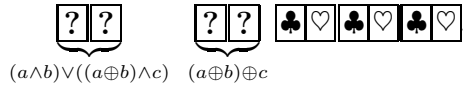




3. Use the four additional cards to apply the half adder to  $(a \oplus b)$  and  $c$ . Then, we have



4. Finally, compute the OR function of  $a \wedge b$  and  $(a \oplus b) \wedge c$  using two additional cards (as shown in Section 2.2). Then, we obtain the carry of the full adder:



The full adder protocol enables us to securely add any two (binary) numbers. For example, we can divide the voters into two groups, let each group execute the summation protocol, and combine the two outputs by using the full adder to obtain the total number of votes.

## 5 Conclusion

In this paper, we explored a new use of card-based cryptographic protocols and devised improved half adder and full adder protocols. The half adder protocol can be used for voting. Whereas conventional voting schemes require  $2n$  cards, the protocol presented here can reduce the number of cards required for voting to  $2\lceil \log n \rceil + 6$ .

In addition to practical applications such as elections, research on card-based protocols along with other physically implemented cryptographic protocols (e.g., [1,3,9]) can aid professional cryptographers in providing intuitive explanations to the general public about the nature of the cryptographic protocols they have constructed or about cryptography in general. We also believe that it will help researchers and teachers with cryptography-related education in the classroom.

**Acknowledgments.** We thank the anonymous referees whose comments helped us improve the presentation of the paper. This work was supported by JSPS KAKENHI Grant Number 23700007.

## References

- Balogh, J., Csirik, J.A., Ishai, Y., Kushilevitz, E.: Private computation using a PEZ dispenser. *Theoretical Computer Science* 306, 69–84 (2003)
- den Boer, B.: More efficient match-making and satisfiability: the five card trick. In: Quisquater, J.-J., Vandewalle, J. (eds.) *EUROCRYPT 1989*. LNCS, vol. 434, pp. 208–217. Springer, Heidelberg (1990)
- Fagin, R., Naor, M., Winkler, P.: Comparing information without leaking it. *Communications of the ACM* 39(5), 77–85 (1996)
- Crépeau, C., Kilian, J.: Discreet solitary games. In: Stinson, D.R. (ed.) *CRYPTO 1993*. LNCS, vol. 773, pp. 319–330. Springer, Heidelberg (1994)

5. Goldreich, O.: Foundations of Cryptography II: Basic Applications. Cambridge University Press, Cambridge (2004)
6. Mizuki, T., Sone, H.: Six-card secure AND and four-card secure XOR. In: Deng, X., Hopcroft, J.E., Xue, J. (eds.) FAW 2009. LNCS, vol. 5598, pp. 358–369. Springer, Heidelberg (2009)
7. Mizuki, T., Kumamoto, M., Sone, H.: The five-card trick can be done with four cards. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 598–606. Springer, Heidelberg (2012)
8. Mizuki, T., Uchiike, F., Sone, H.: Securely computing XOR with 10 cards. Australasian Journal of Combinatorics 36, 279–293 (2006)
9. Moran, T., Naor, M.: Polling with physical envelopes: A rigorous analysis of a human-centric protocol. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 88–108. Springer, Heidelberg (2006)
10. Niemi, V., Renvall, A.: Secure multiparty computations without computers. Theoretical Computer Science 191, 173–183 (1998)
11. Schneider, T.: Engineering Secure Two-Party Computation Protocols. Springer, Heidelberg (2012)
12. Stiglic, A.: Computations with a deck of cards. Theoretical Computer Science 259, 671–678 (2001)
13. Yao, A.: Protocols for secure computations. In: Proceedings of the 23th IEEE Symposium on Foundations of Computer Science, FOCS 1982, pp. 160–164 (1982)

# Asynchronous Signal Passing for Tile Self-assembly: Fuel Efficient Computation and Efficient Assembly of Shapes

Jennifer E. Padilla<sup>1</sup>, Matthew J. Patitz<sup>2</sup>, Raul Pena<sup>3</sup>, Robert T. Schweller<sup>3</sup>, Nadrian C. Seeman<sup>1</sup>, Robert Sheline<sup>3</sup>, Scott M. Summers<sup>4</sup>, and Kingsi Zhong<sup>3</sup>

<sup>1</sup> Department of Chemistry, New York University, New York, NY 10003, USA  
{jp164,ned.seeman}@nyu.edu

<sup>2</sup> Department of Computer Science and Computer Engineering,  
University of Arkansas, Fayetteville, AR 72701, USA  
mpatitz@self-assembly.net

<sup>3</sup> Department of Computer Science, University of Texas – Pan American, Edinburg,  
TX 78539, USA

{nb-raul,rtschweller,b.sheline,zhongxingsi}@utpa.edu

<sup>4</sup> Department of Computer Science and Software Engineering,  
University of Wisconsin – Platteville, Platteville, WI 53818, USA  
summerss@uwplatt.edu

**Abstract.** In this paper we demonstrate the power of a model of tile self-assembly based on active glues which can dynamically change state. We formulate the Signal-passing Tile Assembly Model (STAM), based on the model of Padilla, et al. [1] to be asynchronous, allowing any action of turning a glue on or off, attaching a new tile, or breaking apart an assembly to happen in any order. Within this highly generalized model we provide three new solutions to tile self-assembly problems that have been addressed within the abstract Tile Assembly Model and its variants, showing that signal passing tiles allow for substantial improvement across multiple complexity metrics. Our first result utilizes a recursive assembly process to achieve *tile-type efficient* assembly of linear structures, using provably fewer tile types than what is possible in standard tile assembly models. Our second system of signal-passing tiles simulates any Turing machine with high *fuel efficiency* by using only a constant number of tiles per computation step. Our third system assembles the discrete Sierpinski triangle, demonstrating that this pattern can be *strictly self-assembled* within the STAM. This result is of particular interest in that it is known that this pattern cannot self-assemble within a number of well studied tile self-assembly models. Notably, all of our constructions are at temperature 1, further demonstrating that signal-passing confers the power to bypass many restrictions found in standard tile assembly models.

## 1 Introduction

The abstract Tile Assembly Model (aTAM) [2] created a paradigm for computation to be carried out by a physical assembly process that captured the essence of

the Wang tiling model [3]. Turing machine simulation within the aTAM demonstrated its capacity for universal computation, and many subsequent assembly programs shifted focus to patterns, shapes and structures as the output of tile computation [4–7]. Many modifications to the standard aTAM have been investigated, including several variants that capture the notion of hierarchical assembly [1, 8–12]. Previous work introduced the notion of using signaled glue activation [1, 13], in particular to guide hierarchical assembly, enabling recursive self assembly [1]. Here we develop a general model of signaled tile self-assembly to enrich the tile assembly paradigm with greater capabilities that anticipate advancing techniques in the field of DNA nanotechnology and allow tile assembly to more closely emulate biological and natural processes.

Signaled glue activation was introduced in [1] for the purpose of establishing communication inside an assembly so that by activating glues at the exterior of the assembly it may take on new identities or roles. Interactions between assemblies, as described in hierarchical models such as the 2-HAM [8, 14], can simulate the interactions of individual tiles, coordinated in the STAM by the introduction of signals. In particular, recursive assembly results when supertiles simulate the original tiles of the tile set [1], a strategy outlined here in section 4 in a scheme for efficient line assembly.

Cooperativity, where more than one tile must be in place to determine which tile may be added next, is an important aspect of tile assembly systems. A binding threshold is included in the aTAM, given as the temperature,  $\tau$  of the system. Tiles can bind only if the sum of glue interactions at a particular site meet or exceed  $\tau$ , thus a system at temperature 2 readily includes cooperativity, whereas it is not as readily achieved at temperature 1. In the STAM, cooperativity can occur at temperature 1 via a query process, where a tile binds to an assembly at one edge, then queries a neighboring tile by turning on a set of glues. Information about the neighboring tile is gained based on which of these glues binds to its match. This cooperative effect differs from the aTAM in that STAM tiles may also respond to the identities and binding events of more distant tiles, enabling the constructions given in this paper to operate at temperature 1. Though the constructions here do not demonstrate a full simulation of the aTAM at temperature 2 by the STAM at temperature 1, the results here are significant given the known and conjectured limitations to temperature 1 computation in the aTAM and its variants [15, 16].

We expect glue deactivation to be as easy to implement as glue activation based on plausible DNA strand displacement mechanisms. Therefore, we utilize this new capability to design a Turing Machine that is fuel efficient (Section 5), and to implement the strict self-assembly of the Sierpinski triangle (Section 6). While on first consideration, glue deactivation might be thought to be on par with negative glues, it never requires repulsive forces between tiles, a necessity for negative glues that to our knowledge has yet to be implemented. Glue deactivation serves here to produce a fuel efficient Turing Machine that does not need to copy the state of unchanged positions on the tape. A halting computation produces an output tape, not a transcript of the computation as in the

traditional aTAM simulation of a Turing Machine. Strict self assembly of the Sierpinski triangle is achieved by releasing tiles that are not part of the target structure.

The addition of signaled glue activation and deactivation to tile assembly brings it one step closer to emulating biological processes of self assembly, where communication within a developing and growing living organism are crucial to its survival and success. In this construction, it becomes easier to view the Turing Machine plus its tape as a developing entity, that by following its input instructions, much as a cell follows its genetic recipe, goes through a metamorphosis and emerges from a halting computation as a new entity. The asynchronous growth process of the strict Sierpinski triangle in this model resembles the growth of something such as coral, where the “living” functional part of the system inhabits the growing frontier of the structure, laying down an enduring structure before dying and being washed away. The constructions presented in this paper demonstrate not only a more efficient use of materials (Table 1) in certain cases, but also serve to make the model more relevant in a biological context. The STAM anticipates the increasing sophistication of molecular computation systems, as described in the next section.

**Table 1.** Summary of our Results in the context of previous work in the field. In the Turing machine results,  $Q$  is the number of states of the Turing machine being simulated and  $S_i$  is the length of the tape at step  $i$  in the computation.

$n \times 1$ Lines	Tiles	Signals	Temperature	Glue Activation	Glue Deactivation
aTAM	n	-	1	-	-
STAM (Thm.1)	$O(1)$	$O(\log n)$	1	Yes.	No.

Turing Machine	Space	Fuel	Temp	Tiles	Signals	Glue Act.	Glue Deact.
aTAM ([4])	$O(\sum S_i)$	$O(S_i)$	2	$O(Q)$	-	-	-
3D aTAM ([15])	$O(\sum S_i)$	$O(S_i)$	1	$O(Q)$	-	-	-
Negative Glues ([17])	$O(S_i)$	$O(S_i)$	2	$O(Q)$	-	-	-
Negative Glues ([18])	$O(S_i)$	$O(1)$	8	$O(Q)$	-	-	-
STAM (Thm.2)	$O(S_i)$	$O(1)$	1	$O(Q)$	$O(1)$	Yes.	Yes.

Sierpinski Triangle	Strict	Tiles	Signals	Temp	Scale	Glue Act.	Glue Deact.
aTAM ([5])	No.	7	-	2	1	-	-
STAM (Thm.4)	Yes.	19	5	1	2	Yes.	Yes.
STAM (Thm.3)	No.	5	4	1	1	Yes.	No.

## 2 Physical Basis for the Model

The generalized model presented here has been designed to take into consideration a DNA implementation of all aspects of signaled assembly. We envision a physical implementation where Watson-Crick DNA base pairing provides specific glue interactions as it has done before for DNA implementations of the

aTAM [2, 5]. Additionally, we suggest that toehold-mediated DNA strand displacement reactions [19] may be the basis for the new elements of our model: binding-induced signaling, and glue activation or deactivation. The physical body of a tile might be composed entirely from a DNA origami structure [20, 21] in order to provide more room for signal pathways than the smaller DNA structures that have been used to implement the passive tiles of the aTAM [22]. Many known and tested DNA strand exchange mechanisms [23–25], cascades [26], and walkers [23, 27–29] suggest possibilities for implementing the signal pathway, including logic gates for responding to multiple inputs and transducers for ensuring that the activated glue can have a different sequence from the glues on the input edges [30]. Details of a plausible DNA origami tile construction are given in [1].

### 3 STAM Notation and Model

In this section we define the Signal-passing Tile Assembly Model (STAM), an extension of the 2-Handed Assembly Model (2HAM) [8, 9, 11, 12], which itself is an extension of the Tile Assembly Model (TAM) [2]. The STAM is a refined version of the model presented in [1], in which the basic tiles of the TAM are augmented with the ability to receive information, in the form of *signals*, from neighboring tiles in an assembly, and to pass signals to neighboring tiles. A very important feature of signals is that each signal can only move across any given tile one time - they are not reusable.

The STAM that we define is a highly generalized model which imposes minimal restrictions on aspects such as the speed of signals and orderings of events. This generalized version, while perhaps difficult to create well-behaved constructions within, provides a framework that is intended to be maximally independent of the specific details of potential physical implementations of actual signal tiles, such as those using mechanisms suggested in [1]. Valid constructions within this model, such as all of the constructions presented within the following sections of this paper, will therefore also work correctly within more restricted versions of the model (for instance, where signal timing or ordering can be guaranteed).

#### 3.1 Informal Description of the 2HAM

Since the STAM is an extension of the 2HAM, we now give a brief, informal description of the 2HAM.

A *tile type* is a unit square with four sides, each having a *glue* consisting of a *label* (a finite string) and *strength* (a positive integer value). We assume a finite set  $T$  of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile*. A *supertile* (a.k.a., *assembly*) is a positioning of tiles on the integer lattice  $\mathbb{Z}^2$ . Two adjacent tiles in a supertile *interact* if the glues on their abutting sides are equal (in both label and strength) and *bind* with that shared strength. Each supertile induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact and where the weight of that edge is the strength of their bond. The supertile is  $\tau$ -*stable* if every cut of its

binding graph has strength at least  $\tau$ . That is, the supertile is stable if at least energy  $\tau$  (i.e. a cut across bonds whose strengths sum to at least  $\tau$ ) is required to separate the supertile into two parts. A *tile assembly system* (TAS) is a pair  $\mathcal{T} = (T, \tau)$ , where  $T$  is a finite tile set (or more generally a finite set of supertiles) and  $\tau$  is the *temperature*, a parameter specifying the minimum binding energy required for a supertile to be stable. Given a TAS  $\mathcal{T} = (T, \tau)$ , a supertile is *producible* if either it is an element of  $T$ , or it is the  $\tau$ -stable result of translating two producible assemblies without overlap or rotation. A supertile  $\alpha$  is *terminal* if for every producible supertile  $\beta$ ,  $\alpha$  and  $\beta$  cannot be  $\tau$ -stably attached. A TAS is *directed* (a.k.a., *deterministic*, *confluent*) if it has only one terminal, producible supertile. Given a connected shape  $X \subseteq \mathbb{Z}^2$ , a TAS  $\mathcal{T}$  *strictly self-assembles*  $X$  (also *produces*  $X$  *uniquely*) if every producible, terminal supertile places tiles exactly on those positions in  $X$  (appropriately translated if necessary). Given a pattern  $Y \subseteq \mathbb{Z}^2$  (which must not necessarily be connected), a TAS  $\mathcal{T}$  *weakly self-assembles*  $Y$  if every producible, terminal supertile places a subset of tiles  $B \subseteq T$  exactly on those positions in  $Y$  (appropriately translated if necessary). Weak self-assembly can be thought of as using a subset of tile types to “paint a picture” of  $Y$  on a possibly larger canvas of tiles composing a terminal assembly.

### 3.2 High-Level Description of the STAM

In the STAM, tiles are allowed to have sets of glues on each edge (as opposed to only one glue per side as in the TAM and 2HAM). Tiles have an initial state in which each glue is either “**on**” or “**latent**” (i.e. can be switched **on** later). Tiles also each implement a transition function which is executed upon the binding of any glue on any edge of that tile. The transition function specifies a set of glues (along with the sides on which those glues are located) and an action for each: 1. turn that glue **on** (only valid if it is currently **latent**), or 2. turn that glue **off** (valid if it is currently **on** or **latent**). This means that glues can only be **on** once (although may remain so for an arbitrary amount of time or permanently), either by starting in that state or being switched **on** from **latent**, and if they are ever switched to **off** then no further transitions are allowed for that glue. This essentially provides a single “use” of a glue (and thus the implicit signal sent by its activation and binding). Note that turning a glue **off** breaks any bond that the glue may have formed with a neighboring tile. Also, since tile edges can have multiple active glues, when tile edges with multiple glues are adjacent, it is assumed that all glues in the **on** state bind (for a total binding strength equal to the sum of the strengths of the individually bound glues). The transition function defined for each tile type is allowed a unique set of output actions for the binding event of each glue along its edges, meaning that the binding of any particular glue on a tile’s edge can initiate a set of actions to turn an arbitrary set of the glues on the sides of the same tile either **on** or **off**. As the STAM is an extension of the 2HAM, binding and breaking can occur between tiles contained in pairs of arbitrarily sized supertiles. In order to allow for physical mechanisms which implement the transition functions of tiles but are arbitrarily slower or faster than the average rates of (super)tile attachments and detachments, rather than

immediately enacting the outputs of transition functions, each output action is put into a set of “pending actions” which includes all actions which have not yet been enacted for that glue (since it is technically possible for more than one action to have been initiated, but not yet enacted, for a particular glue).

An STAM system consists of a set of tiles and a temperature value. To define what is producible from such a system, we use a recursive definition of producible assemblies which starts with the initial tiles and includes any supertiles which can be formed by doing the following to any producible assembly: 1. executing any entry from the pending actions of any one glue within a tile within that supertile (and then that action is removed from the pending set), 2. binding with another supertile if they are able to form a  $\tau$ -stable supertile, or 3. breaking apart into two separate supertiles along a cut whose total strength is less than  $\tau$ .

As an overview, tiles in the STAM pass *signals* to neighboring tiles simply by activating glues which can bind with glues on adjacent edges of neighboring tiles. The information content of a signal is dependent upon the transition function of the receiving tile, that is, by what glue actions the receiving tile initiates upon the binding of its glue. By subsequently activating and deactivating its own glues, the receiving tile can propagate the signal to any of its neighbors. Solely by utilizing the mechanism of glue activation and deactivation initiated and carried out on individual tiles but chained together through series of glue bindings, a global network which is capable of passing information across entire assemblies (and also of allowing them to selectively enable sites for further growth or to discard arbitrary portions of the assembly), is created. However, an important restriction is the “fire once” nature of the signals, meaning that each glue can only transition to any given state once, and therefore the number of signals which a tile can process is a constant dependent upon the definition of the tile type.

The STAM, as formulated, is intended to provide a model based on experimentally plausible mechanisms for glue activation and deactivation, but to abstract them in a manner which is implementation independent. Therefore, no assumptions are made about the speed or ordering of the completion of signaling events (i.e. the execution of the transition functions which activate and deactivate glues and thus communicate with other tiles via binding events). This provides a highly asynchronous framework in which care must be made to guarantee desired results, but which then provides robust behavior independent of the actual parameters realized by a physical system. Furthermore, while the model allows for the placement of an arbitrary number of glues on each tile side and for each of them to signal an arbitrary number of glues on the same tile, this would obviously be limited in physical implementations. Therefore, each system can be defined to take into account a desired threshold for each of those parameters, not exceeding it for any given tile type, and we have also defined the notion of *signal complexity*, as the maximum number of glues on any side of any tile in a given set, to capture the complexity of a tile set.

Due to space constraints, a detailed definition of the STAM is omitted. Please refer to [31] for a significantly more detailed definition of the model.



## 4 Efficient Construction of Linear Assemblies

In the aTAM,  $n \times 1$  lines are inherently inefficient to self-assemble, requiring the worst possible tile complexity of  $n$ . However, using signal-passing tiles it is possible to create  $n \times 1$  lines using no more than 6 tile types, regardless of the value of  $n$ . Of course, the value of  $n$  must somehow be encoded in the system, but rather than requiring  $n$  tile types as in the aTAM, in the STAM the value of  $n$  can be efficiently encoded using  $\log n$  bits to require  $O(1)$  tile types with  $O(\log n)$  signal complexity. The construction we use makes use of a recursive doubling strategy where random tile binding events randomly assign the fate of each supertile at each stage, and is of independent interest in terms of using signal-passing tiles to perform recursive assembly of structures.

**Theorem 1.** *For every  $n \in \mathbb{N}$ , there exists an STAM system  $\mathcal{T} = (T, 1)$ , with  $|T| = O(1)$ , which uniquely assembles an  $n \times 1$  line. Moreover, the signal complexity of  $T$  is  $O(\log n)$  and  $\mathcal{T}$  does not use glue deactivation.*

## 5 Fuel Efficient Turing Machines

Showing that the original abstract Tile Assembly Model is computationally universal is a simple matter of designing a tile assembly system which can simulate a universal Turing machine, as originally shown in [2], and later expanded upon in [15, 17, 32, 33]. While displaying the computational power of the aTAM (and variants prior to the STAM), a common drawback of the constructions has been the number of tiles utilized during the formation of the assembly which simulates the computation, which, in this paper, is referred to as the fuel efficiency of the simulation.

For prior constructions, it has been necessary to make a new copy of the entire tape of the Turing machine between each computational step, with the new copy identical to the original except for the slight difference of a mere two tape cells indicating: 1. the output value in the tape cell left by the tape head, and 2. the tape cell marking the current location of the tape head. This full-scale copying of the tape, including the vast majority of cells which are unchanged, is wasteful in terms of the number of tiles required, experimentally very error prone due to the huge number of tile attachments required, and results in enormous assemblies. In this section, we exhibit a construction which is capable of simulating a universal Turing machine in the STAM, but while doing so only requires a small constant number of tiles (never more than 7) as fuel for each computational step and maintains an assembly which consists of a number of tiles which is only twice the total number of tape cells used by the Turing machine up to that step. It is possible that with significantly fewer binding events in the STAM construction than in those of previous models (even taking into account those used for signaling), it may be the case that the number of errors which occur could decrease, assuming, of course, that the mechanism which carries out the transition function is sufficiently error-free.

Throughout this paper, and without loss of generality, we define Turing machines as follows. Let  $M$  be an arbitrary single-tape Turing machine, such that  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$  with state set  $Q$ , input alphabet  $\Sigma = \{0, 1\}$ , tape alphabet  $\Gamma = \{0, 1, \_ \}$ , transition function  $\delta$ , start state  $q_0 \in Q$ , accept state  $q_{\text{accept}} \in Q$ , and reject state  $q_{\text{reject}} \in Q$ . Furthermore,  $M$  begins in state  $q_0$  on the leftmost cell of the tape, expects a one-way infinite-to-the-right tape, and is guaranteed to never attempt to move left while on the leftmost tape cell.

**Theorem 2 (Fuel efficient Turing machines).** *For any Turing machine  $M$  with input  $w \in \{0, 1\}^*$ , there exists an STAM system  $\mathcal{T}_{M(w)} = (T_{M(w)}, 1)$  with tile complexity  $O(|Q|)$ , signal complexity  $O(1)$ , and fuel efficiency  $O(1)$ , which simulates  $M$  on  $w$  in the following way:*

1.  $T_{M(w)}$  contains an active supertile consisting of  $2|w| + 2$  active tiles representing  $w$  and  $M$ 's start state.
2. If  $M$  halts on  $w$ , then  $\mathcal{A}_{\square}[T_M, 1]$  contains exactly one supertile with  $> 3$  tiles and that supertile contains exactly one ACCEPT (REJECT) tile if  $M$  accepts (rejects)  $w$ .
3. If  $M$  does not halt on  $w$ , then  $\mathcal{A}_{\square}[T_M, 1]$  contains exactly 0 (terminal) supertiles with  $> 3$  tiles.

Our proof of Theorem 2 is by construction. Here, we provide a brief overview.

Our construction works by utilizing a set of tile type templates that, along with the definition of a Turing machine  $M$ , are used to generate the set of active tiles which are specific to  $M$ . The construction uses a pair of tiles to represent each tape cell, with one tile representing the value (0, 1, or  $\_$ ) of that cell and one tile providing a ‘‘backbone’’ which the other attaches to and which also attaches to the backbone tiles of the cells to its left and right. Additionally, there is a special tile for the tape cell representing the rightmost end of the tape, and also, at any given time, exactly one tape cell which also represents one state of  $M$  along with the tape cell value. The location of the tape cell with that information denotes the location of  $M$ 's tape head at that point, and the value of the state tells what state  $M$  is in. Transitions of  $M$  occur in a series of 4 main steps in which tiles bind to the north of the tape cell denoting the head location, then to the north of the tape cell to the immediate left or right (depending on whether or not  $M$ 's transition function specifies a left or right moving transition from the current state while reading the current tape cell value), and along the way cause the dissociation of the tiles representing the tape cell values in both locations and their replacement with tiles which represent the correct output tape cell value of the transition and correctly record the new state and head location at the tape cell immediately to the left or right. Due to the asynchronous nature of glue deactivations, and also the necessity that any ‘‘junk’’ assemblies produced (i.e. those assemblies which break off from the assembly representing the Turing machine tape and which don't contribute to the final ‘‘answer’’) must not be able to attach to any portion of any supertile which represents any stage of the computation, junk assemblies are produced as size 2 or 3 so that any activated glues which would otherwise be able to bind to another supertile are hidden

between the tiles composing the junk assembly. In such a way,  $M(w)$  is correctly simulated while requiring only a constant number of new tiles per simulated transition step, and all junk assemblies remain inert and at size either 2 or 3. If  $M(w)$  halts, there will be one unique, terminal supertile which represents the result of that computation and is of size  $> 3$ . If  $M(w)$  does not halt, only the junk assemblies will be terminal.

## 6 Self-assembly of the Sierpinski Triangle

Discrete self-similar fractals are defined as sets of points in  $\mathbb{Z}^2$ , and consist of infinite, aperiodic patterns. It is difficult, if not impossible, for them to strictly self-assemble in the aTAM, as is shown in [34, 35] where the impossibility of a class of discrete self-similar fractals, including the Sierpinski triangle, strictly self-assembling in the aTAM is proven. The impossibility of strictly self-assembling the Sierpinski triangle in the 2HAM was shown in [11]. Additionally, Doty [36] has shown a generalization of the impossibility proof from [35] which applies to, among other things, scaled versions of the Sierpinski triangle for any scaling factor. Thus, any method of strictly self-assembling the Sierpinski triangle, scaled or not, is of interest.

In this section, we show that weak self-assembly of the Sierpinski triangle is possible in the STAM with fewer tile types (4 versus 7) and lower temperature (1 versus 2) than existing TAM constructions, and we also show that strict self-assembly at scale factor 2 is possible in the STAM at temperature 1, a first for any model at any temperature.

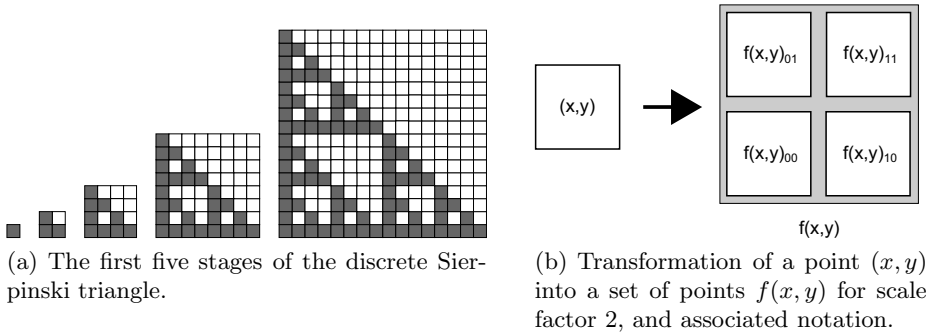
### 6.1 The Discrete Sierpinski Triangle

Here we use the definition of [34]. Let  $V = \{(1, 0), (0, 1)\}$ . Define the sets  $S_0, S_1, S_2, \dots \subset \mathbb{Z}^2$  by the recursion  $S_0 = \{(0, 0)\}$ ,  $S_{i+1} = S_i \cup (S_i + 2^i V)$ , where  $A + cB = \{\mathbf{m} + c\mathbf{n} \mid \mathbf{m} \in A \text{ and } \mathbf{n} \in B\}$ . Then the (standard) discrete Sierpinski triangle is the set  $S_\Delta = \cup_{i=0}^{\infty} S_i$ . See Figure 1a for a depiction of the first five stages (i.e.  $S_0$  through  $S_4$ ).

Our Sierpinski triangle constructions are as stated in the following two theorems. Additionally, in the next section we provide a high-level sketch of the more technically challenging construction for Theorem 4.

**Theorem 3.** *There exists an STAM system that weakly self-assembles the Sierpinski triangle. The system has 5 unique tiles, signal complexity = 4, assembles at temperature  $\tau = 1$ , and does not utilize glue deactivation.*

**Theorem 4.** *There exists an STAM system that strictly self-assembles the discrete Sierpinski triangle at temperature  $\tau = 1$ , with tile complexity = 19, scale factor = 2, signal complexity = 5, and which makes use of glue deactivation, producing terminal junk assemblies of size  $\leq 6$ .*



**Fig. 1.** The Sierpinski triangle and a description of the mapping for the scaled version.

## 6.2 Strict Self-assembly of the Sierpinski Triangle

Our proof of Theorem 4 is by construction. Here, we provide a brief overview.

Define  $f(x, y)$  as the function which takes as input a point  $(x, y)$  and which returns the set of 4 points which correspond to  $(x, y)$  at a scale factor of 2, that is, the  $2 \times 2$  square of points  $\{(2x + a, 2y + b) \mid a, b \in \{0, 1\}\}$ . (For instance,  $f(1, 1) = \{(2, 2), (2, 3), (3, 2), (3, 3)\}$ ). For notation, we will refer to the 4 points in the set  $f(x, y)$  as  $f(x, y)_{00}$ ,  $f(x, y)_{01}$ ,  $f(x, y)_{10}$ , and  $f(x, y)_{11}$  with subscripts corresponding to the values for  $a$  and  $b$ , given as 00, 01, 10, and 11, respectively. See Figure 1b for a clarification of this notation.

Let  $S_{2\Delta} = \{f(x, y) \mid (x, y) \in S_{\Delta}\}$  be the Sierpinski triangle at scale factor 2, i.e. where each point in the original Sierpinski triangle is replaced by a  $2 \times 2$  square of points, which we will refer to as a *block*. To prove Theorem 4, we now present an STAM system,  $\mathcal{T}_{2\Delta} = (T_{2\Delta}, 1)$  which strictly self-assembles  $S_{2\Delta}$ . At a high-level, it does so by weakly self-assembling  $S_{2\Delta}$  by treating each block  $f(x, y)$  as a single tile which receives one input each from the block to its south and the block to its west. Each input is either a 0 or 1, and the block performs the equivalent of an **xor** operation on those inputs and outputs the result to its north and east. A block  $f(x, y)$  which outputs a 1 corresponds to a point  $(x, y) \in S_{\Delta}$  and thus a location which must remain tiled in the final assembly (shown as grey locations in Figure 1a). A block  $f(x, y)$  which outputs a 0 instead corresponds to a point  $(x, y) \notin S_{\Delta}$  and must eventually be removed from the final assembly (shown as white locations in Figure 1a). Whenever a white region is completely tiled and completely surrounded by blocks corresponding to grey positions (note that all white regions in  $S_{\Delta}$  are surrounded by grey positions), glue deactivation is used to “eject” the blocks of that white region as a set of “junk” supertiles. Those junk supertiles are then broken down into constant sized terminal supertiles (of sizes 3, 4, and 6) which are unable to attach to any portion of the infinitely growing assembly, and thus remain inert junk assemblies.

**Acknowledgments.** The authors would like to thank Nataša Jonoska and Daria Karpenko for fruitful discussions and comments on this work. Research

was supported by National Science Foundation Grant CCF-1117210 to J.E.P. and N.C.S. and National Science Foundation Grant CCF-1117672 to M.J.P., R.P., R.T.S., R.S., and X.Z.

## References

1. Padilla, J.E., Liu, W., Seeman, N.C.: Hierarchical self assembly of patterns from the Robinson tilings: DNA tile design in an enhanced tile assembly model. *Natural Computing* 11, 323–338 (2012)
2. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Institute of Technology (June 1998)
3. Wang, H.: Proving theorems by pattern recognition II. *AT&T Bell Labs Tech. J.* 40, 1–41 (1961)
4. Rothmund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In: *STOC 2000: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, Portland, Oregon, United States, pp. 459–468. ACM (2000)
5. Rothmund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology* 2(12), 2041–2053 (2004)
6. Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes. *SIAM Journal on Computing* 36(6), 1544–1569 (2007)
7. Kao, M.-Y., Schweller, R.: Randomized self-assembly for approximate shapes. In: Aceto, L., Damgård, L., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I. LNCS*, vol. 5125, pp. 370–384. Springer, Heidelberg (2008)
8. Aggarwal, G., Cheng, Q., Goldwasser, M.H., Kao, M.-Y., Moisset de Espanés, P., Schweller, R.T.: Complexities for generalized models of self-assembly. *SIAM Journal on Computing* 34, 1493–1515 (2005)
9. Demaine, E.D., Demaine, M.L., Fekete, S.P., Ishaque, M., Rafalin, E., Schweller, R.T., Souvaine, D.L.: Staged self-assembly: nanomanufacture of arbitrary shapes with  $O(1)$  glues. *Natural Computing* 7(3), 347–370 (2008)
10. Becker, F.: Pictures worth a thousand tiles, a geometrical programming language for self-assembly. *Theoretical Computer Science* 410(16), 1495–1515 (2009)
11. Cannon, S., Demaine, E.D., Demaine, M.L., Eisenstat, S., Patitz, M.J., Schweller, R., Summers, S.M., Winslow, A.: Two hands are better than one (up to constant factors). Technical Report 1201.1650, Computing Research Repository (2012)
12. Chen, H.L., Doty, D.: Parallelism and time in hierarchical self-assembly. In: *SODA 2012: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1163–1182. SIAM (2012)
13. Majumder, U., LaBean, T.H., Reif, J.H.: Activatable tiles: Compact, robust programmable assembly and other applications. In: Garzon, M.H., Yan, H. (eds.) *DNA 2007. LNCS*, vol. 4848, pp. 15–25. Springer, Heidelberg (2008)
14. Adleman, L., Cheng, Q., Goel, A., Huang, M.D., Kempe, D., Moisset de Espanés, P., Rothmund, P.W.K.: Combinatorial optimization problems in self-assembly. In: *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, pp. 23–32 (2002)
15. Cook, M., Fu, Y., Schweller, R.: Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d. In: *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms* (2011)
16. Doty, D., Patitz, M.J., Summers, S.M.: Limitations of self-assembly at temperature 1. *Theoretical Computer Science* 412, 145–158 (2011)

17. Doty, D., Kari, L., Masson, B.: Negative interactions in irreversible self-assembly. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16. LNCS, vol. 6518, pp. 37–48. Springer, Heidelberg (2011)
18. Schweller, R., Sherman, M.: Fuel efficient computation in passive self-assembly. In: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana (to appear, 2013)
19. Yurke, B., Turberfield, A.J., Mills, A.P., Simmel, F.C., Neumann, J.L.: A DNA-fuelled molecular machine made of DNA. *Nature* 406(6796), 605–608 (2000)
20. Rothemund, P.W.K.: Folding DNA to create nanoscale shapes and patterns. *Nature* 440(7082), 297–302 (2006)
21. Liu, W., Zhong, H., Wang, R., Seeman, N.C.: Crystalline Two-Dimensional DNA-Origami arrays. *Angewandte Chemie International Edition* 50(1), 264–267 (2011)
22. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. *Nature* 394(6693), 539–544 (1998)
23. Yin, P., Choi, H.M.T., Calvert, C.R., Pierce, N.A.: Programming biomolecular self-assembly pathways. *Nature* 451(7176), 318–322 (2008)
24. Zhang, D.Y., Seelig, G.: Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry* 3(2), 103–113 (2011)
25. Qian, L., Winfree, E.: A simple dna gate motif for synthesizing large-scale circuits. *Journal of The Royal Society Interface* 8(62), 1281–1297 (2011)
26. Dirks, R.M., Pierce, N.A.: Triggered amplification by hybridization chain reaction. *Proceedings of the National Academy of Sciences of the United States of America* 101(43), 15275 (2004)
27. Omabegho, T., Sha, R., Seeman, N.C.: A bipedal DNA brownian motor with coordinated legs. *Science* 324(5923), 67 (2009)
28. Lund, K., Manzo, A.J., Dabby, N., Michelotti, N., Johnson-Buck, A., Nangreave, J., Taylor, S., Pei, R., Stojanovic, M.N., Walter, N.G., Winfree, E., Yan, H.: Molecular robots guided by prescriptive landscapes. *Nature* 465(7295), 206–210 (2010)
29. Wickham, S.F.J., Endo, M., Katsuda, Y., Hidaka, K., Bath, J., Sugiyama, H., Turberfield, A.J.: Direct observation of stepwise movement of a synthetic molecular transporter. *Nature Nanotechnology* 6(3), 166–169 (2011)
30. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. *Science* 314(5805), 1585 (2006)
31. Padilla, J.E., Patitz, M.J., Pena, R., Schweller, R.T., Seeman, N.C., Sheline, R., Summers, S.M., Zhong, X.: Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. Technical Report 1202.5012, Computing Research Repository (2012)
32. Patitz, M.J., Summers, S.M.: Self-assembly of decidable sets. *Natural Computing* 10, 853–877 (2011)
33. Demaine, E.D., Patitz, M.J., Schweller, R.T., Summers, S.M.: Self-Assembly of Arbitrary Shapes Using RNase Enzymes: Meeting the Kolmogorov Bound with Small Scale Factor (extended abstract). In: Schwentick, T., Dürr, C. (eds.) 28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, Dortmund, Germany, March 10–12. LIPIcs, vol. 9, pp. 201–212. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
34. Lathrop, J.I., Lutz, J.H., Summers, S.M.: Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science* 410, 384–405 (2009)
35. Patitz, M.J., Summers, S.M.: Self-assembly of discrete self-similar fractals. *Natural Computing* 9, 135–172 (2010)
36. Doty, D.: Personal communication (2012)

# Control Languages Associated with Tissue P Systems

Ajeesh Ramanujan and Kamala Krithivasan

Department of Computer Science and Engineering  
Indian Institute of Technology Madras, Chennai - 36  
ajeeshramanujan@gmail.com, kamala@iitm.ac.in

**Abstract.** We consider a way to associate a language with the computations of a tissue P system. We assign a label to every rule, where the labels are chosen from an alphabet or the label can be  $\lambda$ . The rules used in a transition should have either the empty label or the same label from the chosen alphabet. In this way, a string is associated with each halting computation, called the *control word* of the computation. The set of all control words associated with computations in a tP system form the *control language* of the system. We study the family of control languages of tP systems in comparison with the families of finite, regular, context-free, context-sensitive, and recursively enumerable languages.

**Keywords:** P systems, tP systems, regular languages, context-free languages, context-sensitive languages, recursively enumerable languages.

## 1 Introduction

P systems, introduced in [1], can be used as language generating/accepting devices in various ways – see, e.g., [2, 3, 5–10]. In all these papers (and several others), words are associated with the sequence of objects emitted by the system, taking all the permutations when more than one symbol is emitted. Another way to associate a string with a computation is by following the membrane *trace* [8, 11] of a specified object as it travels through the system: i.e., a symbol  $b_i$  is generated each time that the traveling object enters the membrane  $i$ .

In this paper we consider neural like-tissue P systems (shortly called tP systems) introduced in [4]. Tissue-like membrane structures are described by graphs, where membranes, also called cells, are nodes of a graph. Here, an edge between two nodes corresponds to a communication channel between cells placed in these nodes. Each cell also has a state which controls the evolution of the objects using rewriting rules of the form  $su \rightarrow s'u'$ , where  $s, s'$  are states, and  $u \rightarrow u'$  is a usual multiset rewriting rules, with target indications in  $u'$ , which direct the movement of objects from one cell to another. One of the cells is designated as the output cell, which also sends objects to the environment, providing in this way an output. When moved between cells, objects can also be replicated, and then copies are sent to all cells to which an edge (synapses) is available from the cell where the rule is applied.

In this paper, strings over arbitrary alphabets are obtained, in the form of control words associated with computations in a tP system whose rules are labeled. This idea was initially considered in [13] and further explored for spiking Neural P system in [14]. We extended the idea to transition P systems in [16]. We continue here this study for the case when tP systems are used.

In some sense, we may say that the control language associated with a tP system is *generated* by this system. Here we assign a label to every rule, where the labels are chosen from a finite alphabet or can be labeled with  $\lambda$ . All rules used in a computation step should have the same label, or they can also be labeled with  $\lambda$ . So in this case a transition is controlled by the label of the rule used. If a rule can be used, use it. Here we consider two cases: first by imposing one important restriction on the application of rules. In that case during a computation step, at least one rule must have a label other than  $\lambda$ . So the generated string has the same length as the number of steps used during computation. In the second case, we relax the restriction by allowing all the rules to have  $\lambda$  label in a single step of computation. A string is generated by the system if and only if the computation of the system halts.

In what follows we will use this terminology, calling the control languages *languages generated* by tP systems, in the sense specified above. Note however the essential difference between this notion and the languages defined by tP systems, in the “standard” sense – see references in [3, 4].

The difference between the case when  $\lambda$  moves (when only rules with the empty label are used) are allowed and the case when such moves are not allowed is essential: in the former case all recursively enumerable languages are generated, while in the latter case only a subset of context-sensitive languages are generated, without covering the family of context-free languages.

The paper is organized as follows. In Section 2, we provide the necessary automata theory prerequisites. In Section 3, we give the definition of a tissue P system as defined in [4]. In Section 4, we introduce and define the control language associated with tP system and we study the families of languages generated by this model.

## 2 Basic Definition

Let  $\Sigma$  be a finite set of symbols called an alphabet. A string  $w$  over  $\Sigma$  is a sequence of symbols from  $\Sigma$ .  $\lambda$  denotes the empty string. The set of all strings over  $\Sigma$  is denoted by  $\Sigma^*$ . The length of a string  $w \in \Sigma^*$  is denoted by  $|w|$ . A language  $L$  over  $\Sigma$  is a set of strings over  $\Sigma$ . The family of finite, regular, context-free, context-sensitive, and recursively enumerable languages are denoted by *FIN*, *REG*, *CF*, *CS*, and *RE* respectively.

The set of natural numbers is denoted by  $\mathbb{N}$ . A *multiset* over a set  $X$  is a mapping  $M : X \rightarrow \mathbb{N}$ ; for  $a \in X$ , we say that  $M(a)$  is the *multiplicity* of  $a$  in  $M$ . For  $Y \subseteq X$  and  $M$  over  $X$ , we define the *projection* of  $M$  on  $Y$  by

$$pr_Y(M)(a) = \begin{cases} M(a), & \text{if } a \in Y, \\ 0, & \text{otherwise.} \end{cases}$$



A *linearly bounded automaton* is a Turing machine restricted to the use of working space linearly bounded with respect to the length of the input. Formally, there exist non-negative integers  $a$  and  $b$  such that for any configuration represented by  $xq_iy$  such that  $q_0w \vdash_M^* xq_iy$  for some  $q_i \in Q$  and we have  $|xy| \leq a|w| + b$ .

A register machine is a construct  $M = (m, H, l_0, l_h, I)$ , where  $m$  is the number of registers,  $H$  is the set of instruction labels,  $l_0$  is the start label (labeling an *ADD* instruction),  $l_h$  is the halt label (assigned to instruction halt), and  $I$  is the set of instructions labeled in a one-to-one manner by the labels from  $H$ . The instructions are of the following forms:

- $l_i : (ADD(r), l_j)$  (add 1 to register  $r$  and then go to the instruction with label  $l_j$ ),
- $l_i : (SUB(r), l_j, l_k)$  (if register  $r$  is non-empty, then subtract 1 from it and go to the instruction with label  $l_j$ , otherwise go to the instruction with label  $l_k$ ),
- $l_h : HALT$  (the halt instruction).

A register machine  $M$  accepts a number  $n$  in the following way: we start with number  $n$  in a specified register  $r_0$  and all other registers being empty (i.e., storing the number 0), we first apply the instruction with label  $l_0$  and we proceed to apply instructions as indicated by the labels (and made possible by the contents of the registers); if we reach the halt instruction, then the number  $n$  is said to be accepted by  $M$ . The set of all numbers accepted by  $M$  is denoted by  $N(M)$ . It is known (see, e.g., [12]) that register machines (even with only three registers, but this detail is not relevant in what follows) accept all sets of numbers which are Turing computable.

### 3 Tissue P Systems (tP Systems)

**Definition 1.** A *tissue P system* [4] of degree  $m \geq 1$ , is a construct  $\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_{out})$ , where:

1.  $O$  is a finite non-empty alphabet (of objects);
2.  $syn \subseteq \{1, 2, \dots, m\} \times \{1, 2, \dots, m\}$  (synapses among cells);
3.  $i_{out} \in \{1, \dots, m\}$  indicates the output cell;
4.  $\sigma_1, \dots, \sigma_m$  are cells, of the form  $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i), 1 \leq i \leq m$ , where:
  - (a)  $Q_i$  is a finite set (of states);
  - (b)  $s_{i,0} \in Q_i$  is the initial state;
  - (c)  $w_{i,0} \in O^*$  in the initial multiset of objects;
  - (d)  $P_i$  is a finite set of rules of the form  $sw \rightarrow s'xy_{go}z_{out}$ , where  $s, s' \in Q_i, w, x \in O^*, y_{go} \in (O \times \{go\})^*$  and  $z_{out} \in (O \times \{out\})^*$  with the restriction that  $z_{out} = \lambda$  for all  $i \in \{1, 2, \dots, m\}$  different from  $i_{out}$ .

A tP system as above is said to be *cooperative* if it contains at least a rule  $sw \rightarrow s'w'$  such that  $|w| > 1$ , and *non-cooperative* in the opposite case.

Any  $m$ -tuple of the form  $(s_1w_1, \dots, s_mw_m)$ , with  $s_i \in Q_i$  and  $w_i \in O^*$ , for all  $1 \leq i \leq m$ , is called a *configuration* of  $\Pi$ ; thus  $(s_{1,0}w_{1,0}, \dots, s_{m,0}w_{m,0})$  is the *initial configuration* of  $\Pi$ .

Using the rules from the sets  $P_i, 1 \leq i \leq m$ , we can define *transitions* among the configurations of the system. We consider three modes of processing the objects and three modes of transmitting objects from a cell to another.

Let us denote  $O_{go} = \{(a, go) | a \in O\}$ ,  $O_{out} = \{(a, out) | a \in O\}$ , and  $O_{tot} = O \cup O_{go} \cup O_{out}$ . For a multiset over  $O_{tot}$ , we denote by  $go(w)$  the submultiset of symbols  $a \in O$ , appearing in  $w$  in the form  $(a, go)$ , and by  $out(w)$  the submultiset of symbols  $a \in O$  appearing in  $w$  in the form  $(a, out)$ . For a node  $\sigma_i$ , we denote  $anc(i) = \{j | (j, i) \in syn\}$  and  $succ(i) = \{j | (i, j) \in syn\}$ .

For  $s, s' \in Q_i, x \in O^*, y \in O_{tot}^*$ , we write  
 $sx \Rightarrow_{min} s'y$  iff  $sw \rightarrow s'w' \in P_i, w \subseteq x$ , and  $y = (x - w) \cup w'$ ,  
 $sx \Rightarrow_{par} s'y$  iff  $sw \rightarrow s'w' \in P_i, w^k \subseteq x, w^{k+1} \not\subseteq x$  for some  $k \geq 1$ , and  $y = (x - w^k) \cup w'^k$ ,  
 $sx \Rightarrow_{max} s'y$  iff  $sw_1 \rightarrow s'w'_1, \dots, sw_k \rightarrow s'w'_k \in P_i, k \geq 1$ , such that  $w_1 \dots w_k \subseteq x, y = (x - w_1 \dots w_k) \cup w'_1 \dots w'_k$ , and there is no  $sw \rightarrow s'w' \in P_i$  such that  $w_1 \dots w_k w \subseteq x$ .

In the first case, only one occurrence of the multiset from the left hand side of a rule is processed; in the second case a maximal change is performed with respect to a chosen rule, in the sense that as many as possible copies of the multiset from the left hand side of the rule are replaced by the corresponding number of copies of the multiset from the right hand side; in the third case a maximal change is performed with respect to all rules which use the current state of the cell and introduce the same new state after processing the objects.

We also write  $sx \Rightarrow_{\alpha} s'y$ , for  $\alpha \in \{min, par, max\}$ , if there is no rule  $sw \rightarrow s'w' \in P_i$  such that  $w \subseteq x$ . This denotes the case when a cell cannot process the current objects in a given state.

The symbols will be sent to the cells related by synapses to the cell  $\sigma_i$  where the rule  $sw \rightarrow s'w'$  is applied according to the following three modes:

- *repl* : each symbol  $a$ , for  $(a, go)$  appearing in  $w'$ , is sent to each of the cells  $\sigma_j$  such that  $(i, j) \in syn$ ;
- *one* : all symbols  $a$  appearing in  $w'$  in the form  $(a, go)$  are sent to one of the cells  $\sigma_j$  such that  $(i, j) \in syn$ , nondeterministically chosen;
- *spread* : the symbols appearing in  $w'$  in the form  $(a, go)$  are nondeterministically distributed among the cells  $\sigma_j$  such that  $(i, j) \in syn$ .

For two configurations  $C_1 = (s_1w_1, \dots, s_mw_m), C_2 = (s'_1w'_1, \dots, s'_mw'_m)$  we write  $C_1 \Rightarrow_{\alpha, \beta} C_2$ , for  $\alpha \in \{min, par, max\}, \beta \in \{repl, one, spread\}$ , if there are  $w'_1, \dots, w'_m$  in  $O_{tot}^*$  such that  $s_iw_i \Rightarrow s'_iw'_i, 1 \leq i \leq m$ , and

- for  $\beta = repl$ , we have,  $w''_i = pr_O(w'_i) \cup \bigcup_{j \in anc(i)} go(w'_j)$ ;
- for  $\beta = one$ , we have,  $w''_i = pr_O(w'_i) \cup \bigcup_{j \in I_i} go(w'_j)$ , where  $I_i$  is a subset of  $anc(i)$  such that the set  $anc(i)$  was partitioned into  $I_1, \dots, I_m$ ; at this transition, all non-empty sets of objects of the form  $\bigcup_{j \in I_k} go(w'_j), 1 \leq k \leq m$ , should be sent to receiving cells (added to multisets  $w''_l, 1 \leq l \leq m$ );

- for  $\beta = spread$ , we have,  $w_i'' = pr_O(w_i') \cup z_i$ , where  $z_i$  is a submultiset of the multiset  $\bigcup_{j \in anc(i)} go(w_j')$  such that  $z_1, \dots, z_m$  are multisets with the property  $\bigcup_{j=1}^m z_j = \bigcup_{j \in anc(i)} go(w_j')$ , and such that all  $z_1, \dots, z_m$  are sent to receiving cells (added to multisets  $w_l'', 1 \leq l \leq m$ ).

During any transition, if no rule is applicable in a cell, then the cell waits until new objects are sent to it from its ancestor cells. A global clock is assumed and the working of the system is synchronized.

A sequence of transitions among configurations of the tP system  $\Pi$  is called a *computation* of  $\Pi$ . A computation which ends in a configuration where no rule in no cell can be used, is called a *halting* computation.

## 4 Control Words of tP Systems

In this section we introduce and define control languages for tP systems and we consider the power of language generated with respect to a control word associated with a computation of a tP system.

Consider a tP system  $\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_{out})$  of degree  $m$ . Assume a total ordering on the rules. Let  $P = P_1 \cup P_2 \cup \dots \cup P_m$  and  $\Sigma$  be a finite alphabet. Define a function  $l : P \rightarrow \Sigma \cup \{\lambda\}$ , called the labeling function, that assigns a label to every rule in  $P$ . With such a system, we associate a language as follows.

Given two configurations  $C_1, C_2$  of  $\Pi$ , we consider only transition  $C_1 \Rightarrow_b C_2, b \in \Sigma$  between configurations which use only rules with the same label  $b$  and rules labeled with  $\lambda$ . We say that such a transition is *label restricted*.

With a label restricted transition we associate the symbol  $b$  if at least one rule with label  $b$  is used; if all used rules have the label  $\lambda$ , i.e. label unrestricted, then we associate  $\lambda$  to this transition. Thus, with any computation in  $\Pi$  starting from the initial configuration to a halting configuration we associate a (control) word. In the label restricted case, the number of steps in the computation is the same as the length of the control word as in each step at least one rule with a label from  $\Sigma$  is used. In the label unrestricted case, the length of the control word is less than or equal to the number of steps in the computation.

The language of control words associated with all label unrestricted halting computations in a tP system  $\Pi$  is denoted by  $L_{\alpha, \beta}^\lambda(\Pi), \alpha \in \{min, par, max\}, \beta \in \{repl, one, spread\}$ , in the mode  $(\alpha, \beta)$ . The superscript indicates the fact that  $\lambda$  steps (all rules applied in one step can have  $\lambda$  labels) are permitted. When only steps where at least one rule with a non-empty label is used, the generated language is denoted by  $L_{\alpha, \beta}(\Pi), \alpha \in \{min, par, max\}, \beta \in \{repl, one, spread\}$ , in the mode  $(\alpha, \beta)$ .

The family of languages  $L_{\alpha, \beta}(\Pi)$  associated with cooperative tP systems  $\Pi$  with at most  $m \geq 1$  cells, and each of them using at most  $r \geq 1$  states is denoted by  $LtP_{m, r}(Coo, \alpha, \beta)$ ; When non-cooperative tP systems are used we write  $LtP_{m, r}(nCoo, \alpha, \beta)$ . In the unrestricted case, the corresponding language family is denoted by  $L^\lambda LtP_{m, r}(Coo, \alpha, \beta)$ . When one (or both) of the parameters  $m, r$  is (are) not bounded, then we replace it (them) with  $*$ , thus obtaining families of the form  $LtP_{m, *}(Coo, \alpha, \beta), LtP_{*, r}(Coo, \alpha, \beta)$ , etc.

Note that, in this paper, output cell in the tP systems plays no role, hence they are omitted. Moreover, whenever possible, we associate directly the labels to rules, writing  $b : sw \rightarrow s'w'$  instead of writing  $l(sw \rightarrow s'w') = b$ .

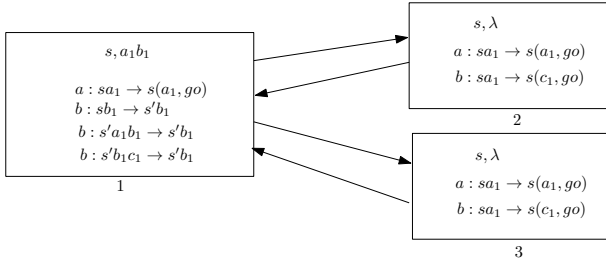
We also use the convention that two language processing devices – generating or accepting – are equivalent if they characterize languages which differ at most in the empty string (otherwise stated, the string  $\lambda$  is ignored when comparing two language processing devices).

Before investigating the power of tP systems with labels, let us consider an example, in order to clarify the definitions and to illustrate the working of our variant.

*Example 1.* Consider the tP system  $\Pi_1 = (\{a_1, b_1, c_1\}, \sigma_1, \sigma_2, \sigma_3, syn)$  with:

$$\begin{aligned} \sigma_1 &= (\{s, s'\}, s, a_1b_1, \{a : sa_1 \rightarrow s(a_1, go), b : sb_1 \rightarrow s'b_1, b : s'a_1b_1 \rightarrow s'b_1, \\ & b : s'b_1c_1 \rightarrow s'b_1\}), \\ \sigma_2 &= (\{s\}, s, \lambda, \{a : sa_1 \rightarrow s(a_1, go), b : sa_1 \rightarrow s(c_1, go)\}), \\ \sigma_3 &= (\{s\}, s, \lambda, \{a : sa_1 \rightarrow s(a_1, go), b : sa_1 \rightarrow s(c_1, go)\}). \\ syn &= \{(1, 2), (1, 3), (2, 1), (3, 1)\}. \end{aligned}$$

The tP system  $\Pi_1$  is graphically represented as in Figure 1, with rectangles representing cells (these rectangles contain the initial state, the initial multiset, and the set of labeled rules), with arrows indicating the synapses.



**Fig. 1.** A labeled tP system

The reader can easily verify that we have:

$$\begin{aligned} L_{min, repl}(\Pi_1) &= \{b^2, ab^3\} \cup \{a^n b^{n+1}, n \geq 2\}, \\ L_{min, \beta}(\Pi_1) &= \{a^n b^2, n \geq 0\}, \text{ for } \beta \in \{one, spread\}, \\ L_{par, repl}(\Pi_1) &= \{a^n b^{2^{\lfloor \frac{n+1}{2} \rfloor + 1}}, n \geq 0\}, \\ L_{par, \beta}(\Pi_1) &= \{a^n b^2, n \geq 0\}, \text{ for } \beta \in \{one, spread\}, \\ L_{max, repl}(\Pi_1) &= \{b^2, ab^3\} \cup \{a^n b^{n+1}, n \geq 2\}, \\ L_{max, \beta}(\Pi_1) &= \{a^n b^2, n \geq 0\}, \text{ for } \beta \in \{one, spread\}, \end{aligned}$$

In the replicative case, the symbols produced by the rule  $sa_1 \rightarrow s(a_1, go)$  from cell 1 are doubled. When the rules are used in the parallel mode, then all the

symbols present in the system are doubled from a step to next step, thereby obtaining power of 2 in the generated language. When the rules are used in the minimal mode, the symbols are processed one by one and we obtain a context-free language in the case of *repl* mode and we obtain a regular language in the case of *one* and *spread* modes. In the maximal mode, we can send copies of  $a_1$  at the same step to cells 2 and 3, hence we obtain the same language as in *min* mode.

Next we investigate the relationship between the family of languages generated by tP systems and the families of finite, regular, context-free, context-sensitive, and recursively enumerable languages.

**Theorem 1.**  $REG \subset LtP_{1,1}(nCoo, \alpha, \beta), \alpha \in \{min, par, max\}, \beta \in \{rep, one, spread\}$ .

*Proof.* Let  $G = (N, T, P, S)$  be a regular grammar that generates  $R$ . Assume that all productions are of the form  $A \rightarrow bB$  or  $A \rightarrow b$  or  $S \rightarrow \lambda$ , where  $A, B \in N, b \in T$ . Let  $v$  be the number of variables in  $G$ . Rename the variables as  $A_i, 1 \leq i \leq v$ , such that  $A_1 = S$  and redefine the production rules using the renamed variables. Let the modified grammar be  $G' = (N', T, P', A_1)$ . Using  $G'$ , we construct a tP system  $\Pi$  with one cell and one state as follow:  $\Pi = (N' \cup \{\$, \sigma_1, \emptyset\})$ , where:

$$\sigma_1 = (\{s\}, s, A_1, \{b : sA_i \rightarrow sA_j | A_i \rightarrow bA_j \in P'\} \cup \{b : sA_i \rightarrow s\$ | A_i \rightarrow b \in P'\}).$$

The tP system  $\Pi$  constructed in Theorem 1 works as follows: The system starts in the initial state  $s$  and with the object  $A_1$  which corresponds to the initial symbol in the cell. The use of a rule  $b : sA_i \rightarrow sA_j$  simulates the use of the production  $A_i \rightarrow bA_j \in P'$  and generates the symbol  $b$ . The system halts when the system uses the rule  $b : sA_i \rightarrow s\$$  corresponding to  $A_i \rightarrow b \in P'$  introducing the terminating symbol  $\$$  in the cell. No further rule can be applied and the system halts.

**Theorem 2.**  $(CF - REG) \cap LtP_{*,*}(\gamma, \alpha, \beta) \neq \emptyset, \gamma \in \{nCoo, Coo\}, \alpha \in \{min, par, max\}, \beta \in \{rep, one, spread\}$ .

*Proof.* Let  $L_2$  be the non-regular context-free language  $\{a^n b^n | n \geq 1\}$ . We construct a tP system  $\Pi_2$  with one cell and one state as follows:

$\Pi_2 = (\{a_1, b_1, c_1\}, \sigma_1, \emptyset)$ , where:

$$\sigma_1 = (\{s\}, s, a_1, \{a : sa_1 \rightarrow sa_1 b_1, a : sa_1 \rightarrow sb_1 c_1, b : sb_1 c_1 \rightarrow sc_1\}).$$

The tP system  $\Pi$  constructed in Theorem 2 works as follows: The system starts in the initial state  $s$  with the object  $a_1$  in the cell. By using the rule  $a : sa_1 \rightarrow sa_1 b_1$  labeled with  $a$ ,  $n - 1$  times, the system generates the string  $a^{n-1}$  and introduces  $n - 1$   $b_1$ 's into the system and stays in the state  $s$ . Then it uses the  $a$  labeled rule  $a : sa_1 \rightarrow sb_1 c_1$  once, generating one more  $a$ , removing the object  $a_1$  and introducing one more  $b_1$  and one  $c_1$  into the system. So after this step, the the multiplicity of object  $b_1$  is  $n$ . Then in the next  $n$  steps, the system uses

the  $b$  labeled rule  $b : sb_1c_1 \rightarrow sc_1$ , generating  $n$   $b$ 's and removing the  $b_1$ 's from the system. After this step, the system contains the terminating object  $c_1$  and the system halts the computation generating the string  $a^n b^n$ .

**Theorem 3.**  $CF - LtP_{*,*}(\gamma, \alpha, \beta) \neq \emptyset, \gamma \in \{nCoo, Coo\}, \alpha \in \{min, par, max\}, \beta \in \{rep, one, spread\}$ .

*Proof.* Consider the context free language  $L = \{ww^R | w \in \{a, b\}^*\}$ . Assume that there exists a tP system  $\Pi = (O, \sigma_1, \sigma_2, \dots, \sigma_m, syn)$  with  $m \geq 1$  cells, each of them using at most  $r \geq 1$  states, that generates  $L$ . Consider a string  $uu^R \in L$  and let  $l$  be the length of  $u$ . After reading  $l$  symbols of  $u$ ,  $\Pi$  must be able to reach as many different configurations as there are strings of length  $l$ . This must hold since  $\Pi$  has to remember the first half of the string  $uu^R$  in order to compare it with the second half. Since the alphabet size is two (the argument is applicable to any finite set of cardinality greater than 1),  $\Pi$  has to reach at least  $2^l$  different configurations after reading  $l$  symbols. If  $\Pi$  cannot reach that many configurations, there are two different strings  $u$  and  $u'$ , where the length of  $u'$  is strictly less than  $u$ , that leads  $\Pi$  to the same configuration. So it is required to prove that for sufficiently large  $l$ , only less than  $2^l$  configurations are reachable. The proof is as follows: Let  $O = \{a_1, a_2, \dots, a_k\}$ . Let  $S = P_1 \cup P_2 \cup \dots \cup P_m$  and  $|S| = n$ . Assume a total order on  $S$ . Let the  $i$ th configuration be represented by two row vectors  $c_i^1 = (s_1, \dots, s_m)$ , of size  $m$  representing the state of the  $m$  cells at the  $i$ th step and  $c_i^2 = (a_1^1, a_2^1, \dots, a_k^1, a_1^2, a_2^2, \dots, a_k^2, \dots, a_1^m, a_2^m, \dots, a_k^m)$ , of size  $mk$ , where  $a_i^j \in O$  represent the the multiplicity of the object  $a_i$  in the cell  $j$ . Every rule in  $\Pi$  is of the form  $s^i a_1^{p_1} a_2^{p_2} \dots a_k^{p_k} \rightarrow s^j a_1^{q_1} (a_1^{s_1}, go)(a_1^{t_1}, out) a_2^{q_2} (a_2^{s_2}, go)(a_2^{t_2}, out) \dots a_k^{q_k} (a_k^{s_k}, go)(a_k^{t_k}, out), p_i, q_i, s_i, t_i \geq 0$  and  $s^i, s^j$  are the state of an arbitrary cell before and after the application of the rule. Application of a rule in cell  $i$  takes away  $p_j$   $a_j$ 's from the cell  $i$  and adds  $q_j, s_j, t_j$   $a_j$ 's to the cells indicated in the target field of the rule and the cell changes the state from  $s^i$  to  $s^j$ . Associated with each rule  $r_k, 1 \leq k \leq n$ , we define a row vectors called modification vector  $\mathbf{v}_k = (b_1^1, b_2^1, \dots, b_k^1, b_1^2, b_2^2, \dots, b_k^2, \dots, b_1^m, b_2^m, \dots, b_k^m)$ , of size  $mk$ , where  $b_i^j \in O$  represent the change in the multiplicity of the object  $a_i$  in cell  $j, 1 \leq j \leq m$  on the application of the rule. Application of each rule modifies the configuration by adding a vector  $\mathbf{v}_k$  corresponding to rule  $k$  and also records the change in the state of the cells by modifying the vector corresponding to the state of the system. Suppose that the rule  $r_i, 1 \leq i \leq n$  is used  $k_i, 1 \leq i \leq n$  times during the computation. The configuration of the tP system gets modified to the configuration  $c_0 + \sum_{i=1}^n k_i \cdot \mathbf{v}_i$ , where  $\sum_i k_i \leq l$ , and  $c_0$  is the initial configuration in the vector form, that correspond to the multiplicity of the objects in the cells initially and the state vector also gets modified accordingly. Hence with  $n$  rules we can reach at most as many configurations as there are such tuples  $(k_1, k_2, \dots, k_n)$ . These  $n$  numbers add exactly up to  $l$  and therefore  $0 \leq k_i \leq l$  for all  $i \in \{1, 2, \dots, n\}$ . So there are at most  $(l+1)^n$  such tuples. Therefore, for sufficiently large  $l$  there are less than  $2^l$  different configurations that are reachable by a tP system that generates  $L$ . This concludes the proof.

**Theorem 4.**  $LtP_{*,*}(\gamma, \alpha, \beta) - CF \neq \emptyset, \gamma \in \{nCoo, Coo\}, \alpha \in \{min, par, max\}, \beta \in \{rep, one, spread\}$ .

*Proof.* We can extend the construction of tP system in Theorem 2 to generate non context-free context-sensitive language such as  $\{a^n b^n c^n | n \geq 1\}$ .

**Theorem 5.**  $LtP_{*,*}(\gamma, \alpha, \beta) \subset CS, \gamma \in \{nCoo, Coo\}, \alpha \in \{min, par, max\}, \beta \in \{rep, one, spread\}$ .

*Proof.* We show how to recognize a control word generated by a tP system with a linear bounded automaton. In order to do this, we simulate a computation of tP system by remembering the number of objects in each cell and the state of the system after the generation of each symbol in the control word and show that the total number of objects in the system is bounded on the length with respect to the control word.

Consider a control language  $L$  of a tP system  $\Pi$ . Let  $w = b_1 b_2 \cdots b_k, k \geq 0$  be a control word in  $L$ . Let the number of cells be  $m$  and the total number of rules in all the cells be  $n$ . We build a multi track nondeterministic LBA  $B$  which simulates  $\Pi$ . In order for  $B$  to simulate  $\Pi$ , it has to keep track of the number of objects in each cell and the state of each cell, after generating each symbol. So  $B$  has a track assigned to every rule of  $\Pi$ , a track for each symbol-cell pair  $(a_i, j) \in O \times \{1, 2, \dots, m\}$ , a track for each state-cell pair  $(q_i, j) \in O \times \{1, 2, \dots, m\}$ , and a track for each triple  $(a_i, j, k) \in O \times \{1, 2, \dots, m\}^2$ .  $B$  keeps track of the configurations of  $\Pi$  by writing a positive integer on each track assigned to the symbol-cell pair  $(a_i, j)$ , denoting the number of objects  $a_i$  in cell  $j$  and on each track assigned to the state-cell pair  $(q_i, j)$ , denoting the state  $q_i$  of cell  $j$ . A single step of the computation of  $B$  is as follows: Based on the current configuration (the multiset of objects in each cell and the current state of each cell), the next symbol to be generated,  $B$  choses a set of rules that are to be applied in the next step by writing an integer on the track corresponding to the rules which indicates the number of times that a particular rule is to be applied. Then for each triple  $(a_i, j, k)$ ,  $B$  examines the chosen rule set and writes the number of objects  $a_i$  leaving from cell  $j$  to cell  $k$  on the corresponding track, decreasing the number on the track for  $(a_i, j)$  accordingly. Then it creates the next configuration by adding the values written on the track for each  $(a_i, j, k)$  to the number stored on the track for  $(a_i, k)$ . The system also modifies the state of each cell by modifying the corresponding track entry assigned to each state-cell pair  $(q_i, j)$ . We can see that in any step of the computation, the tracks contain integers bounded by the number of objects inside  $\Pi$  during the corresponding computation step. So if the number of objects inside the tP system in a configuration  $c$  during a computation is bounded by  $S(i)$ , where  $i$  is the number of symbols generated, then the space used by  $B$  to record the configurations and to calculate the configuration change of  $\Pi$  is bounded by  $t \times \log_b(S(i))$ , where  $b$  denotes the base of the track alphabet of  $B$  and  $t$  denotes the number of tracks used. So we can see that for any sequence of accepting computation  $c_0, c_1, \dots, c_m$  in  $\Pi$ , and in any mode in label restricted computation,  $|c_i| \leq k.d$  where  $k$  is a constant depending on the form of rules, and  $d \leq i$  is the number of non-empty multisets

read by the tP system up to reaching configuration  $c_i$ . Finally,  $B$  checks whether any further rules can be applied. If not, it accepts the string, else it rejects. So the total number of objects in the system is bounded with respect to the input length and so the generated language is context-sensitive.

**Theorem 6.**  $L^{\lambda}tP_{1,1}(\gamma, \alpha, \beta) = RE, \gamma \in \{nCoo, Coo\}, \alpha \in \{min, par, max\}, \beta \in \{rep, one, spread\}$ .

*Proof.* Let  $L \subseteq \Sigma^*$  be a recursively enumerable language. Let  $\Sigma = \{b_1, b_2, \dots, b_l\}$ . Define an encoding  $e : \Sigma \mapsto \{1, 2, \dots, l\}$  such that  $e(b_i) = i$ . We extend the encoding for a string  $w = c_1c_2 \dots c_k$  as follows:  $e(w) = c_1 * (l+1)^{(k-1)} + \dots + c_{(k-1)} * (l+1)^1 + c_k * (l+1)^0$ . We use  $l+1$  as the base in-order to avoid the digit 0 at the left end of the string.

For any  $L$ , there exists a deterministic register machine  $M = (m, H, q_0, h, I)$  which halts after processing the input  $i_0$  placed in its input register if and only if  $i_0 = e(w)$  for some  $w \in L$ . So, it is sufficient to show how to generate the encoding  $e(w)$ , and simulate the instructions of a register machine with a tP system. The value of register  $r$  is represented by the multiplicity of the object  $a_r$  in the cell and the label  $l_i$  is represented by an object  $l_i$ .

The instructions of a register machine are simulated by a tP system with a single state  $q$  as follows:

- Add instruction  $l_i : (ADD(r), l_j)$  is simulated by the instruction  $ql_i \rightarrow qa_r l_j$ . Removes the object  $l_i$  and introduces the objects  $a_r$  and  $l_j$  and stays in the state  $q$ .
- Subtract instruction  $l_i : (SUB(r), l_j, l_k)$  is simulated by the instructions  $ql_i \rightarrow ql'_i l'_i, qa_r l'_i \rightarrow ql''_i, ql''_i \rightarrow ql'''_i, ql'''_i l'''_i \rightarrow ql_j, ql'''_i l'_i \rightarrow ql_k$ ; The object  $l_i$  is replaced by two objects  $l'_i, l''_i$  by using the rule  $ql_i \rightarrow ql'_i l''_i$ . If an object  $a_r$  is present in the system, the number of it gets decreased by one by using the rule  $qa_r l'_i \rightarrow ql''_i$ , which also introduces the object  $l''_i$  in the next step. If no  $a_r$  is present,  $l'_i$  remains but  $l''_i$  is introduced into the system by using the rule  $ql'_i \rightarrow ql'''_i$ . In the next step, either object  $l_j$  or  $l_k$  get introduced into the system depending on the objects present in the system by using the rule  $ql'''_i l'''_i \rightarrow ql_j$  or  $ql'''_i l'_i \rightarrow ql_k$ . In all the above steps, the system stays in the state  $q$ .
- For the halt instruction  $l_h : HALT$ , nothing to do. The system halts when the object  $l_h$  is introduced into the system.

We construct a tP system  $\Pi$  with one cell and one state as follows:  $\Pi = (O, \sigma_1, \emptyset)$ , where:

$$O = \{s\} \cup \{l_i, l'_i, l''_i, l'''_i, l''''_i \mid l_i \in H\} \cup \{a_r \mid 1 \leq r \leq m\} \cup \{l_g, l'_g, l_{g_i}, l_{g(l+1)}, l_{a1} \mid 1 \leq i \leq l\} \cup \{l_{c1}, l_{c2}, l_{c3}, l_{r1}, l_{r2}\}.$$

$$\sigma_1 = (\{q\}, q, s, R_1) \text{ with :}$$

$$R_1 = \{b_i : qs \rightarrow ql_g a_i^i \mid b_i \in \Sigma, 1 \leq i \leq l\} \cup \text{set of all rules corresponding to the register machine instructions (} M \text{ and generating encoding in Step 2) labeled with } \lambda.$$



with one cell performing the following operations ( $a_1$  and  $a_2$  are two distinguished objects of  $\Pi$ , where multiplicity of  $a_1$  represents the encoding corresponding to the symbol generated in each step and the multiplicity of  $a_2$  represents the encoding of the generated string up to a particular step).

1. For some  $1 \leq i \leq l$ , generating symbol  $b_i \in \Sigma$ , is performed by using a rule  $b_i : qs \rightarrow ql_g a_1^i$ , labeled with  $b_i$ , that introduces the object  $l_g$ , which is the label of the first instruction for generating the encoding in Step 2 and the object  $a_1$  with multiplicity  $i$ .
2. Perform the computation  $e(ua) = (l+1)*e(u)+e(a)$ ,  $u \in \Sigma^*$ ,  $a \in \Sigma$ . Assume that the encoding of  $u$  is represented by the multiplicity of object  $a_2$ . The encoding of  $ua$  is performed by the following register machine sub-program.  
 $l_g : (SUB(r_3), l_{g1}, l'_g)$ ;  $l_{gi} : (ADD(r_2), l_{g(i+1)})$ ,  $1 \leq i \leq l$ ;  $l_{g(l+1)} : (ADD(r_2), l_g)$  ;  $l'_g : (SUB(r_1), l_{a1}, l_{c1})$  ;  $l_{a1} : (ADD(r_2), l'_g)$  ;  $l_{c1} : (SUB(r_2), l_{c2}, l_{r1})$  ;  $l_{c2} : (ADD(r_3), l_{c3})$  ;  $l_{c3} : (ADD(r_4), l_{c1})$  ;  $l_{r1} : (SUB(r_4), l_{r2}, s)$  ;  $l_{r2} : (ADD(r_2), l_{r1})$ ;  
 The instructions of the sub-program can be translated to the tP system rules as shown in the beginning of the proof. The multiplicity of objects  $a_1, a_2, a_3$  and  $a_4$  corresponds to the content of registers  $r_1, r_2, r_3$  and  $r_4$ .
3. Repeat from step 1, or, non-deterministically, stop the increase in the multiplicity of object  $a_2$  by using a  $\lambda$  labeled rule  $\lambda : qs \rightarrow qq_0$ , where  $q_0$  is an object that corresponds to the label of the first instruction of the register machine  $M$  in Step 4.
4. Multiplicity of  $a_2$  is equal to  $e(w)$  for some  $w \in \Sigma^+$ . We now start to simulate the working of the register machine  $M$  in recognizing the number  $e(w)$ . If the machine halts, by introducing the object  $h$  corresponding to the halt instruction in  $M$ , then  $w \in L$ , otherwise the machine goes into an infinite loop.

So, we can see that the computation halts after generating a string  $w$  if and only if  $w \in L$ .

## 5 Conclusion

In this paper we investigated the control words associated with computations of tP systems. A generating style was adopted: each step of a computation generates a symbol and a string is obtained by concatenating the generated symbols in a halting computation. To this aim, labels are associated with the rules of a tP system, which are symbols of a given alphabet or they are empty. In each transition, only rules with the same label or with the empty label are used. Also  $\lambda$  moves are possible, when all used rules have the empty label, and no symbol is generated.

The families of languages generated in this way are compared with the families of the Chomsky hierarchy. In the case when  $\lambda$  moves are allowed, all recursively enumerable languages can be generated in this way, in the opposite case all generated languages are context-sensitive, all regular languages can be generated

by tissue P systems, not all context-free languages, but there are non-context-free context-sensitive languages which can be obtained as control languages of tP systems.

## References

1. Păun, G.: Computing with membranes. *Journal of Computer and System Science* 61(1), 108–143 (2000)
2. Păun, G.: *Membrane Computing - An Introduction*. Springer, Berlin (2002)
3. Păun, G., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing*. Oxford Univ. Press (2010)
4. Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. *Theoretical Computer Science* 296, 295–326 (2003)
5. Alhazov, A., Ciubotaru, C., Ivanov, S., Rogozhin, Y.: The family of languages generated by non-cooperative membrane systems. In: Gheorghe, M., Hinze, T., Păun, G., Rozenberg, G., Salomaa, A. (eds.) *CMC 2010. LNCS*, vol. 6501, pp. 65–80. Springer, Heidelberg (2010)
6. Alhazov, A., Ciubotaru, C., Rogozhin, Y., Ivanov, S.: The membrane systems language class. In: *Proc. Eighth Brainstorming Week on Membrane Computing*, Sevilla, pp. 23–35 (2010); *Proc. LA Symposium. RIMS Kōkyūroku Series 1691*, Kyoto University, pp. 44–50 (2010)
7. Ibarra, O.H., Păun, G.: Characterizations of context-sensitive languages and other language classes in terms of symport/antiport P systems. *Theoretical Computer Science* 358, 88–103 (2006)
8. Ionescu, M., Martín-Vide, C., Păun, G.: P systems with symport/antiport rules: The traces of objects. *Grammars* 5, 65–79 (2002)
9. Păun, G.: Languages in membrane computing: Some details for spiking neural P systems. In: Ibarra, O.H., Dang, Z. (eds.) *DLT 2006. LNCS*, vol. 4036, pp. 20–35. Springer, Heidelberg (2006)
10. Păun, G., Rozenberg, G., Salomaa, A.: Membrane computing with an external output. *Fundamenta Informaticae* 41(3), 313–340 (2000)
11. Ionescu, M.: *Membrane Computing. Traces, Neural Inspired Models, Controls*, PhD Thesis, URV Tarragona (2008)
12. Minsky, M.: *Computation - Finite and infinite Machines*. Prentice Hall, Englewood Cliffs (1967)
13. Krithivasan, K., Păun, G., Ramanujan, A.: Control words associated with P systems. In: Gheorghe, M., Păun, G., Pérez-Jiménez, M.J. (eds.) *Proceedings of 10th Brainstorming Week on Membrane Computing, Frontiers of Membrane Computing: Open Problems and Research Topics*, Sevilla, vol. II, pp. 171–250 (2012)
14. Ramanujan, A., Krithivasan, K.: Control words of spiking neural P systems. *Romanian J. of Information Science and Technology* (to appear)
15. Păun, G., Pérez-Jiménez, M.J.: Languages and P systems: recent developments (manuscript)
16. Ramanujan, A., Krithivasan, K.: Control Words of Transition P Systems. In: Bansal, J.C., Singh, P.K., Deep, K., Pant, M., Nagar, A.K. (eds.) *BIC-TA 2012. AISC*, vol. 201, pp. 145–155. Springer, Heidelberg (2013)

# Geometric Methods for Analysing Quantum Speed Limits: Time-Dependent Controlled Quantum Systems with Constrained Control Functions

Benjamin Russell and Susan Stepney

Department of Computer Science, University of York, UK, YO10 5DD  
{bjr502,susan.stepney}@york.ac.uk

**Abstract.** We are interested in fundamental limits to computation imposed by physical constraints. In particular, the physical laws of motion constrain the speed at which a computer can transition between well-defined states. Here, we discuss speed limits in the context of quantum computing. We derive some results in the familiar representation, then demonstrate that the same results may be derived more readily by transforming the problem description into an alternative representation. This transformed approach is more readily extended to time-dependent and constrained systems. We demonstrate the approach applied to a spin chain system.

## 1 Problem and Motivation

The Margolus-Levitin bound [11] defines the limit to the speed of dynamical evolution of a quantum system with time-independent Hamiltonian as imposed by the energy expectation. This and other such speed limit bounds have an interpretation in terms of the maximum information processing rate of a quantum systems [10]. This bound complements the Mandelstam-Tamm inequality [16], a bound to the speed of dynamical evolution of a quantum system in terms of the energy *uncertainty*.

However, in the application of quantum optimal control to quantum computing, a time-*dependent* Hamiltonian is more common [15], and a more complete analysis of the limit to the speed of quantum computers needs to take into account the time dependence of the Hamiltonian. A notable result analogous to the Margolus-Levitin bound, applicable to time dependent systems in the adiabatic regime, can be found in [1].

When considering the ‘ultimate’ physical limits to computation [10], one considers a time-independent system as a model for the fastest possible quantum computer. We can consider a time-dependent control system as a sub-system of a larger time-independent system as follows. Let system  $A$  be the computational system, and let another system  $B$  produce the control fields; these could be considered to be subsystems of the larger system  $A \otimes B$ . If  $B$  is chosen to include all

of the environment of  $A$  that significantly affects  $A$ , then the combined system could be described by a time-independent Hamiltonian, subject to the ‘ultimate’ physical limits. Since the control functions must be implemented by some quantum system, the limits quantum mechanics places on the dynamics of the system producing the control fields places limits on the control fields themselves. Hence, in addition to time dependence, the control functions may be subject to further physical constraints.

Geometric derivations [6,16,9] have been used to determine bounds for time-independent systems. Specifically, [9] provides a strong connection between the quantum speed limit and metric structures on unitary groups. Here we extend the kind of analysis of [9] to time-dependent and constrained systems. We analyse the relationship between constraints on the control functions and the quantum speed limit using tools from Riemannian and related geometries.

The structure of the paper is as follows. First we summarise two complementary geometrical formalisms of quantum mechanics, and use these to rederive known time bounds (§2). Then we further develop the approach, to cover time-dependent and constrained systems (§3). We then apply these results to a specific spin chain system (§4).

## 2 The Geometry of Finite Dimensional Quantum Mechanics

Throughout this paper,  $su(n)$  refers to the special unitary Lie algebra of  $N \times N$  anti-hermitian matrices;  $SU(N)$  refers to the group of  $N \times N$  special unitary matrices.

Quantum mechanical states are typically formulated in term of a complex Hilbert space of states [4]. Quantum time evolution is then typically formulated by considering a unitary time evolution operator  $\hat{U}_t$  acting on the state space. In the case of a Hamiltonian (always taken to be a Hermitian operator) with no explicit time dependence,  $\hat{H}$ , the time evolution is given by:

$$\hat{U}_t = \exp\left(-it\hat{H}\right) \quad (1)$$

This solves the Schrödinger equation for the state if one defines the state after time  $t$  to be  $|\psi_t\rangle = \hat{U}_t|\psi_0\rangle$  for some given initial state  $|\psi_0\rangle$ . There are many different approaches to representing the time evolution operator for an explicitly time-dependent Hamiltonian; common methods include the Dyson series and the Magnus expansion [2].

There are, however, other, more geometrical ways to formulate quantum mechanics. The space of states, in the case of a finite dimensional complex Hilbert spaces,  $\mathbb{C}^N$ , can be formulated as a complex projective space [5]. This construction possesses no redundancy in its description of the state of a quantum system. In the Hilbert space construction, physically distinguishable states correspond to equivalence classes of vectors in Hilbert space (each class is comprised of a complex line or ‘ray’). In contrast, a single point in the projective space description

corresponds to a single physically distinct state. The price for this minimality is the increased difficulty of working with differentiable manifolds rather than with vector spaces.

The space of all unitary operators acting on a (finite dimensional) complex Hilbert space of states forms a Lie group [3], the Lie algebra of which consists of all anti-Hermitian operators. Both of these constructions introduce differential geometry into the picture of quantum dynamics.

## 2.1 Metrics on $CP^n$

We do not attempt a full account of the material discussed in this section; [8] is a clear and rigorous source for the mathematics relating to complex projective spaces, projective Hilbert spaces, and the Fubini-Study metric.

The projective structure corresponding to  $\mathbb{C}^{N+1}$ , considered as a vector space without any inner product or norm structure, is  $CP^N$ , a differentiable manifold. In quantum mechanics the standard inner product structure on  $\mathbb{C}^{N+1}$  is employed to form an inner product space  $\langle \mathbb{C}^{N+1}, \langle \cdot | \cdot \rangle \rangle$ , where the standard inner product is given by:

$$\langle \mathbf{u} | \mathbf{v} \rangle = \sum_{k=0}^N \bar{u}_k v_k \quad (2)$$

After quotienting  $\mathbb{C}^{N+1}/\{\mathbf{0}\}$  into equivalence classes to form  $CP^N$ , a natural choice of Riemannian metric arises for  $CP^N$ : the Fubini-Study metric. The infinitesimal form of this metric, the metric tensor, is given by [8]:

$$ds^2 = \frac{\langle \delta\psi | \delta\psi \rangle}{\langle \psi | \psi \rangle} - \frac{\langle \delta\psi | \psi \rangle \langle \psi | \delta\psi \rangle}{\langle \psi | \psi \rangle^2} \quad (3)$$

where  $|\psi\rangle$  is the point on  $CP^N$  that corresponds to the ray in  $\langle \mathbb{C}^{N+1}, \langle \cdot | \cdot \rangle \rangle$  to which  $|\psi\rangle$  belongs;  $|\delta\psi\rangle$  is an element of the tangent space at this point,  $T_{|\psi\rangle}CP^N$ . (Where no confusion arises, we use the common abuse of notation that does not distinguish between the vector  $|\psi\rangle$  and the corresponding point in  $CP^N$ .) This is the unique metric tensor (up to overall scaling by a positive constant) on  $CP^N$  invariant under all unitary transformations of  $|\psi\rangle$  [8].

Invariance under unitary transformations follows directly from the definition of a unitary operator as an operator that leaves all inner products of states in  $\langle \mathbb{C}^{N+1}, \langle \cdot | \cdot \rangle \rangle$  invariant. This is the projective counterpart to the standard inner product on  $\mathbb{C}^{N+1}$ ; that is, it is compatible with the quotient into rays of  $\langle \mathbb{C}^{N+1}, \langle \cdot | \cdot \rangle \rangle$  rather than the quotient of just the vector space structure. This is readily verified: under the transformation  $|\psi\rangle \mapsto Z|\psi\rangle$ , for any  $Z \in \mathbb{C}/\{0\}$ , the metric is invariant:

$$\begin{aligned}
 ds^2 &\mapsto \frac{\langle \delta\psi | \bar{Z} Z | \delta\psi \rangle}{\langle \psi | \bar{Z} Z | \psi \rangle} - \frac{\langle \delta\psi | \bar{Z} Z | \psi \rangle \langle \psi | \bar{Z} Z | \delta\psi \rangle}{\langle \psi | \bar{Z} Z | \psi \rangle^2} \\
 &= \frac{|Z|^2}{|Z|^2} \frac{\langle \delta\psi | \delta\psi \rangle}{\langle \psi | \psi \rangle} - \left( \frac{|Z|^2}{|Z|^2} \right)^2 \frac{\langle \delta\psi | \psi \rangle \langle \psi | \delta\psi \rangle}{\langle \psi | \psi \rangle^2} \\
 &= ds^2
 \end{aligned} \tag{4}$$

The finite form of this metric is given by [14]:

$$\gamma(|\psi\rangle, |\phi\rangle) = \arccos \sqrt{\frac{|\langle \psi | \phi \rangle|^2}{\langle \psi | \psi \rangle^2 \langle \phi | \phi \rangle^2}} \tag{5}$$

and in this metric, the manifold forms a metric space. The metric clearly has the same invariance and uniqueness properties as its infinitesimal form, eqn.(3).

### 2.2 Deriving the Mandelstam-Tamm Inequality

Our following rederivation of the Mandelstam-Tamm inequality illustrates the usefulness of Riemannian geometry in the context of quantum speed limits. From eqn.(5) we have that  $\forall |\psi\rangle, |\phi\rangle \in \mathbb{C}^N$   $\langle \psi | \phi \rangle = 0$  implies  $\gamma(|\psi\rangle, |\phi\rangle) = \arccos(0) = \frac{\pi}{2}$ . Hence (if all states involved are initially normalised), if  $|\psi_t\rangle$  connects to orthogonal state in time  $\tau$ , i.e.  $\langle \psi_\tau | \psi_0 \rangle = 0$ , then:

$$\begin{aligned}
 L[|\psi_t\rangle] &= \int_{t=0}^{t=\tau} ds \\
 &= \int_{t=0}^{t=\tau} \sqrt{\frac{\langle \delta\psi | \delta\psi \rangle}{\langle \psi | \psi \rangle} - \frac{\langle \delta\psi | \psi \rangle \langle \psi | \delta\psi \rangle}{\langle \psi | \psi \rangle^2}} dt \\
 &= \int_{t=0}^{t=\tau} \sqrt{\langle \delta\psi | \delta\psi \rangle - \langle \delta\psi | \psi \rangle \langle \psi | \delta\psi \rangle} dt \geq \frac{\pi}{2}
 \end{aligned} \tag{6}$$

where  $L$  is the length functional for a curve on the projective space according to the Fubini-Study metric. Here the inequality follows from the definition of a geodesic.

In the case that  $|\psi_t\rangle$  solves the Schrödinger equation for a time-independent Hamiltonian  $\hat{H}$  we have:

$$|\delta\psi_t\rangle = \frac{d}{dt}|\psi_t\rangle = \frac{d}{dt} \exp(-it\hat{H}) |\psi_0\rangle = -i\hat{H} \exp(-it\hat{H})|\psi_0\rangle = -i\hat{H}|\psi_t\rangle \tag{7}$$

Substituting this into eqn.(6), we find:

$$\begin{aligned}
 L[|\psi_t\rangle] &= \int_{t=0}^{t=\tau} \sqrt{\langle \delta\psi_t | \delta\psi_t \rangle - \langle \delta\psi_t | \psi_t \rangle \langle \psi_t | \delta\psi_t \rangle} dt \\
 &= \int_{t=0}^{t=\tau} \sqrt{\langle \psi_t | \hat{H}^2 | \psi_t \rangle - \langle \psi_t | \hat{H} | \psi_t \rangle^2} dt \\
 &= \int_{t=0}^{t=\tau} \Delta E_{|\psi_t\rangle} dt = \int_{t=0}^{t=\tau} \Delta E_{|\psi_0\rangle} dt = \tau \Delta E_{|\psi_0\rangle} \geq \frac{\pi}{2}
 \end{aligned} \tag{8}$$

Here  $\Delta E_{|\psi_t\rangle} dt$  can be replaced by  $\Delta E_{|\psi_0\rangle}$  in the last line since the Hamiltonian is time-independent, which implies that the energy uncertainty is also. From this follows the Mandelstam-Tamm inequality:

$$\tau \geq \frac{\pi}{2\Delta E_{|\psi_0\rangle}} \tag{9}$$

It is worth comparing our derivation to that in [6] (their eqns. 22-25; note that the ‘Wooters distance’ is simply the finite form of the Fubini-Study metric applied to normalised states). There the finite form of the metric is differentiated; here we use the differential form of the metric immediately.

### 2.3 Metrics on $SU(N)$

There is a natural choice of metric tensor on the Lie group of all special unitary operators acting on  $\mathbb{C}^N$ ,  $SU(N)$ . This is due to a general result about symmetric bilinear forms on semi-simple Lie groups. ( $U(N)$  is not semi-simple, and so we specialise from here on to  $SU(N)$ , which is in fact simple.) Again, we do not give a complete description of these constructions; specifically we do not discuss adjoint representations and the general definition of the Killing form, but instead specialise to  $SU(N)$  immediately. A good source for this material is [7].

The Killing form (denoted by  $B$ ) is the unique symmetric bilinear form (up to a positive constant multiple) on  $su(N)$  (which consists of all traceless, anti-hermitian operators on  $\mathbb{C}$ ) satisfying  $\forall x, y, z \in su(N)$ :

1.  $B([x, y], z) = B(x, [y, z])$
2.  $B(s(x), s(y)) = B(x, y)$  for any automorphism  $s$  of  $su(n)$ .

For  $su(n)$ , the Killing form is given by:

$$B(x, y) = 2n \operatorname{Tr}(xy) \tag{10}$$

Then  $g(x, y) = -B(x, y) = 2n \operatorname{Tr}(x^\dagger y)$  is a Riemannian metric on  $SU(n)$ . The length of a smooth curve (according to the metric  $g$ ) on  $\hat{U}_t SU(n)$  is now given by:

$$L[\hat{U}_t] = \int_{t=0}^{t=\tau} \sqrt{2n \operatorname{Tr} \frac{d\hat{U}_t}{dt} \dagger \frac{d\hat{U}_t}{dt}} dt \tag{11}$$

In the case that  $\hat{U}_t$  solves the Schrödinger equation for some (possibly time-dependent) Hamiltonian  $\hat{H}_t$

$$\frac{d}{dt} \hat{U}_t = -i\hat{H}_t \hat{U}_t \tag{12}$$

then the length of this curve can be written in terms of  $\hat{H}_t$ . This follows from the Schrödinger equation and the unitary invariance of the Killing form (due to the cyclic property of the trace):

$$\begin{aligned}
 L[\hat{U}_t] &= \int_{t=0}^{t=\tau} \sqrt{2n \operatorname{Tr} \frac{d\hat{U}_t^\dagger}{dt} \frac{d\hat{U}_t}{dt}} dt & (13) \\
 &= \int_{t=0}^{t=\tau} \sqrt{2n \operatorname{Tr} \left(-i\hat{H}_t\hat{U}_t\right)^\dagger \left(-i\hat{H}_t\hat{U}_t\right)} dt \\
 &= \int_{t=0}^{t=\tau} \sqrt{2n \operatorname{Tr} \left(\hat{U}_t^{-1}\hat{H}_t^2\hat{U}_t\right)} dt \\
 &= \int_{t=0}^{t=\tau} \sqrt{2n \operatorname{Tr} \left(\hat{H}_t^2\hat{U}_t\hat{U}_t^{-1}\right)} dt \\
 &= \int_{t=0}^{t=\tau} \sqrt{2n \operatorname{Tr} \left(\hat{H}_t^2\right)} dt
 \end{aligned}$$

Notice that the dependence on the operator  $\hat{U}_t$  has disappeared. This reduces, in the case of time-independent  $\hat{H}$ , to:

$$L[\hat{U}_t] = \int_{t=0}^{t=\tau} \sqrt{2n \operatorname{Tr} \hat{H}^2} dt = \tau \sqrt{2n \operatorname{Tr} \hat{H}^2} \tag{14}$$

### 2.4 A Bound on the Orthogonality Time

To illustrate the relationship between point to set distances on the special unitary group and speed limits for state transfer problems, we include our deviation of a bound on the orthogonality time (for time-independent systems) similar to the Margolus-Levitin bound.

Consider the shortest time that  $\langle \psi_0 | \psi_t \rangle = \langle \psi_0 | \hat{U}_t | \psi_0 \rangle = 0$  can be achieved. This is the same as the shortest time in which the time evolution operator can be driven from  $\hat{I}$  to some  $\hat{U}_t$  achieving this. Any such  $\hat{U}_t$  achieving this, for a time-independent system, must have the following form for some unitary change of basis matrix  $\hat{V}$ , some unitary  $\hat{A}$  and some  $\theta \in [0, 2\pi]$ :

$$\hat{U}_t = \exp(-it\hat{H}) = \hat{V}^\dagger \hat{B} \hat{V} \tag{15}$$

where

$$\hat{B} = \begin{pmatrix} 0 & -\exp(-i\theta) & 0 \dots 0 \\ \exp(i\theta) & 0 & 0 \dots 0 \\ 0 & 0 & \hat{A} \\ \vdots & \vdots & \\ 0 & 0 & \end{pmatrix} \tag{16}$$

Consider the function  $f_p : SU(N) \rightarrow \mathbb{R}$  (for  $p \geq 1$ ) defined as:

$$f_p(\hat{A}) \stackrel{\text{def}}{=} \operatorname{Tr} \left( \left| \log(\hat{A}) \right|^p \right) \tag{17}$$



where  $|\hat{A}| = \sqrt{\hat{A}^\dagger \hat{A}}$ . Applying this function to both sides of eqn.(15) gives:

$$\text{Tr}(|-it\hat{H}|^p) = \text{Tr}(|\log(\hat{V}^\dagger \hat{B} \hat{V})|^p) \quad (18)$$

This implies:

$$\begin{aligned} t^p \text{Tr}(|\hat{H}|^p) &= \text{Tr}(|\log \hat{B}|^p) \\ &= \text{Tr}(|\log \begin{pmatrix} 0 & -\exp(-i\theta) \\ \exp(i\theta) & 0 \end{pmatrix}|^p) + \text{Tr}(|\log \hat{A}|^p) \\ &\leq \text{Tr}(|\log \begin{pmatrix} 0 & -\exp(-i\theta) \\ \exp(i\theta) & 0 \end{pmatrix}|^p) = \frac{2\pi^p}{2^p} \end{aligned} \quad (19)$$

Taking the  $p^{\text{th}}$  root of each side yields the bound:

$$t \geq \frac{2^{\frac{1}{p}} \pi}{2 \text{Tr}(|\hat{H}|^p)^{\frac{1}{p}}} = \frac{\pi}{2} \left( \frac{2}{\text{Tr}(|\hat{H}|^p)} \right)^{\frac{1}{p}} \quad (20)$$

This is similar to, but not as strong as, the bounds given in [9]. As in [9], it is possible to optimise this bound, to:

$$t \geq \min_{\epsilon \in \mathbb{R}} \frac{\pi}{2} \left( \frac{2}{\text{Tr}(|\hat{H} + \epsilon \hat{I}|^p)} \right)^{\frac{1}{p}} \quad (21)$$

by reassigning a new ground state energy.

This bound coincides with the Margolus-Levitin bound for a two level system and  $p = 1$ . This bound with  $p = 2$  corresponds to the bound arising from the metric induced by the Killing form.

### 3 Speed Limits for Time Dependent Controls with Constraints

A mathematical method for obtaining answers to the following questions about a time-dependent quantum system with constrained control functions is relevant to quantum computing:

1. Given two states, what is the least time the system can transfer between them (if the constraint permits this transformation) and which control functions cause this to happen?
2. Given a desired time evolution operator, what is the least time the system can transfer from the identity on the unitary group to it (if the constraint permits this transformation) and which control functions cause this to happen?

In many physically plausible cases, the time-dependent Hamiltonian for a candidate system for the implementation of quantum gates can be cast in the form:

$$\hat{H}(t) = \hat{H}_{int} + \sum_n^M f_n(t) \hat{H}_n \quad (22)$$

where  $\hat{H}_{int}$  is the time-independent portion of the Hamiltonian;  $M$  is the number of control functions;  $f_n$  are control functions, and  $H_n$  is the  $n^{th}$  control Hamiltonian.

### 3.1 Constraints as Submanifolds of $su(N)$

Consider a system with Hamiltonian  $\hat{H}(t)$  given as in eqn.(22). One can form a geometric interpretation of a constraint on the control functions  $f_n$  by considering the relationship between a constraint given (perhaps implicitly) by an equation of the form  $F(f_1, \dots, f_M) = c$  and submanifolds of the tangent spaces to  $SU(n)$ .

In cases where  $F : \mathbb{R}^M \rightarrow \mathbb{R}$  is a sufficiently smooth function the level sets  $\{(f_1, \dots, f_M) \in \mathbb{R}^M \text{ s.t. } F(f_1, \dots, f_M) = c\}$  foliate  $\mathbb{R}^M$ . The intuitive picture of a level set in this context is given by imagining the set of all vectors in a vector space with the same length according to some norm. That is: each level set is a disjoint submanifold of  $\mathbb{R}^M$ ; each level set has the same dimension  $(M - 1)$ ; the union of all levels sets is  $\mathbb{R}^M$ . These level sets can be carried over to the tangent spaces to  $SU(N)$  at each point  $\hat{A} \in SU(N)$ ,  $T_{\hat{A}}SU(n)$ , by considering the submanifold of tangent vectors compatible with the constraint given. This set can be expressed as:

$$\mathcal{A}_F \stackrel{\text{def}}{=} \left\{ i\hat{H} \in T_{\hat{A}}SU(n) \text{ s.t. } \hat{H} = \hat{H}_{int} + \sum_{n=0}^M f_n \hat{H}_n, F(f_1, \dots, f_M) = c \right\} \quad (23)$$

### 3.2 Sphere Bundles and Speed Limits

We now consider the relationship between speed limits arising from a constraint  $F$  on the control functions and a special type of Riemannian metric on  $SU(n)$ . For the purpose of simplifying the statement of the following theorem, define:

**Definition 1.** A smooth parametrised curve  $\hat{U}(t) \in SU(n)$  (parametrised by  $t \in [0, \tau]$ ) is said to be Constraint Compatible for a constraint function  $F$  if  $\forall t \in [0, \tau], d\hat{U}/dt \in \mathcal{A}_F$ .

**Theorem 1.** Each Riemannian metric  $g : TSU(n) \times TSU(n) \rightarrow \mathbb{R}$  on  $SU(n)$  s.t.  $\forall \hat{A} \in \mathcal{A}_F, g(\hat{A}, \hat{A}) \leq 1$  satisfies for each smooth, constraint compatible curve on  $\hat{U}(t) \in SU(n)$ :

$$L[\hat{U}(t)] \leq \tau \quad (24)$$

*Proof.*

$$L[\hat{U}(t)] = \int_{t=0}^{t=\tau} \sqrt{g(\hat{A}, \hat{A})} dt \leq \int_{t=0}^{t=\tau} 1 dt = \tau \quad (25)$$

□

### 4 Application to Spin Chains

We now apply these results to a specific quantum system of relevance in quantum computing: a spin chain. A controlled Heisenberg spin chain of  $N$  spins (with coupling constants  $J_x, J_y, J_z$  has Hamiltonian [12]:

$$\left( \sum_{k \in \{x,y,z\}} J_k \left( \sum_{n=0}^{N-2} \hat{I}_2^{\otimes n} \otimes \sigma^k \otimes \sigma^k \otimes \hat{I}_2^{\otimes N-n-2} \right) \right) + \left( \sum_{n=0}^{N-1} f_n(t) \hat{I}_2^{\otimes n} \otimes \sigma^z \otimes \hat{I}_2^{\otimes N-n-1} \right) \tag{26}$$

Apply the constraint that the total energy used to produce the control functions is less than  $\kappa^2$ , to obtain:

$$\sum_{k=0}^{N-1} f_n(t)^2 \leq \kappa^2 \tag{27}$$

This implies (after some algebra) the following theorem:

**Theorem 2.** *The Riemannian metric that is the largest multiple of the Killing form of  $su(2^N)$  such that all controlled Heisenberg model Hamiltonians (with  $N$  spins) that obey the constraint satisfy  $g_{op}(H_t, H_t) \leq 1$  is given by:*

$$g_{op}(x, y) = \frac{-B(x, y)}{2^{N+1} ((N - 1)(J_x^2 + J_y^2 + J_z^2) + \kappa^2)} = \frac{\text{Tr}(x^\dagger y)}{(N - 1)(J_x^2 + J_y^2 + J_z^2) + \kappa^2} \tag{28}$$

*Proof.* Omitted: simple but laborious.

This theorem, combined with eqn.(25), yields:

$$\tau \geq \int_{t=0}^{t=\tau} \sqrt{g_{op}(\hat{H}_t, \hat{H}_t)} dt \tag{29}$$

$$= \frac{1}{\sqrt{(2^{N+1}) ((N - 1)(J_x^2 + J_y^2 + J_z^2) + \kappa^2)}} \int_{t=0}^{t=\tau} \sqrt{-B(-i\hat{H}_t, -i\hat{H}_t)} dt \tag{30}$$

$$= \frac{1}{\sqrt{(N - 1)(J_x^2 + J_y^2 + J_z^2) + \kappa^2}} \int_{t=0}^{t=\tau} \sqrt{\text{Tr}(\hat{H}_t^2)} dt \tag{31}$$

for any values of the control functions. We appeal to the following facts:

1. The one parameter subgroups of unitary groups are the geodesics of the Killing form since it is (strictly  $-B$  is the metric) a bi-invariant metric [8].

2. Stone’s theorem [13] guarantees that the one parameter subgroups take the form  $\exp(-it\hat{H})$  for some hermitian  $\hat{H}$ , which is nothing other than the form of the time evolution operator for a time independent quantum system with Hamiltonian  $\hat{H}$ .

Using these, the known length of the appropriate geodesic (which can be readily calculated by finding the length of the one-parameter subgroup connecting the identity to the desired transformation according to the metric given by  $-B$ ), and statements made above, we get:

$$\tau \geq \frac{1}{\sqrt{(2^{N+1})((N-1)(J_x^2 + J_y^2 + J_z^2) + \kappa^2)}} \int_{t=0}^{t=\tau} \sqrt{-B(-i\hat{H}_t, -i\hat{H}_t)} dt \tag{32}$$

$$\geq \frac{\pi}{\sqrt{(N-1)(J_x^2 + J_y^2 + J_z^2) + \kappa^2}} \tag{33}$$

Thus we conclude that a Heisenberg spin chain with  $N$  spins, constrained as described, cannot transfer from one state to an orthogonal state in less time than:

$$\tau \geq \pi / \sqrt{(N-1)(J_x^2 + J_y^2 + J_z^2) + \kappa^2} \tag{34}$$

The uniqueness properties of the Killing form guarantee that this is the best (largest lower bound on  $t$ ) speed limit available from any bi-invariant, unitarily invariant metric, as any such metric is a multiple of the Killing form and  $g_{opt}$  is the largest multiple meeting the premises of theorem eqn.(1).

## 5 Conclusions and Further Work

We have demonstrated the use of geometric formalisms in deriving time bounds on quantum systems. We have used this approach to rederive known results in a more compact and elegant manner. More importantly, the new formulation allows us to extend the approach to time dependent and constrained systems, as relevant to quantum computation. We have demonstrated this for a spin chain system.

Next steps include:

- Determining which metric-like structures on unitary group can be used to derive quantum speed limits, including an investigation into the possible role of Finsler functions as these generalise Riemannian metrics but can still produce speed limits.
- Understanding more clearly the relationship between problems 1 and 2 (as discussed in §3). Understanding this relationship in terms of homogeneous spaces (in the sense that  $CP^N \cong SU(N)/U(N-1)$ ) and Riemannian symmetric spaces. Are metric-like structures on special unitary groups the best method for deriving speed limits for state transfer problems 1 or do metrics on complex projective spaces suffice.

- Extending the analysis of the relationship between constrained control functions and geometric derivations of quantum speed limits, to determine which classes of metric-like structures can yield speed limit theorems for which classes of constraints.
- Extending the approach to other Hamiltonian systems, particularly to investigate how quickly one can transfer from a separable state to a maximally entangled state in the presence of constrained control functions.

**Acknowledgements.** We would like to thank Sam Braunstein for many helpful discussions, and Eli Hawkins for much input, particularly his observations on theorem (1). Russell is supported by an EPSRC DTA grant.

## References

1. Andrecut, M., Ali, M.K.: The adiabatic analogue of the Margolus–Levitin theorem. *Journal of Physics A: Mathematical and General* 37(15), L157 (2004)
2. Blanes, S., Casas, F., Oteo, J.A., Ros, J.: The Magnus expansion and some of its applications. *Physics Reports* 470(5-6), 151–238 (2009)
3. Bump, D.: *Lie Groups*. Springer (2004)
4. Gasiorowicz, S.: *Quantum Physics*. John Wiley & Sons (1995)
5. Jia, B., Lee, X.-G.: Quantum states and complex projective space. *ArXiv Mathematical Physics e-prints* (2007)
6. Jones, P.J., Kok, P.: Geometric derivation of the quantum speed limit. *ArXiv Mathematical Physics e-prints* 82(2), 022107 (2010)
7. Knapp, A.W.: *Lie Groups Beyond an Introduction*. Progress in Mathematics. Birkhäuser, Basel (2002)
8. Kobayashi, S., Nomizu, K.: *Foundations of Differential Geometry*. Wiley (1996)
9. Lee, K.-Y., Chau, H.F.: Relation between quantum speed limits and metrics on  $U(n)$ . *Journal of Physics A Mathematical General* 46(1), 015305 (2013)
10. Lloyd, S.: Ultimate physical limits to computation. *Nature* 406(6799), 1047–1054 (1999)
11. Margolus, N., Levitin, L.B.: The maximum speed of dynamical evolution. *Physica D* 120, 188–195 (1998)
12. Mohn, P.: *Magnetism in the Solid State: An Introduction*. Springer (2003)
13. Stone, M.H.: On one-parameter unitary groups in Hilbert space. *Ann. Math.* 33(2), 643–648 (1932)
14. Wells, R.O.N.: *Differential Analysis on Complex Manifolds*. Graduate Texts in Mathematics. Springer (1980)
15. Werschnik, J., Gross, E.K.U.: *Quantum Optimal Control Theory*. *ArXiv e-prints*, arXiv:0707.1883 (July 2007)
16. Zwierz, M.: Comment on “Geometric derivation of the quantum speed limit”. *ArXiv Mathematical Physics e-prints* 86(1), 016101 (2012)

# Numerical Analysis of Quantum Speed Limits: Controlled Quantum Spin Chain Systems with Constrained Control Functions

Benjamin Russell and Susan Stepney

Department of Computer Science, University of York, UK, YO10 5DD  
{bjr502,susan.stepney}@york.ac.uk

**Abstract.** We are interested in fundamental limits to computation imposed by physical constraints. In particular, the physical laws of motion constrain the speed at which a computer can transition between well-defined states. Certain time bounds are known, but these are not tight bounds. For computation, we also need to consider bounds in the presence of control functions. Here, we use a numerical search approach to discover specific optimal control schemes. We present results for two coupled spins controlled in two scenarios: (i) a single control field influencing each spin separately; (ii) two orthogonal control fields influencing each spin.

## 1 Introduction

Computers operate under physical laws, which constrain their operation. In particular, the physical laws of motion constrain the speed at which a computer can transition between well-defined states.

In the case of quantum systems, certain fundamental bounds on these transition times are known. The Margolus-Levitin bound [5] demonstrates the limit to the speed of dynamical evolution of any quantum system with a time-independent Hamiltonian as imposed by the energy expectation. This and other such speed limit bounds have an interpretation in terms of the maximum information processing rate of a quantum systems [4]. This bound complements the Mandelstam-Tamm inequality [12], a bound to the speed of dynamical evolution of a quantum system in terms of the energy *uncertainty*.

When considering a quantum computer, we consider a system  $A$  implementing a computation and a system  $B$  producing the control fields specifying the particular computation. Together these can be considered to be subsystems of a larger quantum system  $A \otimes B$ . This highlights that the control functions are implemented by some quantum system, and the limits quantum mechanics places on the dynamics of the system producing the control fields places limits on the control fields themselves. Hence, as well as the computational system  $A$ , the control functions  $B$  are also subject to physical constraints.

These constraints arise variously. The energy available for the production of control fields limits is one obvious restriction. However there are other limitations that arise from physical constraints, presented by physical laws rather than

engineering difficulties or limitations on resources, on the devices producing the control fields. For example, allowing more energy to produce the control fields could allow the production of fields with greater amplitudes, however, no amount of energy could facilitate any part of the device producing control fields to move faster than the vacuum speed of light; this can constrain the frequency of time-varying fields. Ultimately the device producing the control fields is subject to the laws of quantum mechanics and relativity, which limit the detail with which a control field can be specified, and the rate at which it can change in time.

As a consequence, when attempting to understand the physical limits to the speed of quantum computers, we must also take into account the limits placed on computation speed by constraints on the control functions.

First we introduce an exemplar problem and certain plausible constraints (§2). Next we describe the optimisation problem approach (§3). Then we describe the numerical approach we use for finding optimal solutions (§4). We then apply this approach to a two-bit chain, deriving specific results (§5).

## 2 The Exemplar Problem

### 2.1 Heisenberg Spin Chain

To investigate the effect of constraints on the control function, we consider an exemplar problem relevant to quantum computation: a controlled Heisenberg spin chain of  $N$  spins (with coupling constants  $J_x, J_y, J_z$ . This has Hamiltonian [6]:

$$\begin{aligned} & \sum_{k \in \{x, y, z\}} J_k \left( \sum_{n=0}^{N-2} \hat{I}_2^{\otimes n} \otimes \sigma^k \otimes \sigma^k \otimes \hat{I}_2^{\otimes N-n-2} \right) \\ & + \sum_{n=0}^{N-1} g_n(t) \hat{I}_2^{\otimes n} \otimes \sigma^z \otimes \hat{I}_2^{\otimes N-n-1} \\ & + \sum_{n=0}^{N-1} h_n(t) \hat{I}_2^{\otimes n} \otimes \sigma^y \otimes \hat{I}_2^{\otimes N-n-1} \end{aligned} \quad (1)$$

where:  $\sigma^k$  is the  $k^{th}$  standard Pauli matrix; the  $g_n$  and  $h_n$  represent the externally generated, potentially time-dependant, control fields in the  $z$  and  $y$  directions respectively. For the case of a single,  $z$  direction, control field per spin, the  $h_n$  are zero.

Two central problems in quantum optimal control are: 1) Determine whether or not a specific system can implement a desired unitary time evolution (controlability); and, if so, 2) how quickly this desired transformation can be achieved (time optimality) [11]. It is the second of these, specifically for a Heisenberg model spin chain, that we address here.

## 2.2 Band-Limited Fourier Series for Representing Control Functions with Bounded Rate of Change

In [8] we study geometric methods for determining speed limits on implementing quantum information processing tasks in the presence of constraints of the control functions of a constrained quantum system. There, no comment is made about constraints on the *time derivatives* of the control functions. Such constraints represent the maximum rate at which a control function can change. Here, we have chosen to represent control functions  $g_k(t)$  ( $t \in [0, \tau]$ ,  $\tau \leq 1$ ) which have bounded rate of change by band-limited Fourier series:

$$g_k(t) = \frac{a_{k,0}}{2} + \sum_{m=1}^M (a_{k,m} \sin(mt) + b_{k,m} \cos(mt)) \quad (2)$$

where the relationship between the Fourier coefficients  $\{a_{k,m}, b_{k,m}\}$  and the control function  $g_k$  are given by the usual formula. (For simplicity in the following discussions we ignore the orthogonal control field  $h_k$ ; they are either treated analogously to the  $g_k$  in the two control field case, or are zero in the one control field case.)

We can easily find a bound on the magnitude of the derivative of each control function  $g_k$  in terms of its degree of band-limiting  $M$  and its largest Fourier coefficient  $A_k = \max\{|a_{k,m}|, |b_{k,m}|\}$ :

$$\left| \frac{d}{dt} g_k(t) \right|^2 = \left| \frac{d}{dt} \sum_{m=1}^M (a_{k,m} \sin(mt) + b_{k,m} \cos(mt)) \right|^2 \quad (3)$$

$$= \left| \sum_{m=1}^M m (a_{k,m} \cos(mt) - b_{k,m} \sin(mt)) \right|^2 \quad (4)$$

$$\leq \sum_{m=1}^M m^2 |a_{k,m} \cos(mt) - b_{k,m} \sin(mt)|^2 \quad (5)$$

$$\leq \sum_{m=1}^M m^2 (a_{k,m}^2 \cos^2(mt) - a_{k,m} b_{k,m} \sin(2mt) + b_{k,m}^2 \sin^2(mt)) \quad (6)$$

$$\leq \sum_{m=1}^M m^2 (a_{k,m}^2 + b_{k,m}^2 + 1) \quad (7)$$

$$\leq (2A_k + 1) \sum_{m=1}^M m^2 = (2A_k + 1)M(M+1)(2M+1)/6 \quad (8)$$

## 2.3 Band-Limited Fourier Series for Representing Control Functions with Bounded Power

In this band-limited Fourier representation, the constraint that the total energy used to produce each of the control fields individually (the power) is bounded



by  $\kappa$  takes a simple form due to Parseval's theorem. The desired constraint is:

$$\int_0^\tau g_k^2(t) dt \leq \kappa^2 \quad (9)$$

which represents the constraint on the total power used in the production of the control fields. Parseval's theorem tell us that:

$$\int_0^\tau g_k^2(t) dt = \frac{1}{\tau} \sum_{m=0}^M (a_{k,m}^2 + b_{k,m}^2) \quad (10)$$

### 3 Optimisation Problem

#### 3.1 General Optimisation

We wish to find control functions which maximise functionals (whose relevance is described in [1] and elsewhere) of the form:

$$F[\mathbf{g}] = \Re \text{Tr}(\hat{O}^\dagger \hat{U}_t) \quad (11)$$

where  $\mathbf{g}$  is the vector of control functions for a controlled quantum system,  $\hat{O}$  is a desired unitary transformation and  $\hat{U}_t$  is the time-evolution operator obtained from applying the control functions to the system in question.

Maximising such functionals is equivalent to minimising the euclidean distance between the operators  $\hat{O}$  and  $\hat{U}_t$ :  $\|\hat{O} - \hat{U}_t\|^2 = \text{Tr}((\hat{O} - \hat{U}_t)^\dagger (\hat{O} - \hat{U}_t))$ . This can be seen by expanding out the right hand side and discarding terms that are constants as they have no effect on the optimisation.

#### 3.2 Goal Operators

We need to choose specific unitary transformations as candidates for optimisation (the  $\hat{O}$  transform in eqn (11)). Permutation matrices are clearly unitary as they are orthogonal, and they are suitable initial candidates for goal operators.

As the  $n \times n$  permutation matrices over  $\mathbb{C}$  are a faithful representation of the symmetric group of all permutations of  $n$  letters [7], there will be  $n!$  such matrices. In the case of a system of  $N$  qubits, the time evolution operator is a  $2^N \times 2^N$  matrix, and thus we consider all permutation matrices of the same size as potential goal operators.

There are  $(2^N)!$  such matrices. For a  $N = 2$  system, there are 24 such matrices. The identity can be excluded from further consideration as it can be trivially 'implemented' by any quantum system: after no time, nothing happens!

## 4 Numerical Method

#### 4.1 Gradient Descent

Consider the problem of optimising (minimising) a function  $f : \mathbb{R}^N \rightarrow \mathbb{R}$  (for some  $N \in \mathbb{N}$ ). If one were performing gradient descent to minimise (assuming

---

**Algorithm 1.** Gradient descent pseudo-code

---

```

1:  $\mathbf{x} := \text{random\_vector}$ 
2: while  $f(\mathbf{x}) \leq \text{threshold}$  do
3:    $\mathbf{x}' := \mathbf{x} + \text{move\_size} * \text{random\_unit\_vector}$ 
4:    $\Delta := f(\mathbf{x}') - f(\mathbf{x})$ 
5:   if  $\Delta < 0$  then
6:      $\mathbf{x} := \mathbf{x}'$ 
7:   end if
8: end while

```

---

the existence of at least a single minimum) such an  $f$ , starting from a randomly chosen initial condition, then algorithm 1 would describe the method.

The ‘Grape’ (Gradient Ascent Pulse Engineering) algorithm for the discovery of control schemes has been well studied in the context of quantum computing and quantum optimal control in the presence of constrained control functions [9], [10]. It facilitates applying an iterative gradient ascent (or descent depending on the problems formulation, the two are equivalent) method to find control functions which maximise functionals of the form of eqn.(11).

The control ‘landscape’ for such functionals has also been studied [2], and has been found to potentially possess multiple optima. In the presence of local optima, gradient ascent methods of optimisation can become ‘trapped’. Hence one needs a more sophisticated search algorithm.

## 4.2 Simulated Annealing

Simulated annealing (SA) [3] is an alternative method to gradient descent for optimising a (real valued, sufficiently smooth) function of several (finitely many, real valued) variables that attempts to overcome the difficulty of local optima.

SA varies the gradient descent method by including a probabilistic acceptance of non-improving moves, in order to escape from local optima. In order to achieve this, a ‘cooling schedule’ is introduced via the introduction of a global ‘temperature’ variable that decreases as the system ‘cools’. When the system is ‘hot’ there is a relatively high chance of accepting a non-improving move, but after it has cooled significantly, this probability drops to zero. The probability of accepting a non-improving move is given by the Boltzmann distribution.

Algorithm 2 describes the method. Here  $T_0$  is the initial temperature, and  $\delta T$  controls the cooling rate. These parameters are to be chosen according to the specific application; there is no known general principle for choosing the most effective values and experimentation is frequently needed to find effective values [3].

## 4.3 SA for Constrained Fourier Series

In order to impose constraints, we augment standard SA with a rejection sampling method; moves are allowed only if the proposed new state does not violate the constraint.

---

**Algorithm 2.** Simulated Annealing pseudo-code

---

```

1:  $\mathbf{x} := \text{random\_vector}$ 
2:  $T := T_0$ 
3: while  $f(\mathbf{x}) \leq \text{threshold} \wedge 0 \leq T$  do
4:    $\mathbf{x}' := \mathbf{x} + \text{move\_size} * \text{random\_unit\_vector}$ 
5:    $\Delta := f(\mathbf{x}') - f(\mathbf{x})$ 
6:   if  $\Delta < 0 \vee \text{rand}(0, 1) \leq \exp(-\Delta/T)$  then
7:      $\mathbf{x} := \mathbf{x}'$ 
8:   end if
9:    $T := T - \delta T$ 
10: end while

```

---



---

**Algorithm 3.** Simulated Annealing for constrained Fourier series pseudo-code

---

```

1: repeat
2:    $\mathbf{G} := \text{random\_vector}$ 
3: until  $C(\mathbf{G})$ 
4:  $T := T_0$ 
5: while  $f(\mathbf{x}) \leq \text{threshold} \wedge 0 \leq T$  do
6:   repeat
7:      $\mathbf{G}' := \mathbf{G} + \text{move\_size} * \text{random\_unit\_vector}$ 
8:   until  $C(\mathbf{G}')$ 
9:    $\Delta := f(\mathbf{G}') - f(\mathbf{G})$ 
10:  if  $\Delta < 0 \vee \text{rand}(0, 1) \leq \exp(-\Delta/T)$  then
11:     $\mathbf{G} := \mathbf{G}'$ 
12:  end if
13:   $T := T - \delta T$ 
14: end while

```

---

Rejection sampling is implemented by repeating the generation of  $\mathbf{x}'$  in the SA algorithm 2 (line 4) until the constraint is satisfied.

Let  $\mathbf{G}$  be the relevant vector of Fourier coefficients  $\{a_{k,m}, b_{k,m}\}$ . Let  $C()$  be the Boolean-valued constraint on the Fourier coefficients. Then algorithm 3 describes our rejection sampling constrained SA method.

We use constrained SA to search for the Fourier coefficients for each of the control functions. We impose the following fidelity constraint:

$$\phi = \int_0^\tau g_k^2(t) dt = \frac{1}{\tau} \sum_{n=0}^M (a_n^2 + b_n^2) \leq \kappa^2 \quad (12)$$

and we arbitrarily choose  $\kappa = 1$  for convenience.

#### 4.4 The Fitness Function

We analyse the effectiveness of constrained SA in the discovery of time optimal control functions that result in a spin-chain systems implementing a permutation

of set of (orthonormal) basis vectors. This can be achieved by considering functionals of the form:

$$F[\mathbf{g}, \tau] = \Re \text{Tr}(\hat{P}^\dagger \hat{U}_\tau) - \tau^2 \quad (13)$$

where  $P$  is a permutation matrix (see § 3.2). Here we are maximising  $F$ ; the  $\tau^2$  term is included to ‘punish’ slow implementations and reward faster ones.

In order to find time optimal solutions, we vary the Fourier coefficients of the control functions as described in algorithm 3; we also vary  $\tau$  (the time the spin chain’s dynamics is evolved for in order to calculate the fitness function) by a random amount between  $-0.001$  and  $0.001$  with each iteration of SA. This is performed at the same step of the algorithm as varying the Fourier coefficients (algorithm 3, line 7). A similar method of rejection sampling is used to prevent  $\tau$  becoming negative.

The simulation was performed using a standard geometric integrator build into the Matlab symbolic math package.

In the SA process, the termination criterion was chosen so that the algorithm would terminate only if a fitness of  $F \geq 0.7$  was achieved (eqn.(13)), and a fidelity of  $\phi \geq 0.92$  was achieved (eqn.(12)), or 500 iterations had been performed.

## 5 Results

We present the results for six specific permutations (§3.2) in table 1. These demonstrate all the behaviours exhibited by the SA algorithm during our experiments.

Throughout our investigation we choose the Fourier band limit (eqn.(2)) to be  $M = 5$ .

### 5.1 Two Qubits, Each with Two Orthogonal Control Fields

In the two control field case ( $h_n \neq 0$ ), we found the algorithm to exhibit a variety of behaviours with the parameters chosen. We saw three types of run:

1. The matrices which achieved a fidelity of greater than 0.92 converged quickly (in about 200 iterations). The reduction of the gate implementation time as the SA proceeded did not seem to limit the progress towards a high fidelity control scheme. This suggests that the algorithm did not find solutions approaching the time optimal ones, as no trade off between fidelity and time optimality was observed. An example graph of the progress of fidelity during such a run of SA can be seen in fig.1, and for the progress of implementation time in fig.2.
2. Some of the matrices which did not achieve a fidelity of greater than 0.92 seemed to stagnate in their progress to increasing fidelity as the implementation time reached a critical low point. These ‘stagnation points’ occur at a fidelity of around 0.5 – 0.6; after this little progress was made before the algorithm timed out. This suggests that a point was reached where time optimality and fidelity were in direct conflict. An example graph of the progress of fidelity during such a run of SA can be seen in fig.3; and for the progress of implementation time in fig.4.

**Table 1.** 2-qubit results with one and two control fields per qubit

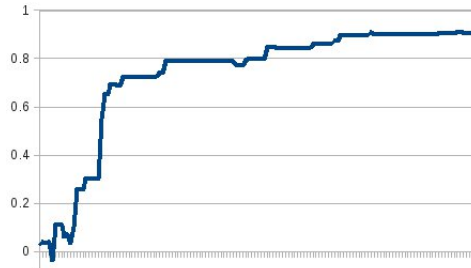
$P$	1 control field ( $g$ )		2 control fields ( $g, h$ )	
	max fidelity $\phi$	min time $\tau$	max fidelity $\phi$	min time $\tau$
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	0.057	0.499	0.540	0.456
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	0.563	0.564	0.553	0.486
$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	–	–	0.92	0.679
$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	–	–	0.587	0.592
$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$	–	–	0.92	0.693
$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$	0.078	0.854	0.92	0.693

3. Other matrices which did not achieve a fidelity of greater than 0.92 seemed to be progressing but simply too slowly to achieve a suitable result before the 500 iteration timeout. Further work will include extending the timeout, with better “stagnation” detection.

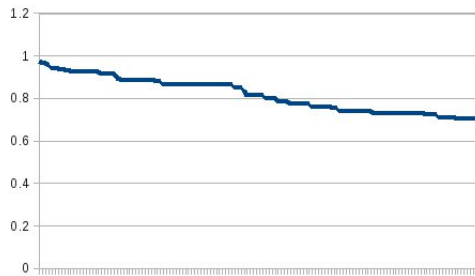
This collection of behaviours suggest that a modification to the fitness function is needed. High fidelity is essential for the implementation of quantum gates, and should be given a higher precedence over time optimality. A fitness function that rewards both speed and high fidelity, but without allowing time optimality to compromise fidelity, is needed. The search for such a fitness function will also be the basis for further work.

Some of the optima found in the two control fields per qubit cases are close to the global ones (in terms of the fidelity alone, not time optimality); in future work the control schemes found with the SA method will be compared explicitly to theoretical optima for time optimality. The cases this applies to are those with two control fields per qubit which reached a fidelity of 0.92 or greater.

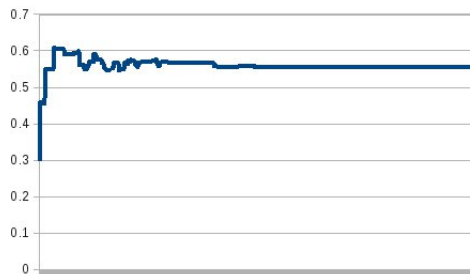
The results attempt to provide a minimum implementation time, given the energy constraint or at least to demonstrate that the SA method is capable or providing such information about a system. This will allow us to investigate the tightness of speed limit bounds for specific quantum operations.



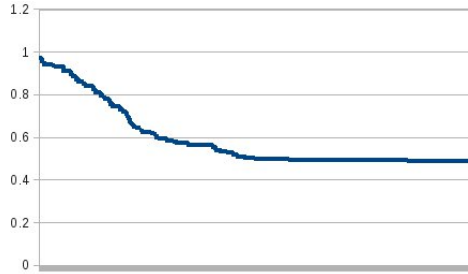
**Fig. 1.** Progress of fidelity  $\phi$  in a two field case where convergence occurred



**Fig. 2.** Progress of implementation time  $\tau$  in a two field case where fidelity convergence towards 1 occurred



**Fig. 3.** Progress of fidelity  $\phi$  in a two field case where stagnation occurred



**Fig. 4.** Progress of implementation time  $\tau$  in a two field case where stagnation occurred

## 5.2 Two Qubits, Each with a Single Control Field

In the case of two qubits each with a single control field, the results were found to be less promising, and no fast, high fidelity schemes were found. What was observed was either the SA converged to a solution with fidelity far from one, or it failed to converge, with fidelity and implementation times remaining close to their initial values until timeout (cases where this happened are marked – in the results table 1). This may be because this system in question is simply not ‘controllable’, and cannot implement the gate required, with the single field constrained control functions of the form used. However, it is difficult to tell this situation apart from a failure of the algorithm due to bad choice of parameters.

In the cases where convergence occurred, there was a similar pattern of search behaviours as in the two control field case. Either a point where time optimality was hindering progress towards a higher fidelity solution was reached and progress stagnated, or fidelity began to converge (frequently at about 0.5) as the implementation continued to decrease the time, unhampered by the convergence of fidelity. In the later cases the algorithm timed out, and it is apparent that a higher timeout value is needed to probe this scenario further.

## 6 Conclusions and Further Work

We have introduced a form of SA, constrained SA, suitable for searching for optimal solutions under constraints. We have found that this constrained SA shows potential as an effective method for discovering optimal control schemes for Heisenberg spin chains subject to constrained control functions. We have also found that it opens many directions for future work. The approach is readily extensible to other quantum computational systems, and to other constraints and could be a good tool for studying constrained, quantum time optimal control.

The numerical results additionally provide information on the implementability of specified gates using particular control schemes.

Next steps include:

1. a direct comparison of constrained SA to GRAPE
2. a comparison of the numerically discovered implementation time of optimal control schemes to theoretical speed limit bounds
3. an investigation of the role of the Fourier band limit  $M$  on the time optimality of control schemes discovered by constrained SA
4. an investigation of other relevant physical constraints
5. design of a better fitness function for SA so that fidelity is never compromised.

**Acknowledgments.** We thank Simon Poulding for many helpful discussions. Russell is supported by an EPSRC DTA grant.

## References

1. Altafini, C., Ticozzi, F.: Modeling and Control of Quantum Systems: An Introduction. *IEEE Transactions on Automatic Control* 57, 1898–1917 (2012)
2. Hsieh, M., Rabitz, H.: Optimal control landscape for the generation of unitary transformations. *Phys. Rev. A* 77, 042306 (2008)
3. Laarhoven, P.J.M., Aarts, E.H.: *Simulated Annealing: Theory and Applications*. Springer (1987)
4. Lloyd, S.: Ultimate physical limits to computation. *Nature* 406(6799), 1047–1054 (1999)
5. Margolus, N., Levitin, L.B.: The maximum speed of dynamical evolution. *Physica D* 120, 188–195 (1998)
6. Mohn, P.: *Magnetism in the Solid State: An Introduction*. Springer (2003)
7. de B. Robinson, G.: *Representation theory of the symmetric group*. Edinburgh University Press (1961)
8. Russell, B., Stepney, S.: Geometric methods for analysing quantum speed limits: Time-dependent controlled quantum systems with constrained control functions. In: Mauri, G., Dennunzio, A., Manzoni, L., Porreca, A.E. (eds.) *UCNC 2013. LNCS*, vol. 7956, pp. 198–208. Springer, Heidelberg (2013)
9. Schulte-Herbruggen, T.: Controlling quanta under constraints (2007), presentation <http://www.impan.pl/BC/Arch/2007/CCQ/Schulte-Herbrueggen.pdf>
10. Schulte-Herbruggen, T.: Quantum compilation by optimal control of open systems (2007), presentation [http://www.physik.uni-siegen.de/quantenoptik/events/marialaach07/schulte-herbrueggen\\_marialaach07.pdf](http://www.physik.uni-siegen.de/quantenoptik/events/marialaach07/schulte-herbrueggen_marialaach07.pdf)
11. Werschnik, J., Gross, E.K.U.: *Quantum Optimal Control Theory*. ArXiv e-prints (July 2007), arXiv:0707.1883 (quant-ph)
12. Zwierz, M.: Comment on “Geometric derivation of the quantum speed limit”. *Physical Review A* 86, 016101 (2012)



# Combinatorial Optimization in Pattern Assembly<sup>\*</sup>

## (Extended Abstract)

Shinnosuke Seki

Helsinki Institute for Information Technology (HIIT),  
Department of Information and Computer Science, Aalto University, P.O. Box 15400,  
FI-00076, Aalto, Finland  
`shinnosuke.seki@aalto.fi`

**Abstract.** Pattern self-assembly tile set synthesis (PATS) is an NP-hard combinatorial problem to minimize a rectilinear tile assembly system (RTAS) that uniquely self-assembles a given rectangular pattern. For  $c \geq 1$ ,  $c$ -PATS is a subproblem of PATS which takes only the patterns with at most  $c$  colors as input. We propose a polynomial-time reduction of 3SAT to 60-PATS in order to prove that 60-PATS is NP-hard.

## 1 Introduction

Tile self-assembly is an algorithmically rich model of “programmable crystal growth.” Well-designed molecules (square-like “tiles”) with specific binding sites can deterministically form a single target shape even subject to the chaotic nature of molecules floating in a well-mixed chemical soup. Such tiles were experimentally implemented as DNA double-crossover molecules in 1998 [7].

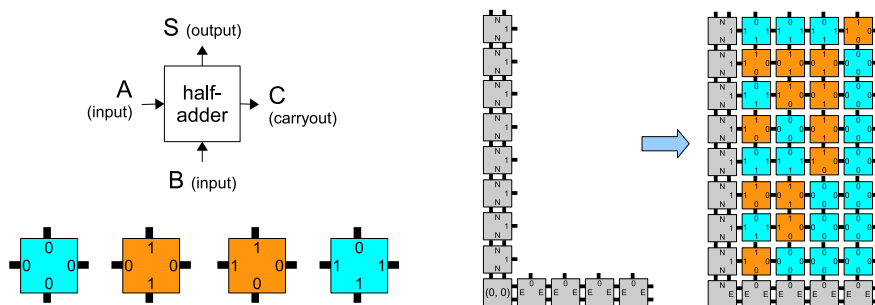
Shape-building is one primary goal of self-assembly; pattern-painting is another. Based on the abstract Tile Assembly Model (aTAM) introduced by Winfree [6], Ma and Lombardi have first shed light on this problem and formalized it in the name of *patterned self-assembly tile set synthesis* (PATS) in [3,4]. PATS aims at minimizing the number of tile types necessary for a rectilinear TAS (RTAS) to uniquely assemble a given rectangular pattern.

PATS was recently proved to be NP-hard [1]. Nevertheless, it is not until the number of colors available for input patterns is upperbounded by some constant  $c \geq 1$  that PATS gets practically meaningful, as summarized in DNA 18 as: “*any given logic circuit can be formulated as a colored rectangular pattern with tiles, using only a constant number of colors.*” We call this variant the  $c$ -PATS. The main contribution of this paper is to prove the next theorem.

**Theorem 1.** *60-PATS is NP-hard.*

---

<sup>\*</sup> The author appreciates valuable comments from Ho-Lin Chen, David Doty, and the anonymous referees on the earlier version of this paper. This work is financially aided by HIIT Pump Priming Grant No. 902184/T30606.



**Fig. 1.** (Left) Four tile types implement together the half-adder with two inputs  $A$ ,  $B$  from the west and south, the output  $S$  to the north, and the carryout  $C$  to the east. (Right) Copies of the “half-adder” tile types turn the L-shape seed into the binary counter pattern, whose origin is at the bottom-left corner as illustrated.

(2-PATS was claimed NP-hard [4], but the proof was incorrect, as noted in [1]). A file including all figures in this paper in color can be found at the website of the author (<http://users.ics.aalto.fi/sseki>).

## 2 Rectilinear TAS and $c$ -PATS

This section begins with a terse explanation of rectilinear Tile Assembly System (RTAS) using examples. Then we define  $c$ -PATS. An excellent introduction to the fundamental model called the abstract Tile Assembly Model is in [5].

A (*rectangular*) *pattern* (of width  $w$  and height  $h$ ) is a function from the rectangular domain  $\{(x, y) \mid x \in \{0, 1, \dots, w - 1\}, y \in \{0, 1, \dots, h - 1\}\}$  to  $\mathbb{N}$ . We denote the image of a pattern  $P$  by  $\text{color}(P)$ . That is, any color in  $\text{color}(P)$  appears at least once on  $P$ . We say that  $P$  is  $k$ -*colored* if  $|\text{color}(P)| \leq k$ .

The self-assembly of binary counter (Fig. 1) illustrates how a rectilinear TAS (RTAS) works. A *tile type* is a square of some color whose four sides are *labeled*. Hence a tile type is identified by its color and four labels, which are read in the counter-clock wise order starting at north (N); for instance, the second orange tile type in Fig. 1 (Left) is (1, 1, 0, 0, orange). Given a tile type  $t$  and a direction  $d \in \{N, W, S, E\}$ , we denote by  $t(d)$  the label at  $d$ . Assume that there is an infinite supply of copies of the four tile types (two blue, two orange) in Fig. 1; the copies are simply called *tiles*. Using them, the RTAS tiles the plain delimited by the L-shape (gray) structure called *seed* according to the following rule.

**RTAS’s Tiling Rule:** A tile can attach at a position  $(x, y)$  if its west label matches the east label of the tile on  $(x - 1, y)$  and its south label matches the north label of the tile on  $(x, y - 1)$ .

This rule suggests that it is not until its west and south neighbors are tiled that a position becomes attachable. At the initial time point, therefore, the sole attachable position is (1, 1). See the L-shape seed in Fig. 1; a tile of type (1,

1, 0, 0, orange) can attach at (1, 1), while no tile of other three types can due to label-mismatching. The attachment makes the two positions (1, 2) and (2, 1) attachable. In this manner, the tiling proceeds from south-west to north-east *rectilinearly* until no attachable position is left. After the attachment thus terminates, the resulting pattern is considered the output and called *terminal pattern (assembly)*. The  $5 \times 9$  binary counter pattern in Fig. 1 is terminal.

A *rectilinear TAS* (RTAS) is a pair  $\mathcal{T} = (T, \sigma_L)$  of a set  $T$  of tile types and an L-shape seed  $\sigma_L$ . Its *size* is the cardinality of  $T$ . An RTAS is *directed* if there are no distinct tile types  $t_1, t_2 \in T$  such that  $t_1(\mathbb{W}) = t_2(\mathbb{W})$  and  $t_1(\mathbb{S}) = t_2(\mathbb{S})$  (the directedness of TASs is defined in a different but equivalent way). It is clear from the definition that a directed RTAS generates a unique terminal pattern, and in this case, we say that it *uniquely self-assembles the pattern*.

The *pattern self-assembly tile set synthesis* (PATS), proposed by Ma and Lombardi [3], aims at computing the minimum size directed RTAS that uniquely self-assembles a given rectangular pattern. (The solution to PATS is required to be directed here, but not originally. This, however, does not change the problem, as being observed in [2].) PATS is an NP-hard problem [1]. By restricting the number of colors  $c$  used to draw input patterns, a practically-meaningful subproblem of PATS is formulated as follows.

**Definition 1.** *c*-COLORED PATS (*c*-PATS)

GIVEN: a *c*-colored pattern  $P$ ;

FIND: a smallest directed RTAS that uniquely self-assembles  $P$ .

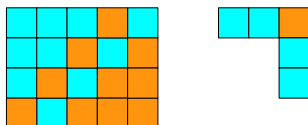
In Sect. 4, we will prove that 60-PATS is NP-hard. This is done by a polynomial-time reduction of 3SAT to the decision variant of 60-PATS: given a 60-colored pattern and an integer  $n$ , decides whether there is a directed RTAS with at most  $n$  tile types that uniquely self-assembles the given pattern.

### 3 Basic Combinatorial Results

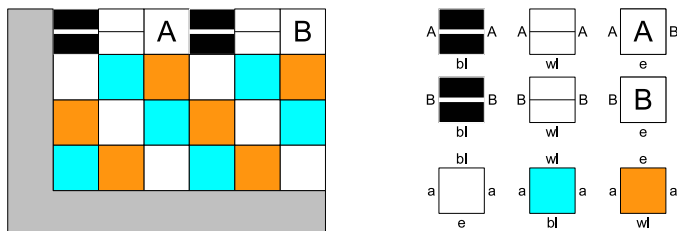
Let us show several basic results on directed RTASs, which will be used later in order to prove Theorem 1. Recall that an RTAS is directed if and only if it contains no distinct tile types  $t_1, t_2$  with  $t_1(\mathbb{W}) = t_2(\mathbb{W})$  and  $t_1(\mathbb{S}) = t_2(\mathbb{S})$ . This allows us to design a simple pattern  $P$  such that in order for *any* directed RTAS to uniquely self-assemble a pattern including  $P$ , the RTAS must contain *at least* 2 tile types of some specific color. We introduce several such patterns below.

The binary counter pattern  $P_{bc}$  (Fig. 1) contains one of such pattern  $P$ . At the positions (2, 2) and (4, 2) of distinct colors, an RTAS  $\mathcal{T}$  must put tiles of distinct types  $t_1, t_2$  (there is no chameleon tile type). Moreover, in order for  $\mathcal{T}$  to be directed, either  $t_1(\mathbb{W}) \neq t_2(\mathbb{W})$  or  $t_1(\mathbb{S}) \neq t_2(\mathbb{S})$  must hold. The west and south neighbors of these positions being all blue, the label discrepancy implies that  $\mathcal{T}$  contains blue tiles that disagree with each other on either east or north labels, and hence, are of distinct types. This is formalized as follows.

**Lemma 1.** *Let  $\mathcal{T}$  be a directed RTAS that uniquely self-assembles a pattern  $P$ . For a color  $i$  and positions  $(x_1, y_1), (x_2, y_2)$ , if  $P(x_1-1, y_1) = P(x_1, y_1-1) =$*



**Fig. 2.** (Left) If this blue-orange subpattern is on a pattern  $P$ , a directed RTAS needs at least 2 blue tile types and at least 2 orange ones in order to uniquely self-assemble  $P$ ; (Right) This pattern has a directed RTAS contain at least 2 blue tile types.



**Fig. 3.** A 3-mosaic pattern  $P_{\text{mos}(3)}$  (the gray part can be regarded as the seed). Using the 9 tile types shown right, an RTAS can uniquely self-assemble this pattern, and this is the sole minimum tile set.

$P(x_2-1, y_2) = P(x_2, y_2-1) = i$  but  $P(x_1, y_1) \neq P(x_2, y_2)$ , then  $\mathcal{T}$  has at least two tile types of color  $i$ .

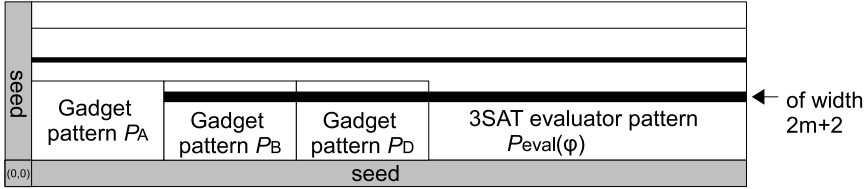
**Corollary 1.** *If a directed RTAS uniquely self-assembles a pattern on which the pattern shown in Fig. 2 (Left) appears, then it has at least 2 blue tile types and at least 2 orange tile types.*

Another pattern of interest is shown in Fig. 2 (Right). With tiles of the same type  $t$  at all the four blue positions,  $t$  would satisfy  $t(W) = t(E)$  and  $t(N) = t(S)$  and hence a blue tile (of this type) would fill the orange position. Thus, in order for a directed RTAS to assemble a pattern including this pattern, at least 2 blue tile types are needed. This observation is formally described as follows.

**Lemma 2.** *Let  $\mathcal{T}$  be a directed RTAS that uniquely self-assembles a pattern  $P$ . For a color  $i$  and a position  $(x, y)$ , if  $P(x-2, y) = P(x-1, y) = P(x, y-1) = P(x, y-2) = i$  but  $P(x, y) \neq i$ , then  $\mathcal{T}$  has at least two tile types of color  $i$ .*

**Mosaic Pattern.** plays the critical role in the proof of Theorem 1. For  $k \geq 1$ , a  $k$ -mosaic pattern  $P_{\text{mos}(k)}$  has the property that if a directed RTAS uniquely self-assembles a pattern including  $P_{\text{mos}(k)}$ , then there are at least  $k - 1$  colors in  $\text{color}(P_{\text{mos}(k)})$  such that the RTAS contains at least 2 tile types of the color.

Fig. 3 shows a 3-mosaic pattern. It contains 7 colors: white, blue, orange, lined-black, lined-white, A, and B. Assume that the gray part allows us to encode any information. How many tile types are necessary and sufficient for a directed RTAS  $\mathcal{T}$  to uniquely self-assemble it? Trivially, 7 types are necessary as there is no chameleon tile type. The key issue is that in order for tiles  $t_A$  and  $t_B$  to attach



**Fig. 4.** The pattern  $P(\phi)$  to which a 3SAT instance  $\phi$  is reduced. The 3 black lines represent variable wires, and the lowest one of width  $2m + 2$  is a fake one. Note that this one stems from the right of  $P_A$ , while the others originate from the seed.

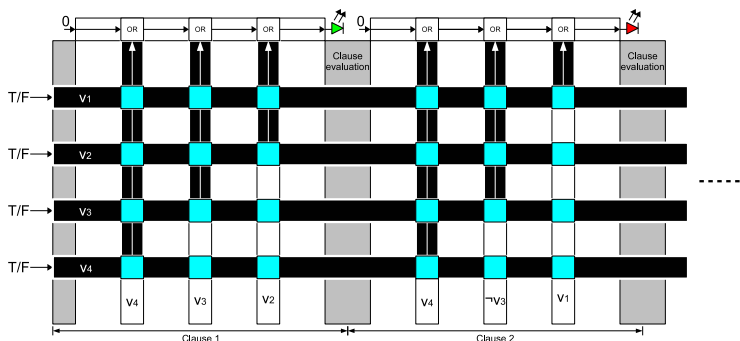
at the positions **A** and **B** exclusively, they must receive different labels either from the south or from the west. In other words,  $\mathcal{T}$  must carry a 1-bit signal as labels to these positions. Fig. 3 demonstrates that 9 types are enough by having 2 lined-black tile types and 2 lined-white ones carry the signal rightward (to be **A** or not to be). The mosaic region is “thicker” in the sense that carrying the signal through it costs more tile types. Indeed, no matter which path in the region is taken, one encounters all of white, blue, orange positions. If it were not for two white tile types, the signal is lost at the diagonal white stripes, and this argument is valid also for blue and orange. As a result, extra 3 tile types would be needed. As such, not only coloring (how many tile types to be drawn by each color) but also label allocation is *uniquely* determined (up to renaming) for smallest tile type sets. This unique label allocation plays a critical part in the proof of Theorem 1.

### 4 Polynomial-Time Reduction of 3SAT to 60-PATS

Our proof of Theorem 1 takes the classic approach: a polynomial-time many-one reduction from 3SAT to the decision variant of 60-PATS with  $n = 60 + 24$ . A given instance of 3SAT is a formula  $\phi$  that is a conjunction of clauses consisting of exactly three literals (a variable or its negation); the  $m$  variables of  $\phi$  are indexed as  $v_1, v_2, \dots, v_m$ . We will propose a pattern  $P(\phi)$  to which  $\phi$  is reduced and a set  $T$  of  $60 + 24$  tile types with the following properties:

1. For any  $t_1, t_2 \in T$ ,  $t_1(W) \neq t_2(W)$  or  $t_1(S) \neq t_2(S)$  holds;
2.  $P(\phi)$  is a snapshot of the circuit called the *3SAT evaluator* when it evaluates  $\phi$  to be true according to an assignment  $\mathbf{b}$  given as input. Tiles in  $T$  enable a directed RTAS to simulate the evaluation, which results in the unique self-assembly of  $P(\phi)$  from the L-shape seed encoding  $\phi$  and  $\mathbf{b}$ .
3. Any directed RTAS with less than  $60 + 24$  tile types cannot uniquely self-assemble  $P(\phi)$ .
4. If a directed RTAS  $(T', \sigma'_L)$  with  $|T'| = 60 + 24$  uniquely self-assembles  $P(\phi)$ , then  $T'$  is isomorphic to  $T$  (up to label renaming) and  $\sigma'_L$  encodes  $\phi$ .

From the property 2, if an assignment  $\mathbf{b}$  satisfies  $\phi$ , then the L-shape seed that encodes  $\mathbf{b}$  and  $\phi$  uniquely self-assembles into  $P(\phi)$  using tiles in  $T$  of size  $60 + 24$ .



**Fig. 5.** The 3SAT evaluator for the specific 3SAT instance  $(v_4 \vee v_3 \vee v_2) \wedge (v_4 \vee \neg v_3 \vee v_1)$ . Literals (white stripes) are evaluated at substituters (blue squares), and the evaluation (black stripes with a white arrow) is transmitted upward for the evaluation of clause they belong to. The design principle works for arbitrary number of variables or clauses. LED signals at the top indicate that the input satisfies Clause 1 but not Clause 2.

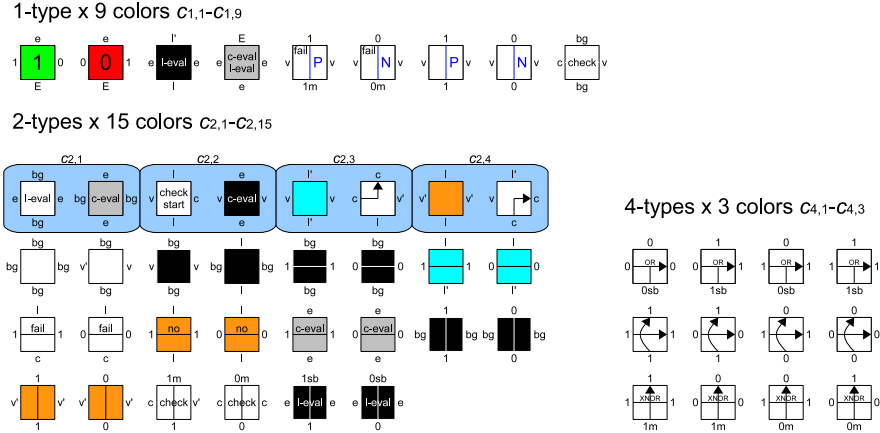
Moreover this RTAS is directed due to 1. Conversely, 3 and 4 mean that the sole minimum (60+24 tile types) tile type set for  $P(\phi)$  is  $T$ , and in order for a directed RTAS with  $T$  to uniquely self-assemble  $P(\phi)$ , its seed must encode  $\phi$  properly. We design  $T$  so as for its tile types to evaluate  $\phi$  according to an assignment encoded on the L-shape seed. Since  $P(\phi)$  is a snapshot for  $\phi$  to be thus evaluated *true*, the existence of such a minimum RTAS implies the satisfiability of  $\phi$ . This is an overview of the proof. We are going to see more details.

A blueprint of  $P(\phi)$  in Fig. 4 shows that  $P(\phi)$  is composed of four subpatterns: 3SAT evaluator pattern  $P_{eval}(\phi)$  and three gadget patterns  $P_A, P_B, P_D$ , which play an auxiliary role in verifying the properties 3 and 4 above.

### 4.1 3SAT Evaluator Pattern

Let us begin with the 3SAT evaluator pattern. It is designed based on a digital circuit called the *3SAT evaluator for  $\phi$*  (see Fig. 5). Just as the name suggests, it evaluates  $\phi$  according to a given assignment. It must be noted first that this circuit is designed to be planar (no wire goes over the others) and rectilinear (signals always transmit from south-west to north-east). These properties enable a directed RTAS to simulate this circuit, using tiles in  $T$ .

As input, the 3SAT evaluator for  $\phi$  takes from the left an assignment  $\mathbf{b} = (b_1, \dots, b_m)$  according to which  $\phi$  is evaluated, where  $b_1, \dots, b_m \in \{0, 1\}$  (0:false, 1:true). It is provided with  $m$  horizontal wires, which transmit the input rightward along with the index  $i$  of the variable they represent. It is also provided with vertical wires, which represent literals in  $\phi$ . We refer to the horizontal wire for variable  $v_i$  simply as the *variable wire*  $v_i$ , and a vertical wire for literal  $v_j$  (resp.  $\neg v_j$ ) as a *literal wire*  $v_j$  (resp.  $\neg v_j$ ). Any variable wire is connected with any literal wire at their intersection by a device called *substituters*. At the intersection of the variable wire  $v_i$  and a literal wire  $v_j$  or  $\neg v_j$ , the substituter



**Fig. 6.** The 51 tile types (of 27 colors) to simulate the 3SAT evaluator by an RTAS. Two tile types surrounded by the blue rounded rectangle are of the same color, though they are drawn with different colors in order to clarify their roles.

compares their indices, and if  $i = j$ , it substitutes the Boolean value  $b_i$  into the literal by an XNOR gate and transmits the result to the north; otherwise, it does nothing but merely “lets one go over the other.” For each clause, its three literals thus evaluated are conjugated by three OR gates, which determine if the clause is satisfied or not, and the evaluation is output at the top as LED signal (red:unsatisfied, green:satisfied; see Fig. 5).

Now we propose a tile type set  $T_{eval}$  (Fig. 6) with which a directed RTAS simulates the 3SAT evaluator for  $\phi$  according to an assignment  $\mathbf{b}$ . It consists of 51 tile types of 27 colors: 1 type of each of 9 colors  $c_{1,1}, c_{1,2}, \dots, c_{1,9}$ , 2 types of each of 15 colors  $c_{2,1}, \dots, c_{2,15}$ , and 4 types of each of 3 colors  $c_{4,1}, c_{4,2}, c_{4,3}$ . The assignment  $\mathbf{b} = (b_1, \dots, b_m)$  and instance  $\phi$  are encoded on the vertical and horizontal axes of L-shape seed, respectively (see Fig. 7). Observe that a variable  $v_i$  is encoded with its assigned value  $b_i$  as  $b_i v^{2^i}$  while the literal  $v_j$  and its negation are encoded as  $1^{2^j} 1$  and  $1^{2^j} 0$  (1:positive, 0:negative), respectively. There are two things to note. One is that on the vertical axis of the seed, the variables  $v_1, \dots, v_m$  are encoded in this order from the top. The other is that below the wire  $v_m$  is another wire of width  $2m + 2$  (see Fig. 4), which encodes no variable but prevents us from cheating the reduction, as stated later.

Let  $P_{eval}(\phi, \mathbf{b})$  be the pattern that tiles in  $T_{eval}$  self-assembles from the seed (see Fig. 7). On it, any literal wire  $v_j$  encounters the  $m$  variable wires. At its intersection with the variable wire  $v_i$ , a pattern that visualizes the mechanism of substituter in the 3SAT evaluator self-assembles. See Fig. 8 (Middle). When the wires meet, the **check-start** tile attaches and triggers the assembly of diagonal zig-zag snake. When the snake hits the right periphery of the literal wire, one of the 2 **check** tile attaches, and if the variable wire  $v_i$  is waiting to its north with the input  $b_i$  (this happens if and only if  $i = j$ ), one of the 4 **XNOR** tiles selectively attaches, substitutes  $b_i$  into the literal, and outputs the result to the

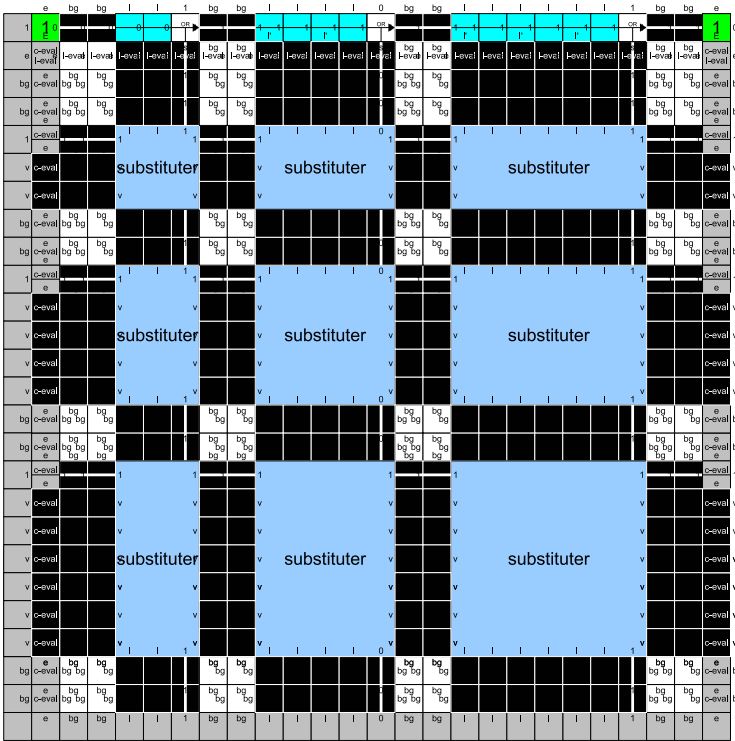


Fig. 7. Using tiles in the set  $T_{eval}$ , this pattern uniquely self-assembles from the L-shape seed that encodes a 3SAT instance with 3 variables  $v_1, v_2, v_3$  and a clause  $\{v_1, \neg v_2, v_3\}$  with the assignment  $(1, 1, 1)$ .

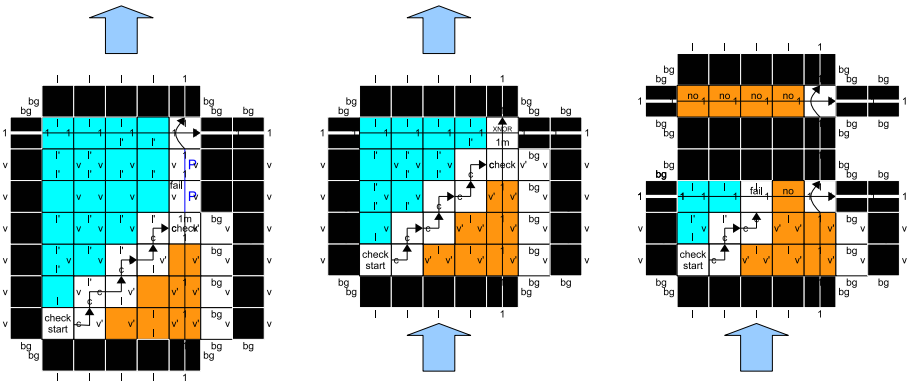


Fig. 8. Patterns occurring at the intersection of the literal wire  $v_2$  (Left) with the variable wire  $v_3$ , (Middle) with the wire for matching variable  $v_2$ , or (Right) with the variable wire  $v_1$  and with the thinnest possible wire, which encodes no variable.



north (1:true, 0:false; note that before and after the substitution, the signal 1/0 through a literal wire is interpreted in different ways). Fig. 8 also shows how the literal wire goes over the variable wire without being substituted in other two cases of  $i > j$  (Left) and  $i < j$  (Right). Until being substituted, the literal wire  $v_j$  has already crossed the variable wires  $v_m, v_{m-1}, \dots, v_{j+1}$ , which are thicker than itself, and it is going to cross the remaining variable wires  $v_{j-1}, \dots, v_1$  in this order, which are thinner, while carrying the substituted value. In its crossing a thicker variable wire, tile types P and N (of color  $c_{1,5}, \dots, c_{1,8}$ ) *visually* tell which of 1/0 signal it carries, while its crossing a thinner one leaves no such visual clue. Two facts of significance follow from this property. One is that any literal wire encounters the thicker “fake” variable of width  $2m + 2$  before being substituted and leaves the visual evidence of whether the literal is positive or negative (this information is encoded on the seed as labels, that is, invisibly). The other is that the encounter order conceals from  $P_{\text{eval}}(\phi, \mathbf{b})$  any hint of what the encoded assignment  $\mathbf{b}$  is. OR tile types evaluate clauses as shown in Fig. 7.

The *only* positions on  $P_{\text{eval}}(\phi, \mathbf{b})$  whose color depends on the encoded assignment  $\mathbf{b}$  are the LED positions. Redrawing all these positions by green (satisfied) yields the pattern  $P_{\text{eval}}(\phi)$ . Further combining it with the three gadget patterns  $P_A, P_B, P_D$  gives the pattern  $P(\phi)$ . Besides the 27 main colors on  $P_{\text{eval}}(\phi)$ , 33 auxiliary colors appear on the gadget patterns (A1-11 and AB1-4 on  $P_A$ , B1-9 and BC1 on  $P_B$ , and D1-7 and D-bg on  $P_D$ ), and in total, 60-colors are on  $P(\phi)$ . The set  $T$  of  $60 + 24$  tile types we propose is the union of  $T_{\text{eval}}$  and another set  $T_{\text{aux}}$  of 33 tile types of these auxiliary colors (one type per color). The space is not enough to illustrate how a directed RTAS  $(T, \sigma_L)$  uniquely self-assembles the whole pattern  $P(\phi)$ . Instead, Figs. 7, 8, 9, and 10 exhibit how tile types in  $T$  self-assemble its essential components. It must be noted that  $P_A$ , being thus assembled, produces the fake variable wire of width  $2m + 2$  below other variable wires, as shown in Fig. 4.

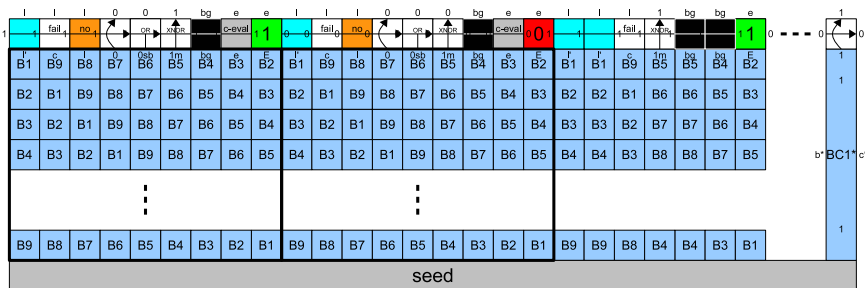
Note that an unsatisfiable  $\phi$  may admit another set of  $60 + 24$  tile types with which a directed RTAS uniquely self-assembles  $P(\phi)$ . The gadget patterns rule out this possibility and show that assemblability implies satisfiability.

## 4.2 Assemblability Implies Satisfiability

In the rest, we assume that the satisfiability of  $\phi$  is not sure. Imagine then that you are given  $60+24$  *uncolored* tile types, and asked to color them and allocate labels to them so as for a directed RTAS with the resulting tile type set to uniquely self-assemble  $P(\phi)$ . Then we will observe that this is possible only when  $\phi$  is satisfiable. This completes the proof of Theorem 1.

First of all, at least one tile type per color is trivially needed. On  $P(\phi)$ , we then find subpatterns to which Lemmas 1, 2, and Corollary 1 are applied to draw 6 of the remaining 24 uncolored ones by  $c_{2,1}, \dots, c_{2,4}, c_{2,5}$  (white), and  $c_{2,6}$  (black) (for instance, a subpattern for Corollary 1 is on the substituter pattern in Fig. 8; compare it with Fig. 2).

The three gadget patterns are modifications of the mosaic pattern introduced in Sect. 3. Their role is to cooperatively force us to draw the 18 uncolored



**Fig. 9.** A part of the gadget pattern  $P_B$ . Thick black rectangulars indicate mosaics.

tile types in the same way as  $T_{eval}$ , and furthermore, to allocate labels in the isomorphic manner to  $T_{eval}$  (see Fig. 6). More specifically,

$P_A$ : (This and  $P_B$  play analogous roles. Since the space is limited and  $P_A$  is vertically long, we can only present a part of  $P_B$  in Fig. 9.)  $P_A$  is responsible for the coloring and label allocation of tile types for upward signal transmission (those of colors  $c_{2,12}, \dots, c_{2,15}, c_{4,1}, c_{4,2}, c_{4,3}$  in Fig. 6). Due to the property of mosaic patterns,  $n_a \geq 7$  of the 18 uncolored tile types are to be drawn by some of these 7 main colors and A1-8. Moreover, the number  $n_a$  would become at least 8 (one uncolored tile type wasted) unless the choice is for the 7 main colors and labels are allocated so as to transmit 0/1-signal upward as corresponding tile types in  $T_{eval}$  do (ref. Fig. 3). The isomorphic label allocation will not have been settled until  $P_B$  and  $P_D$  are considered.

$P_B$ : See Fig. 9. This plays the analogous role to  $P_A$  but rather for tile types to transmit signals rightward (those of colors  $c_{2,7}, \dots, c_{2,11}, c_{4,1}, c_{4,2}, c_{4,3}$  in Fig. 6). Among the remaining (at most 11) uncolored tile types,  $n_b \geq 8$  are to be drawn by some of the main 8 colors and B1-9. This has the preference for the main colors as  $P_A$ ; that is, unless the choice is for the 8 main colors and labels are allocated so as for them to transmit 0/1-signal horizontally,  $n_b$  would become at least 9. The right half of  $P_B$  is responsible for the label allocation, which we will discuss shortly.

$P_D$ : See Fig. 10. After  $P_A$  and  $P_B$  have been taken into account, at most 3 tile types remain uncolored. We claim that  $P_D$  needs 3 uncolored tile types no matter how we have drawn as well as allocated glues according to  $P_A$  and  $P_B$ , and they are to be drawn with the colors  $c_{4,1}, c_{4,2}, c_{4,3}$ . Hence, the consumption of uncolored tile types due to  $P_A$  and  $P_B$  must have been minimized, reflecting the preference of  $P_A, P_B$  for the  $T_{eval}$ 's coloring and glue allocation. In summary,  $c_{4,1}, c_{4,2}, c_{4,3}$  each is used to draw 4 tile types (in total 12 tile types),  $c_{2,1}, \dots, c_{2,15}$  each is used to draw 2 tile types (in total 30 tile types), and the other colors including the auxiliary ones appear on sole tile type.

**Label Allocation.** Having completed the coloring, we briefly explain the reason why we must also allocate labels in such a way as shown in Fig. 6 at least for

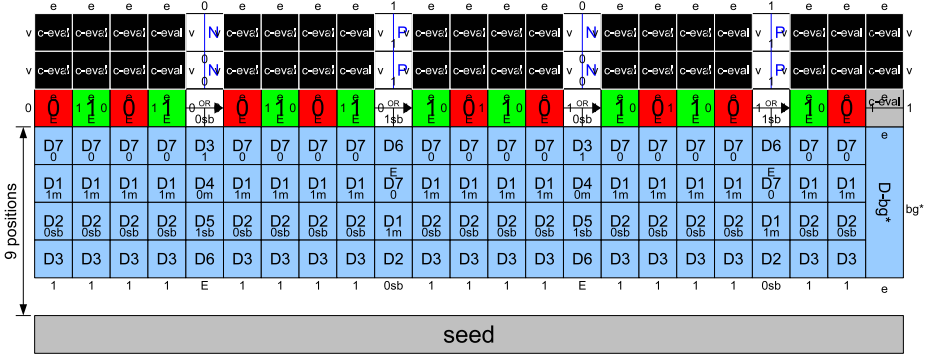


Fig. 10. A part of the gadget pattern  $P_D$

colors that are responsible for transmitting or processing 0/1 signals for 3SAT evaluation (in fact, this is enough).

For each of the colors  $c_{1,1}, \dots, c_{1,9}$  and all auxiliary ones, there is only one tile type of the color in our tile set. When a directed RTAS self-assembles  $P(\phi)$  using tiles in this set, therefore, at all positions of the color, tiles of this type attach. Let  $t_1, t_0, t_{eval}$  be the sole tile types of the colors  $c_{1,1}$  (1),  $c_{1,2}$  (0), and  $c_{1,4}$  (c-eval 1-eval), respectively. On  $P(\phi)$ , there are two  $c_{1,4}$ -colored positions whose north neighbors are colored 1 and 0, respectively. Thus,  $t_1(S) = t_0(S) = t_{eval}(N)$ . Since we are constructing a tile type set for a *directed* RTAS,  $t_1(W) \neq t_0(W)$  must hold. Let  $t_1(W) = 1$  and  $t_0(W) = 0$  with  $1 \neq 0$ . The pattern 0 1 on  $P_D$  sets  $t_0(E) = 1$ , and likewise  $t_1(E) = 0$ . In the same manner, for the tile types  $t_P, t_N$  with respective colors  $c_{1,7}, c_{1,8}$ , we have  $t_P(W) = t_N(W)$  and  $t_P(N) = t_P(S) \neq t_N(N) = t_N(S)$ .

Now these four tile types  $t_1, t_0, t_P, t_N$  allocate labels to the other tile types. For the label allocation to the 4 OR tile types (color  $c_{4,1}$ ), it should suffice to refer to Fig. 10. The remaining part of  $P_D$  (not shown in Fig. 10) is responsible for thus allocating labels to the tile types of colors  $c_{4,2}, c_{4,3}$ . The one risk to be taken into consideration resides in the 4 tile types for wire crossing (color  $c_{4,2}$ ). If the north label of the (sole) tile type of color D1 is 1 and that of the D4-colored one is 0, then the resulting 4 tile types for wire crossing certainly flips the signal vertically. In order to render this conversion harmless, we encode the variable  $v_i$  for  $1 \leq i \leq m - 1$  (except  $v_m$ ) on the seed rather as  $b_i v^{2i} 0$ ; this produces a wire at the lower end of the variable wire  $v_i$ . As such, any signal certainly goes through wire crossing *even* times, and the effect of signal flip is cancelled.

As for colors with two tile types, let us consider the two tile types  $t_{1b1}, t_{1b2}$  of lined blue color ( $c_{2,8}$ ). They must be allocated labels so as to transmit the 0/1-signal horizontally and  $t_{1b1}(S) = t_{1b2}(S)$ , due to  $P_B$ . Besides the label allocation shown in Fig. 6, there is another one:  $t_{1b1}(W) = t_{1b2}(E) = 1$  and  $t_{1b1}(E) = t_{1b2}(W) = 0$ . These two allocations are equivalent in their signal propagation ability as long as the signal propagation distance is even, and in fact, the pattern  $P(\phi)$  is designed so as to satisfy this property. The argument so far is also true for the two tile types of lined black color ( $c_{2,7}$ ).

These tile types help to allocate labels to the tile types of the colors  $c_{2,9}$ ,  $c_{2,10}$ , and  $c_{2,11}$ . See the right half of  $P_B$  (Fig. 9), where a  $c_{2,9}$ -position is sandwiched by 0 and 1 indirectly. Let  $t_{\text{fail1}}$  be the tile type that attaches there. The east label of  $t_0$ , which is 1 as mentioned above, is transmitted to the right by  $t_{1b1}, t_{1b2}$  and hence,  $t_{\text{fail1}}(\mathbf{W}) = t_0(\mathbf{E}) = 1$ . Similarly,  $t_{\text{fail1}}(\mathbf{E}) = t_1(\mathbf{W}) = 1$ . In this way,  $P_B$  also allocates labels to  $t_{\text{fail2}}$  as  $t_{\text{fail2}}(\mathbf{W}) = t_{\text{fail2}}(\mathbf{E}) = 0$ . This guarantees that an assignment signal (0/1) is not converted by them ( $P_B$  just guarantees that they carry the 1-bit signal horizontally, and cannot rule out by itself the possibility that these signals be flipping). Though being not included in Fig. 9,  $P_B$  contains an analogous pattern for two tile types of the other two colors.

**Choice of Seed.** The choice of assignment to variables, encoded on the vertical axis, is at our will. Can we choose 0/1 for each literal likewise? If so, this reduction could be fooled by encoding other 3SAT instance instead. The fake variable wire prevents us from thus cheating since at the intersection with it, the 0/1-information encoded on the seed is visually revealed as being explained before.

## References

1. Czeizler, E., Popa, A.: Synthesizing minimal tile sets for complex patterns in the framework of patterned DNA self-assembly. In: Stefanovic, D., Turberfield, A. (eds.) DNA 18. LNCS, vol. 7433, pp. 58–72. Springer, Heidelberg (2012)
2. Göös, M., Orponen, P.: Synthesizing minimal tile sets for patterned DNA self-assembly. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16. LNCS, vol. 6518, pp. 71–82. Springer, Heidelberg (2011)
3. Ma, X., Lombardi, F.: Synthesis of tile sets for DNA self-assembly. IEEE T. Comput. Aid. D. 27(5), 963–967 (2008)
4. Ma, X., Lombardi, F.: On the computational complexity of tile set synthesis for DNA self-assembly. IEEE T. Circuits-II 56(1), 31–35 (2009)
5. Rothmund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In: Proc. of STOC 2000, pp. 459–468 (2000)
6. Winfree, E.: Algorithmic Self-Assembly of DNA. Ph.D. thesis, California Institute of Technology (June 1998)
7. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature 394, 539–544 (1998)

# Towards Computation with Microchemomechanical Systems

Andreas Voigt, Rinaldo Greiner, Merle Allerdißen, and Andreas Richter

TU Dresden, Center for Advancing Electronics Dresden and Chair of Polymeric  
Microsystems,  
01062 Dresden, Germany  
andreas.richter7@tu-dresden.de  
www.polymems.de

**Abstract.** Labs-on-chips are promising candidates for the realization of chemical information systems, where data are embodied in the form of chemical concentrations. In this paper we present the concept of microchemomechanical systems, a lab-on-a-chip technology based on intrinsically active components. The active components are valves fabricated from phase-changeable polymers that provide a direct feedback mechanism and exhibit a transistor-like functionality. Therefore this microfluidic platform facilitates the realization of logic operations, if-then structures and the sampling of chemical signals. In analogy with electronic von Neumann CPUs, control and execution unit are integrated on a single chip. Due to the intrinsic activity of the valves and their small size, microchemomechanical systems are highly suitable for large-scale integration.

**Keywords:** phase-changeable polymers, hydrogels, chemical information processing, microchemomechanical systems.

## 1 Introduction

Chemical information processing is based on the use of chemical concentrations as carriers of information. The discovery of the Belousov-Zhabotinsky reaction at the end of the 50s was a crucial step for this emerging field. Another important cornerstone in the 80s was the proof, that light-sensitive Belousov-Zhabotinsky chemicals can be used for chemical image processing [1]. In the 90s the construction of cellular automata based on reaction-diffusion processes was demonstrated theoretically [2]. At the beginning of the current century it has been verified experimentally that reaction-diffusion systems can be used to set up logic gates [3]. It is worth noting, that the idea of chemical computation has developed to the point, where its core concepts are utilized for software-based artificial chemical computing schemes [4].

Fundamental logic computation schemes, including logic gates, flip-flops, counters, oscillators and modulators, have also been realized by Prakash and Gershenfeld [5] with bubble microfluidics. The development of continuous microfluidics

has been less advanced from a computational perspective. Currently large-scale integrated microfluidics is dominated by microelectromechanical systems. The flow is steered by use of valves that are operated by pneumatic control ([6], [7], [8], [9]). Several thousand valves have been integrated onto a single chip. However, since the valves have to be controlled by an external control unit, the systems are expensive, and the further scalability may be limited. More importantly, there is no direct feedback mechanism between the fluids and the controls. Looking at the success story of integrated electronic ICs, one can clearly see that the inherent feedback mechanisms of electronic transistors was one of the key factors.

We introduce the concept of microchemomechanical microfluidic systems ( $\mu$ CHEMS) [10]. The valves used for flow control are based on phase-changeable polymers. Their main advantage is the fact that they are active components directly controlled by the fluid. As in the case of electronic von Neumann CPUs both control unit and execution unit are integrated on the chip. The scalability is not limited by the need of an external control. Furthermore the smart valves have an inherent decision making ability. Therefore they are predestined for the use in computing operations such as sampling, Boolean functions and if-then structures. We see two possible applications for  $\mu$ CHEMS. Firstly they represent a “beyond CMOS technology” and may play an elemental role in the construction of an unconventional computer. Secondly, the introduction of methods from computer science, information theory and signal processing into microfluidics can provide a means for the construction of easier-to-operate, faster and multi-functional systems for the analysis and synthesis of chemicals.

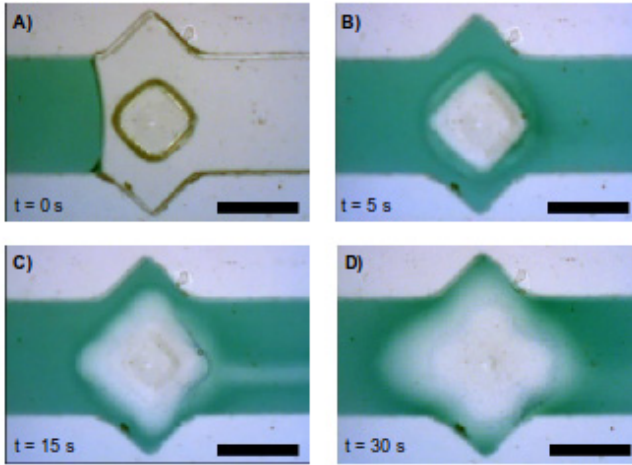
In section 2 we discuss the experimental foundation of  $\mu$ CHEMS, the smart valves and their materials. In section 3 the theoretical foundation is laid out. The terminology and methodology of chemomechanical information processing and a comparison with von Neumann CPUs is shown. In section 4 we demonstrate a microchemomechanical module for chemical sampling. In section 5 the conclusion is given, and section 6 presents an outlook how to further broaden the applicability of  $\mu$ CHEMS.

## 2 Experimental Foundations

### 2.1 Material Basis

The basis of the chemomechanical valves are *phase-changeable polymers*, i. e. polymers that can change their properties, e. g. volume or mechanical characteristics, at least once (without necessarily undergoing a phase transition). This change in volume is used to open or close fluidic channels. Three types of polymers are used in our research: a) conventional hydrogels, b) smart hydrogels, c) soluble polymers.

*Hydrogels* are cross-linked polymers that significantly change their volume by absorbing or releasing water. Conventional hydrogels undergo only one swelling process, while stimuli-sensitive (smart) hydrogels undergo reversible and reproducible volume phase transitions depending on small changes in environmental



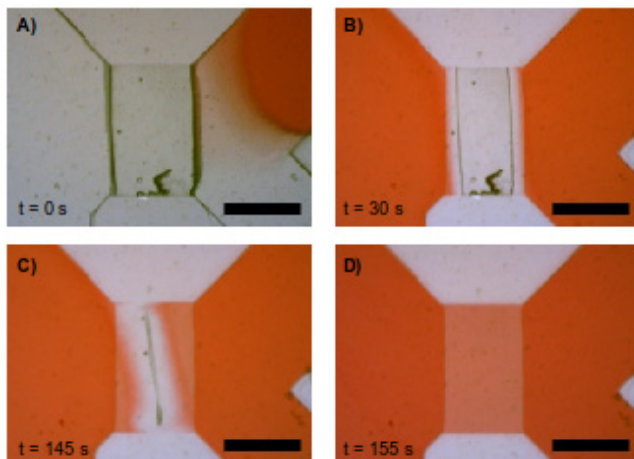
**Fig. 1.** Closing valve made of sodium acrylate, a conventional hydrogel. The closing procedure is initiated by the contact of the valve with water. (Scale bar  $500\ \mu\text{m}$ ; water colored by food color.)

parameters such as temperature, pH value or the concentration of a solute. We employ sodium acrylate (SA), a conventional polyelectrolytic hydrogel, which is characterized by its fast swelling times [11]. As a smart hydrogel we employ poly(*N*-isopropylacrylamide) (PNIPAAm) with swelling times typically two orders of magnitude lower.

*Soluble polymers* are not cross-linked and hence do not absorb water like hydrogels but – due to the breakup of intermolecular polymer-polymer interactions – they undergo a permanent dissolution process. The dissolution time depends on the material used, the size and on the flow rate of the process medium, since a saturation zone develops around the polymer in the process which has to be removed for efficient dissolution. Polyethylene glycol (PEG) and Poly(vinyl alcohol) (PVA) are used for the opening valves in this work.

## 2.2 Valves

The unique feature of the valves used for flow control in  $\mu\text{CHEMS}$  is the fact that they are controlled directly by the fluid and its constituents. This corresponds to a decision making processes comparable to the control of electric currents in transistors. Three different types of valves can be built from the materials mentioned above: a) water controlled closing valves, b) concentration/temperature controlled valves, c) water controlled opening valves. It is important to note that the switching times of the valves can be scaled over several orders of magnitude [10]. This is crucial for the construction of microfluidic chips designed to perform well-defined fluidic operations.



**Fig. 2.** Opening valve made of polyethylene glycol (PEG), a dissolvable polymer. The dissolution process is driven by the water streaming past the valve. (Scale bar  $500\ \mu\text{m}$ ; water colored by food color.)

Water controlled closing valves (Fig. 1) are based on SA. They start swelling the moment they get into contact with water and hence close the microchannel after a defined time. Afterwards, the channel remains closed permanently. The closing time can be defined by two parameters: i) The concentration of the cross-linking agent  $N,N'$ -methylenebisacrylamide (BIS); this influences the stiffness and hence the swelling time constant and the maximum swelling degree of the gel ii) The ratio of the volume of the unexpanded gel to the volume of the valve chamber. By appropriate choice of these parameters the closing time of the valve can be scaled from milliseconds to minutes. The valves can be operated with maximal back pressures between 60 and 75 kPa.

In contrast to the SA valves, concentration/temperature controlled valves made of PNIPAAm can be closed and reopened over a lifetime of many cycles. They change their state depending on the concentration of alcohol solvated in water and on their temperature. Hence they are very versatile components that have e. g. been used to build chemical transistors ([12], [13]) and chemostats [14]. Their switch time is typically in the range between minutes and hours.

Water controlled opening valves, based on soluble polymers, start to dissolve the moment they are brought into contact with water. When the polymer layer has dissolved to a certain thickness the valve is opened by a breakthrough due to the pressure difference on the opposite sides of the valve, which is an irreversible process. The opening time can be adjusted by choice of the material, the geometry and the water flow rate. We use PEG as a material for our standard opening valves (Fig. 2). Their opening times can be scaled between 20 s and 600 s. PVA is used for thin large-area membranes separating chambers. The membrane dissolution time is scalable between a few seconds and hours.



### 3 Theoretical Foundations

Here we present first considerations that will be useful for a future modeling of  $\mu$ CHEMS systems. A complete model will also comprise a description of the channel structure and the precise behavior of the valves. Promising approaches for future simulations include the use of existing tools for electric circuits and the set-up of a distinct model based on a 2d graph.

#### 3.1 Terminology

A microchemomechanical processor (Fig. 3) has a number of inlets into which a solvent (usually water) containing one or several solutes is pumped. The chip itself consists of the channel structures, valves and – possibly – sensing devices like fluorescence detectors and electrodes. It is useful to define a clear terminology for a future description in terms of methods from mathematics, electronics or computer science. The four most important characteristic terms are chemical data, system state, input and output.

*Chemical data* are represented as concentrations  $c_i(\mathbf{r}, t)$  of different chemicals (index  $i$ ) on the chip. Chemical data can be manipulated in two ways: a) by mechanical translation due to the pumping process for a given valve configuration, b) by chemical reactions. Additionally there are diffusion processes which can usually be neglected for transport, but can also be utilized for the mixing of chemicals in reaction chambers or along long channels. It is useful to distinguish pure data chemicals from control chemicals, where control chemicals play an active role in the switching of smart hydrogel valves.

The *input* is determined by the  $m$  inlets of a chip. If  $n$  chemicals are involved, the complete input signals are given by an  $m \times n$  matrix  $s_{11}(t), \dots, s_{mn}(t)$  where the first index denotes the inlet and the second index denotes the chemical. Usually most of the entries of the matrix will be zero. As with chemical data, it is useful to distinguish between input control signals and input data signals.

There are several quantities that can serve as an *output* of microfluidic chips: a) the fluid output at the  $p$  channel outlets  $t_{11}(t), \dots, t_{pn}$ , b) the signals  $f_1(t), \dots, f_q(t)$  from  $q$  fluorescence detectors sensing chemical reactions, c) the signals  $e_1(t), \dots, e_r(t)$  from  $r$  chemical sensitive electrodes.

The *system state* at time  $t$  is characterized by

1. The *valve state*: If there are  $u$  valves on the chip, this is a  $u$ -dimensional vector  $\mathbf{v}(t)$ , where  $v_i = 1$  denotes that valve  $i$  is open, and  $v_i = 0$  denotes that valve  $i$  is closed. Depending on the chip configuration it is more adequate to treat these vector components as Boolean or as continuous values.
2. The *data state*, given by the totality of all chemical concentrations  $c_1(\mathbf{r}, t), c_2(\mathbf{r}, t), \dots, c_n(\mathbf{r}, t)$  for all positions  $\mathbf{r}$ .

Note that we have refrained from defining the term *program*. There are various quantities or phenomena that bear resemblance to the normal usage of the word “program”: a) the input control signal, b) the succession of valve states, c) the succession of operations (pushing of fluids, chemical reactions).

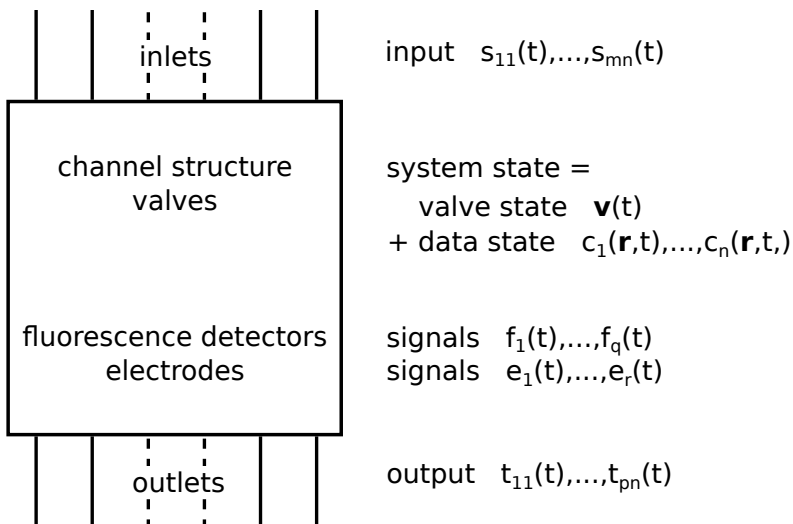


Fig. 3. Schematic overview of a microchemomechanical processor

### 3.2 Processing of Information

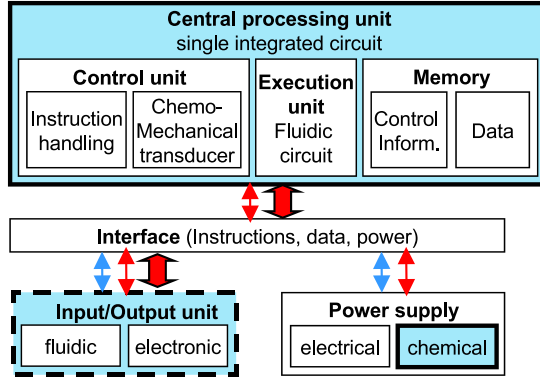
Mechanical translation of chemical concentrations is achieved by pumping the fluids through the channels or into reaction chambers. Aside from the pump parameters the effect of pumping is defined by the channel geometry and the current valve state. To a good approximation the microfluidic chip is a linear resistive network where the pumps may be either pressure sources (e. g. gravitational pumping) or current sources (e. g. peristaltic pumps). However, in contrast to electronics, where signals are transmitted by voltage or current variations, the signals in microfluidic processors are not pressure waves or flow rate waves in water. Instead the chemical concentration distributions are shifted with the flow by convective transport.

Chemical reactions can take place, if at a position  $\mathbf{r}$  the concentrations  $c_i$  and  $c_j$  of two different chemicals  $A_i$  and  $A_j$  are unequal to zero. In principle the reaction kinetics can be either stochastic [15] or deterministic [16]. For a small number of molecules stochastic models have to be used, while for large numbers of molecules the process can be considered deterministic [4]. In the simplest deterministic case the following behavior occurs: In case a reaction



can happen, and if the reactions takes place in a single step without intermediates, the rate of reaction is given by

$$r = k c_i^a c_j^b, \quad (2)$$



**Fig. 4.** Analogy of microchemomechanical systems with electronic von Neumann CPUs

where  $k$  is the rate coefficient of the reaction. The temporal development of chemical concentrations due to chemical reactions is governed by rate equations of this or a more complex type.

Note that both mechanical transport and chemical reactions facilitate several kinds of parallelizability, i. e. calculations occurring simultaneously. There is a lateral parallelizability of processes that happen in different stream; there is a longitudinal parallelizability of processes along the same stream; multiple different chemical reactions may happen simultaneously.

For a mathematical description of the switching process of hydrogel valves it is necessary to analyze the swelling (or deswelling) process. It is determined by the diffusion of polymer chains into the solvent, the polymer-solvent interaction energy and the rubber elasticity of the polymer network. An approximative description of the swelling process that is suitable in many cases has been given by Tanaka [17], who proposes that it follows an exponential decay with a characteristic time constant

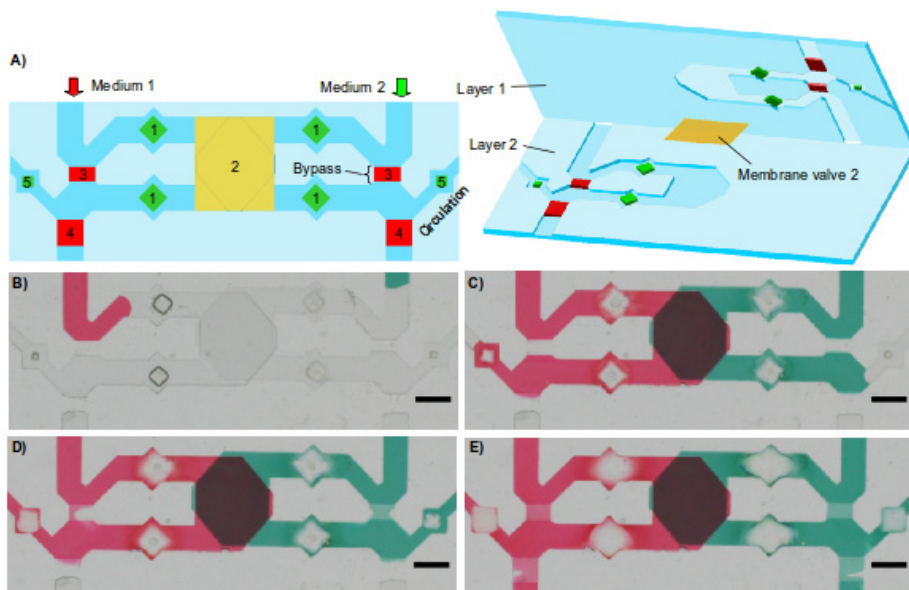
$$\tau = l^2/D. \quad (3)$$

Here  $l$  is the characteristic length scale of the hydrogel and  $D$  is the cooperative diffusion coefficient that is assumed to be a constant specific for a certain hydrogel. This law applies both to conventional and smart hydrogels. The equilibrium state of a smart hydrogel for a given set of external parameters results from a balance of the mentioned chemical processes, where the fundamental equations have been developed by Flory and Rehner ([18], [19]). In general the switch position of a smart hydrogel valve will be a sigmoid type function of the input parameters:

$$s = f_{sigm}(c_1, \dots, c_n, T, pH, \dots). \quad (4)$$

### 3.3 Comparison with an Electronic Von Neumann CPU

One defining characteristic of electronic von Neumann CPUs is the fact, that both control unit and execution unit are integrated on a single chip. In analogy a control unit and an execution unit – both on-chip – can be discerned for



**Fig. 5.** The sampling module samples two different chemical signals and unites them in a reaction chamber. (Scale bar 1 mm; water colored by food color. )

**Table 1.** Characteristics of the valves used in the sampling module

valve	material	type	time constant
1	SA	closing	45 s
2	PVA	opening	7 min
3	PEG	breakthrough	-
4	PEG	opening	3 min
5	SA	closing	3 min

$\mu$ CHEMS chips (Fig. 4). The *control unit* is responsible for generating a succession of valve states  $\mathbf{v}(t)$ . In MEMS chips this is achieved by a direct, external control of the valves, i.e. the control unit is NOT integrated on the chip. In  $\mu$ CHEMS chips the valves are controlled by the control substance. The succession of valve state  $\mathbf{v}(t)$  results from the control signal(s) for a given channel geometry and valve arrangement. There are various options for the architecture of the control unit:

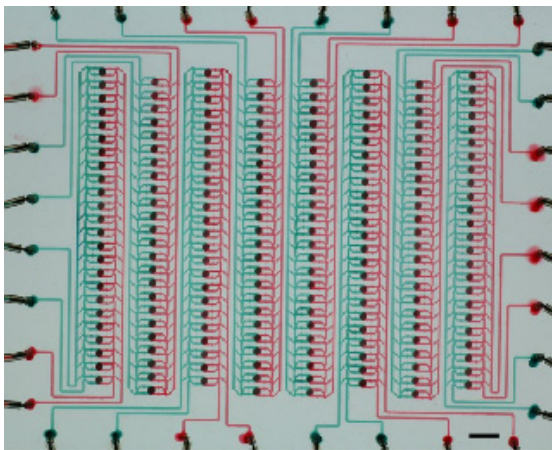
1. It can be in a layer separate from the data [20].
2. The control unit can be in the same layer as the data where the valves are controlled by the presence of water.
3. There can be an arrangement where the control chemicals are the data chemicals of interest.
4. The control unit can be in the same layer as the data, where control chemicals and data are different chemicals in the solution.

The *execution unit* manipulates the data states  $c_i(\mathbf{r}, t)$  (including the control data). An elementary operation consists in the pumping process for one fixed valve state.

## 4 Sampling of Chemical Substances

Equidistant sampling plays a crucial role in chemical analysis, e. g. in monitoring of bioreactors, enzyme analysis, analysis of factors of growth and quality control. We present a module for automated parallel sampling of two chemical input signals (Fig. 5). The module is designed for cascading, leading to a repeated sample process.

Medium 1 and medium 2 flow in different layers of the chip. Table 1 shows the types of valves used in the module and their parameters. Note that the valves 2 are membranes separating the two layers. The operation group consists of valves 1-3. When the medium enters the module (5B), bypass 3 is in the closed state. Therefore the medium flows into the chamber, passing valves 1 (5C). After 45 s valves 1 close. This is the core step of sampling, where a defined volume of the fluid is kept in the chamber. Meanwhile valve 2 starts dissolving, eventually leading to the union of both media in the chamber, which potentially starts a chemical reaction. The long dissolution time (7 min) of valve 2 was chosen in order to compensate for arrival time discrepancies between the different media. Due to the now closed valves 1 there is a pressure build-up that leads to the breakthrough of valve 3 (5D). Valves 4 and 5 are the timing group that specifies the interval until which the fluid will reach the next cascade. At the beginning the fluid flows through the bypass via valve 5. After a defined time (here: 3 min) valve 5 will close and valve 4 will open, guiding the fluid to the next cascade (5E). The parameters of the timing group can be scaled independently of the



**Fig. 6.** Photo of the microfluidic chip designed for equidistant long-term investigations. (Water colored by food color.)

operation group. Due to the fact that the valve time constants can be varied over a large range, it is possible to choose a timing of e. g. 2 h which facilitates the construction of chips for long-term investigations over several days. A chip (Fig. 6) designed with 192 modules, corresponding to 2096 valves, is described in [10]. Note the analogy of the sampling of a chemical signal described here with the sampling of a signal in electronic signal processing.

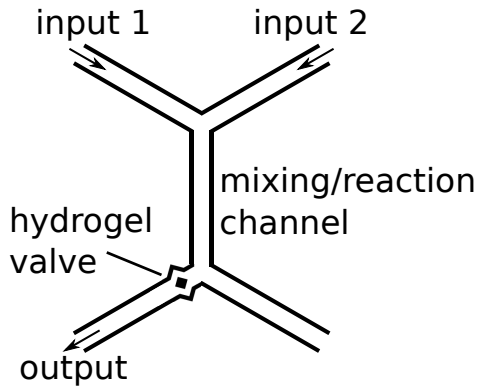
## 5 Conclusions

Chemical information processing is a paradigm inherent in nature characterized by its resilience, power and reliability [4]. Technological chemical information processing as a “beyond CMOS” technology has yet a lot to learn from nature. Labs-on-chips are promising candidates for the implementation of chemical computing, since they facilitate the manipulation of very small amounts of substances. The microchemomechanical systems ( $\mu$ CHEMS) presented here are labs-on-chips that utilize active valves with an inherent decision making ability, which makes it possible to implement elementary computing operations with simple modules and to integrate control unit and execution unit on a single chip.  $\mu$ CHEMS are excellently suitable for large scale integration. The number of valves that can be integrated on a single chip is almost identical to the number of transistors (2300) of the Intel 4004, the first commercial electronic microprocessor ([21], [22]). In addition, it seems likely that multiple types of parallelizability can be utilized with  $\mu$ CHEMS. In our vision microchemomechanical systems will be helpful both in solving computational problems from natural and life sciences, and in the automation of complex or repetitive chemical analysis and synthesis processes.

## 6 Outlook

As an outlook we want to point out to certain computational operations by use of smart hydrogels that will be worked on in the near future. First of all it is worthwhile to highlight the analogy of equation 4 to the transfer function of an artificial neuron. Following the common construction of logic gates from artificial neurons, we can build logic gates using smart hydrogel valves (Fig. 7). Two flows containing concentrations  $c_a$  and  $c_b$  of the same chemical are added by a junction. In the downstream channel sections the concentrations are mixed by diffusion, resulting in the averaged concentration  $(c_a + c_b)/2$ . Let the switching threshold concentration of the valve be  $c_0$ . We then define a concentration of e. g.  $c_a$  (or  $c_b$ ) =  $1.5c_0$  as “1” while the state “0” is represented by a concentration of 0. Hence the valve will be open exactly if both input states are “1”, while it will close otherwise. Exactly the same structure can be used as an OR gate, if we define state “1” as (e. g.)  $2.5c_0$ , while state “0” corresponds to a concentration of 0. The gate will be open if either of the input states is “1”.

Another operation that will be interesting (however more difficult to implement) in the future is the if-then type structure. Since smart hydrogels are



**Fig. 7.** Basic module for realizing logic gates and if-then structures

responsive to certain chemicals, the flows on the microfluidic chip can be guided depending on the outcome of a chemical reaction. To achieve this the same basic set-up as for the logic gates can be used. However, this time  $c_a$  and  $c_b$  are concentrations of two different chemicals. Let us assume that the smart hydrogel is not sensitive towards the two chemicals. This time the downstream channel section is not mainly used for diffusive mixing, but as a chemical reactor. Depending on whether the hydrogel is sensitive towards the result of the reaction it will be open or closed. A successful implementation of if-then structures will broaden the applicability of microchemomechanical systems immensely, both for a potential use in pure chemical information processing and automated chemical analysis and synthesis.

**Acknowledgments.** This work is supported by the German Research Foundation (Heisenberg Chair of A.R., RI 1294/2-1, 3-1, 6-1, 7-1), the Saxon State Ministry for Science and the Arts as well as the European Social Fund. The work is also partly supported by the German Research Foundation (DFG) within the Cluster of Excellence “Center for Advancing Electronics Dresden”.

## References

1. Kuhnert, L., Agladze, K.I., Krinsky, V.I.: Image processing using light-sensitive chemical waves. *Nature* 337, 244–247 (1989)
2. Adamatzky, A.: Information-processing capabilities of chemical reaction-diffusion systems. 1. belousov-zhabotinsky media in hydrogel matrices and on solid supports. *Advanced Materials for Optics and Electronics* 7, 263–272 (1997)
3. Adamatzky, A., De Lacy-Costello, B.: Experimental logical gates in a reaction-diffusion medium: The xor gate and beyond. *Physical Review E* 66, 046112 (2002)
4. Dittrich, P.: Chemical computing. In: Banâtre, J.-P., Fradet, P., Giavitto, J.-L., Michel, O. (eds.) UPP 2004. LNCS, vol. 3566, pp. 19–32. Springer, Heidelberg (2005)

5. Prakash, M., Gershenfeld, N.: Microfluidic bubble logic. *Science* 315, 832–835 (2007)
6. Mark, D., Haeberle, S., Roth, G., von Stetten, F., Zengerle, R.: Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications. *Chem. Soc. Rev.* 39, 1153–1182 (2010)
7. Arora, A., Simone, G., Salieb-Beugelaar, G.B., Kim, J.T., Manz, A.: Latest developments in micro total analysis systems. *Anal. Chem.* 82, 4830–4847 (2010)
8. Chin, D., Linder, V., Sia, S.K.: Commercialization of microfluidic point-of-care diagnostic devices. *Lab Chip* 12, 2118–2134 (2012)
9. Gervais, L., de Rooij, N., Delamarche, E.: Microfluidic chips for point-of-care immunodiagnosics. *Adv. Mater.* 23, H151–H176 (2011)
10. Greiner, R., Allerdissen, M., Voigt, A., Richter, A.: Fluidic microchemomechanical integrated circuits processing chemical information. *Lab Chip* 12, 5034–5044 (2012)
11. Richter, A., Paschew, G., Klatt, S., Lienig, J., Arndt, K.-F., Adler, H.-J.: Review on hydrogel-based pH sensors and microsensors. *Sensors* 8, 561–581 (2008)
12. Gerlach, G., Arndt, K.-F. (eds.): *Hydrogel Sensors and Actuators: Engineering and Technology*. Springer (2009)
13. Richter, A., Tuerke, A., Pich, A.: Controlled double-sensitivity of microgels applied to electronically adjustable chemostats. *Adv. Mater.* 19, 1109–1112 (2007)
14. Richter, A., Wenzel, J., Kretschmer, K.: Mechanically adjustable chemostats based on stimuli-responsive polymers. *Sens. Actuat. B* 125, 569–573 (2007)
15. Gillespie, D.T.: Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry* 58(1), 35–55 (2007), PMID: 17037977
16. Houston, P.L.: *Chemical Kinetics and Reaction Dynamics*. Dover Publications (2006)
17. Tanaka, T., Fillmore, D.J.: Kinetics of swelling of gels. *J. Chem. Phys.* 70, 1214–1218 (1979)
18. Flory, P.J., Rehner, J.: Statistical mechanics of crosslinked polymer networks i. rubberlike elasticity. *J. Chem. Phys.* 11, 512–520 (1943)
19. Flory, P.J., Rehner, J.: Statistical mechanics of crosslinked polymer networks ii. swelling. *J. Chem. Phys.* 11, 521–526 (1943)
20. Allerdissen, M., Greiner, R., Richter, A.: Microfluidic microchemomechanical systems. *Adv. Sci. Technol.* 81, 84–89 (2013)
21. Augarten, S.: *State of the art: a photographic history of the integrated circuit*. Ticknor & Fields (1983)
22. Faggin, F., Hoff, M.E., Mazor, S., Shima, M.: The history of the 4004. *IEEE Micro*, 10–20 (December 1996)



# Evolutionary Programming Using Distribution-Based and Differential Mutation Operators

Md. Tanvir Alam Anik and Saif Ahmed

Department of Computer Science and Engineering,  
Bangladesh University of Engineering and Technology, Dhaka, Bangladesh  
{tanviranik, iconicsaif}@gmail.com

**Abstract.** In this paper, we propose an evolutionary programming (EP) algorithm that incorporates both distribution-based and differential mutation operators in one algorithm. Distribution-based mutation operators are the ones that employ probability distribution functions such as Gaussian, Cauchy distributions for mutation. Thus the balance between exploration and exploitation is obtained by two different categories of mutation operators.

**Keywords:** Distribution-based mutation operators and differential mutation operators.

## 1 Introduction

Evolutionary programming (EP), evolution strategies (ESs), and genetic algorithms (GAs) are three main streams of Evolutionary algorithms (EAs) and have been extensively used in global optimization problems. Since mutation is the main operator in EP, a number of innovative mutation operators based on various probability distributions e.g. Gaussian mutation (CEP) [2], Cauchy mutation (FEP) [1], a combination of Cauchy and Gaussian mutations [1],[3] have been proposed to improve the performance of EP. Differential Evolution (DE), another simple yet effective algorithm for global optimization uses both crossover and mutation to produce offspring. DE executes its mutation by adding a weighted difference vector between two individuals to a third individual. This paper describes a new EP algorithm based on dual mutation scheme (DMEP) that integrates both distribution-based and differential mutation operators to improve explorative and exploitative search abilities of EP.

## 2 DMEP Algorithm

DMEP attempts to achieve the balance between exploration and exploitation by two different categories of mutation operators. A simple island model has been adopted to partition the main population into two sub-populations. We call these

sub-populations: Island-1(IS1) and Island-2(IS2). Individuals of IS1 and IS2 are mutated by distribution-based and differential mutation operators respectively. Each type contains mutation operators capable of producing different search step sizes.

## 2.1 Distribution-Based Mutation Operators

In order to globally explore wider regions of a search space heavy tail distribution like Cauchy distribution [1] is suitable. Meanwhile, short tail distribution like Gaussian distribution [2] can promote local exploitation. The arithmetic mean of Cauchy and Gaussian distributions [3] can be effective when a search step size in between Cauchy and Gaussian distributions is necessary. In this regard, Cauchy, Gaussian and arithmetic mean of Cauchy and Gaussian distributions (MMO) [3] have been chosen as the distribution-based mutation operators for DMEP to provide different search step sizes while producing offspring.

## 2.2 Differential-Based Mutation Operators

Differential mutation operators have been included in the proposed scheme to achieve a proper tradeoff between exploration and exploitation. A mutation operation that incorporates globally best individual of the entire population can increase exploitative search ability. Meanwhile, a mutation operation that incorporates local neighbors of an individual based on its position in the current population not considering geographical nearness or similar fitness values can increase explorative search ability. Thus DMEP includes DE/target-to-global-best/1 and DE/target-to-local-best/1: two differential mutation operators details of which can be found in [4].

## 3 Conclusion

In summary, we have proposed an EP algorithm that obtains global exploration and local exploitation abilities with the help of two different categories of mutation operators. Details about the behavior of these mutation operators and experimental evaluation of DMEP over some benchmark functions are the focuses of our future work.

## References

1. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation* 3(2), 82–102 (1999)
2. Back, T., Schwefel, H.-P.: An overview of evolutionary algorithms for parameter optimization. *Evolutionary Comput.* 1(1), 1–23 (1993)
3. Chellapilla, K.: Combining mutation operators in evolutionary programming. *IEEE Transactions on Evolutionary Computation* 2(3), 91–96 (1998)
4. Das, S., Abraham, A., Chakraborty, U.K., Konar, A.: Differential evolution using a neighborhood-based mutation operator. *IEEE Trans. Evol. Comput.* 13(3), 526–553 (2009)

# A P System Parsing Word Derivatives\*

Artiom Alhazov, Elena Boian, Svetlana Cojocaru, Alexandru Colesnicov,  
Ludmila Malahov, Mircea Petic, and Yurii Rogozhin

Institute of Mathematics and Computer Science, Academy of Sciences of Moldova  
Academiei 5, Chişinău, MD-2028, Moldova  
{artiom, lena, Svetlana.Cojocaru, kae, mal, mirsha, rogozhin}@math.md

**Abstract.** This paper describes membrane computational models parsing affixed Romanian words with prefixes, suffixes, terminations, and alterations in the root. An algorithm for Romanian affixes extraction is given, and several models of P systems are proposed.

**Keywords:** affixation, morphemes, parsing, P system models, membrane computing, linguistic resources.

This paper discusses construction of membrane systems to parse Romanian words with affixes. This contributes to replenishment of corpora and dictionaries, and to formation of morphological word nest for derivation. Affixation is the most productive technique to form new Romanian words as the Romanian language possesses 86 prefixes and approx. 600 suffixes [2]. We proposed in [1] several models of P systems to select affixed Romanian words. We here continue that work, allowing a derivation step to have more than one root alternation, addition of a prefix and a suffix, replacement of a termination, as well as all of the above.

**Word Derivation Model.** Assume that we have a finite set of word pairs  $A$  of root alternations and finite languages  $\text{Pref}$  of prefixes,  $\text{RR}$  of roots,  $\text{Suf}$  of suffixes and  $T$  of terminations ( $T$  may include the empty word), all over a finite alphabet  $V$ . We also write elements of  $A$  in the form  $x \rightarrow y$ . We use  $\overline{\text{Pref}}$ ,  $\widehat{\text{Suf}}$  to denote the sets  $\text{Pref}$ ,  $\text{Suf}$ , where all symbols of each word have lines or hats over them. These two cases correspond to operations of adding a prefix and adding a suffix. We denote the marked terminations by  $\boxed{T} = \{\boxed{t} \mid t \in T\}$ , and the termination rewriting rules by  $\overline{T} = \{\overline{t_1} \rightarrow \overline{t_2} \mid t_1, t_2 \in T\}$ . Let  $Op = \overline{\text{Pref}} \cup \widehat{\text{Suf}} \cup \overline{T} \cup A$ . The fourth case ( $A$ ) corresponds to an operation of performing an alternation.

Assume we have a finite language  $M$  over  $Op$  of control words  $s = o_{i_1} \cdots o_{i_k}$  corresponding to the derivation steps consisting of operations described above. We now define them more formally, using the syntax  $o(w)$  to denote the result of operation  $o$  over a word  $w$  (note that the result of some operations may be undefined on some words, the corresponding choice not leading to any result):

$$\begin{aligned} \overline{p}(w) &= \overline{p}w, \widehat{s}(w\boxed{t}) = w\widehat{s}\boxed{t}, (\overline{x} \rightarrow \overline{y})(w\overline{x}) = w\overline{y}, \\ (x \rightarrow y)(w_1xw_2) &= w_1yw_2, (o_{i_1} \cdots o_{i_m})(w) = o_{i_1}(\cdots(o_{i_m}(w))\cdots). \end{aligned}$$

\* The authors acknowledge the project STCU-5384 “Models of high performance computations based on biological and quantum approaches”.

We speak about the problem of **accepting** a language obtained by removing the prefix, suffix and termination marks from the words of the minimal language  $L$ , such that if  $wt \in Q$ , then  $w\boxed{t} \in L$ , and if  $w \in L$  and  $s \in M$  then  $s(w) \in L$  if it is defined. Our acceptor also produces the lexical decomposition of the input.

**Definitions.** Let  $O$  be a finite set of elements called symbols, then the set of words (strings) over  $O$  is denoted by  $O^*$ , and the empty word is denoted by  $\lambda$ .

A P system with string-objects and input is a tuple consisting of the working alphabet, an input alphabet, a membrane (tree) structure of  $p$  membranes, *initial* multisets of strings over  $O$  in regions  $i$ ,  $1 \leq i \leq p$ , finite sets of rules defining the behavior of strings from  $O$  in regions  $i$ ,  $1 \leq i \leq p$ , and the input region.

A rule  $x \rightarrow (y, tar)$  for region  $i$  can be applied to a string  $uxv$  in region  $i$ , resulting in a string  $uyv$  in region specified by  $tar$ . Whenever there are multiple ways to apply different rules, we assume presence of sufficient number of copies.

**Main Result.** We describe the P system  $\Pi$  accepting words  $x$  given in form  $\$1x\$2$ . Let  $Op = \{o_1, \dots, o_k\}$  and  $T = \{t_1, \dots, t_n\}$ . We also define a set  $W = \text{Suf}(M^r)$  of suffixes of the mirror language of  $M$ .

$\Pi$  reversely applies operations of adding affixes and alternations in terminations and in the rest of the word, according to the control words from  $M$ . End markers  $\$1$  and  $\$2$  ensure that affixes are only removed from the appropriate ends of the word. Affixes are moved outside of the interval between  $\$1$  and  $\$2$ .

First  $\Pi$  marks a termination in the word, sending the string to the skin. The main evolution is reduced to selecting and performing reverse derivation steps in regions corresponding to the operations; the skin controls the substeps of the process. At any time, the system sends a copy of the word into a region corresponding to its termination, and back to the skin, unmarking the termination and moving it to the left of all suffixes, separated by a hyphen from the root, in case the control symbol was  $\langle \lambda \rangle$ . If the word between the markers (the root and the termination) matches some word in RR, the resulting word is sent out.

Besides accepting words, the system also decomposes the word. To do so, instead of removing prefixes and suffixes, they are moved outside of the interval between  $\$1$  and  $\$2$ .

The terminations  $T$  are  $\lambda$ , a, ă, e, ea, i, ică, ie, iu, î, l, o, u, ui, uie.

Examples with root alternations: words  $\$1\text{întineri}\$2$ ,  $\$1\text{fetiță}\$2$ ,  $\$1\text{mulțime}\$2$ ,  $\$1\text{deșteaptă}\$2$ ,  $\$1\text{brăduț}\$2$  and  $\$1\text{desprăfuire}\$2$  (youthen, little girl, multitude, dignified (fem.), small spruce, undusting) will yield output  $\widehat{m}\text{-tânăr--}\widehat{i}$ ,  $\widehat{f}\text{-ă--}\widehat{t}$ ,  $\widehat{m}\text{t-}\widehat{i}\widehat{m}$ ,  $\widehat{d}\text{e-}\widehat{t}$ ,  $\widehat{b}\text{rad--}\widehat{u}\widehat{t}$ , and  $\widehat{d}\text{e-}\widehat{p}\text{raf--}\widehat{u}\widehat{i} - \widehat{r}\widehat{e}$ , respectively.

This model can be used for other languages with similar word derivation.

## References

1. Alhazov, A., Boian, E., Cojocaru, S., Colesnicov, A., Malahov, L., Petic, M., Ciubotaru, C.: Membrane Models of Romanian Word Affixation. Applied Linguistics and Linguistic Technologies: MegaLing-2012. Kyiv (in print, 2013) (in Russian)
2. Graur, A., Avram, M.: Word formation in Romanian, vol. II, p. 310. Romanian Academy Press, Bucharest (1978) (in Romanian)

# Computational Power of Protein Interaction Networks

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science  
Blvd. Carol I no.8, 700506 Iași, Romania  
baman@iit.tuiasi.ro, gabriel@info.uaic.ro

**Abstract.** It is proved that an abstract model of protein-protein interaction derived from membrane computing can simulate all computable functions by using a small number of components, not so complex proteins (having at most lengths two, where length is an abstract measure of complexity), and operations inspired by endocytosis (*pino*, *phago*) and exocytosis (*exo*).

An important part of cell activity is realized by a complex protein-protein network. Most of the actions taking place in a cell are in fact controlled by proteins bound on cells membranes. These proteins can be of two types: peripheral proteins (placed on the internal or external side of a cell membrane) and integral proteins (having parts on both internal and external sides of a membrane). In order to cope with the increased complexity of protein-protein interaction networks, their complexity is correlated with protein lengths. Proteins on membranes are often arranged in large complexes in order to transduce extracellular signals into intracellular ones.

Membrane systems represent a class of computing devices inspired by living cells that are complex hierarchical membrane structures with a flow of materials and information which underlies their functioning [2]. The membranes contain multisets of symbols (also called objects), evolution rules acting on these objects, and possibly other membranes. Inspired by the peripheral proteins of cells and since membranes are highly dynamic, several types of membrane systems were previously investigated [1].

We prove that protein interaction networks using proteins of small complexity (length) acting according to various biological inspired operations can simulate all computable functions.

*Protein-Protein Interaction Networks* For an alphabet  $V = \{a_1, \dots, a_n\}$ , we denote by  $V^*$  the set of all strings over  $V$ .  $V^*$  is a monoid with  $\lambda$  as its unit element, and  $V^+ = V^* \setminus \{\lambda\}$ . By  $V^\sim = \{a_1 \sim \dots \sim a_n \mid a_1, \dots, a_n \in V, n \geq 1\}$  we denote the set of protein complexes. In order to illustrate the fact that proteins can interact, we denote by  $a$  the proteins and by  $\bar{a}$  the co-proteins that can interact with  $a$ . We use  $\overline{a_1 \sim \dots \sim a_n}$  as a shorthand notation for  $\bar{a}_1 \sim \dots \sim \bar{a}_n$ .

**Definition 1.** A protein-protein interaction networks with  $n$  membranes is a construct  $\Pi = (V, \mu, u_1, \dots, u_n, v_1, \dots, v_n, R)$ , where:

1.  $V$  is a finite (non-empty) alphabet of proteins;
2.  $\mu$  is a membrane hierarchical structure with  $n \geq 2$  membranes; the membranes are bijectively labelled by  $\{1, \dots, n\}$ ;
3.  $u_1, \dots, u_n$  are multisets of proteins bounded to the  $n$  membranes;
4.  $v_1, \dots, v_n$  are multisets of proteins placed inside the  $n$  membranes;
5.  $R$  is a finite set of rules of the following forms:
  - $[a]_b \rightarrow [ ]_{a \sim b}, a \in V, b \in V^\sim$  (bondin)
  - $[ ]_{ba} \rightarrow [ ]_{b \sim a}, a \in V, b \in V^\sim$  (bondout)
  - $u[v]_a \bar{a} \rightarrow [[u']_c v']_d, a, \bar{a} \in V^\sim, u, v, u', v' \in V^*, c, d \in (V^\sim)^*$  (pino)
  - $[[u]_a v]_{\bar{a}} \rightarrow u'[v']_{cd}, a, \bar{a} \in V^\sim, u, v, u', v' \in V^*, c, d \in (V^\sim)^*$  (exo)
  - $[u]_a [v]_{\bar{a}} \rightarrow [[[u']_c]_d v']_b, a, \bar{a} \in V^\sim, u, v, u', v' \in V^*, c, d, b \in (V^\sim)^*$  (phago)

Starting from an initial configuration of the network (given by the initial membrane structure and multisets of proteins), the evolution takes place by applying the rules activated by protein-protein interactions. A rule is applicable when all the involved proteins and membranes appearing in its left-hand side are available. In each step a membrane can be used in at most one mobility rule. A halting configuration is reached when no rule is applicable. The result of a halting evolution consists of all the vectors describing the multiplicity of proteins inside and on all the membranes (a non-halting evolution provides no output).

*Computational Power of Protein Interactions.* The rules (pino) and (phago) are used to increase the number of membranes, while rule (exo) is used to decrease the number of membranes. We combine the rules (pino) and (phago) with (exo) just to balance the number of membranes.

In the following results, three membranes represent the minimum number with respect to the operations of endocytosis and exocytosis.

**Theorem 1.** *Protein-protein interaction networks with three membranes and proteins of length two can simulate any computable function by using rules of types (bondin), (bondout), (pino) and (exo).*

Comparing to Theorem 1, the higher number of membranes for the next result is triggered by the use of (phago) operation and lack of (bondout).

**Theorem 2.** *Protein-protein interaction networks with four membranes and proteins of length two can simulate any computable function by using rules of types (bondin), (phago) and (exo).*

Up to our knowledge this is the first quantitative approach in terms of an abstract measure of complexity (called length in this paper) that studies the computational power of protein interaction networks.

## References

1. Aman, B., Ciobanu, G.: *Mobility in Process Calculi and Natural Computing.* Springer (2011)
2. Păun, G., Rozenberg, G., Salomaa, A. (eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press (2010)

# Towards an All-Optical Soliton FFT in the 3NLS-Domain

Anastasios G. Bakaoukas

Department of Computing, Coventry University, Coventry, UK  
ab3369@coventry.ac.uk, Anastasios.Bakaoukas@gmail.com

**Abstract.** An all-optical soliton method for calculating the FFT (Fast Fourier Transform) algorithm is presented. The method comes as an extension of the calculation methods (Soliton Gates) as they become possible in the Cubic Nonlinear Schrödinger Equation (3NLSE) domain, and provides a further proof of the computational abilities of the scheme. The method involves collisions entirely between first order solitons in optical fibers whose propagation evolution is described by the 3NLSE.

## 1 Introduction

There is a number of studies in which the use of soliton optical pulses for the purposes of carrying out computations has been investigated [1, 2]. For the purposes of this study only temporal solitons (involving a balance between the Kerr type non-linearities and the dispersive effects in glass fibres) are concerned. At this early point the fact that the interactions between solitons of this type can be a relatively long-range phenomenon need to be emphasised, because the Kerr non-linearity is a relatively weak effect. Temporal solitons in optical fibres where the non-linearity is of the Kerr type, are well described by the 3NLS Equation which, for very short (fs) pulses, requires corrections to account for higher order dispersion, Raman scattering etc. If pulse widths are such that these higher order effects can be neglected, then Solitons in optical fibres, are solutions of the integrable 3NLSE and since collisions between fibre solitons are elastic they were not previously considered to be capable of useful computation [2].

## 2 All-Optical Soliton FFT

The Half-adder processor scheme, first introduced in [3], forms the essential central building block on which the overall FFT Soliton computational scheme is wrapped around, to realise in the end the complete “Butterfly” calculation process which directly leads to the all-optical soliton FFT computational arrangement. The system reads the collision envelopes at distance and time specified points and uses this information to generate solitons with an appropriate phase value to represent the output of each “gate”. The phase values of two of the output solitons determine the “sum” and “carry” outputs at the end of the computation process whilst all other solitons are superfluous to this calculation.

The scheme is flexible enough to be gradually get “packed” in fixed-purpose calculation lengths.

The “Two 2-bit Numbers Multiplier” involves a Half-adder as its lying-in-its-heart functional unit (Three-bit Adder arrangement). The particular arrangement forms the compact small-scale equivalent of the “Two maximum-number-of-bits Numbers Multiplier”, which for general purpose calculations must involve Full-adders as well as Half-adders in its arrangement. The reason behind choosing the Two 2-bit Numbers Multiplier is only the fact that the particular arrangement possesses all the functionalities and properties need to be demonstrated, while at the same time gives us the ability to keep the material presented at a minimum of extension and complexity. The Two 2-bit numbers multiplier arrangement requires the addition of another four AND gates, to accommodate initial bit multiplications. For the remaining part of the “Butterfly” calculation process, we need a soliton arrangement to convert a positive bit-number to a negative one. In order to achieve this we adopt the method of complementing each digit in a bit-number in turn (change 1 for 0 and 0 for 1) and then add 1 to the result. That way, the bit-number taken out of the procedure corresponds to a bit-number representing the negative equivalent of the initial bit-number. A series of collisions between the solitons carrying the bit-number values and a single “Control Soliton” with a phase value opposite to the one possessed by the “Control Soliton” that generated the initial bit-number, is enough to produce the bit-number complement. The appropriate “Control Soliton” to achieve the complement calculation must possess a phase value of 0, in turn corresponding to a bit value of 0. The addition of 1 to the complement can be easily achieved by means of Full-adder arrangements internally consisting of two interconnecting Half-adder arrangements and an OR gate.

After the complement of a bit-number has been calculated, subtracting it from another bit-number requires the addition between the complement calculated and the second bit-number. That way only Half-adder and Full-adder arrangements are required for the realisation of all the calculations involved in the “Butterfly” arrangement. Addition and subtraction calculations appear at the final stages of the “Butterfly”, those that actually are giving the result and passing the values calculated to the next processing stage of the overall FFT calculation arrangement. Having completed the identification of the individual parts out of which the soliton “Butterfly” arrangement consists of, we can present the schematic of the overall arrangement.

## References

1. Jakubowski, M.H., Steiglitz, K., Squier, R.K.: Computing with Solitons Multi-Valued Logic (Special Issue on Collision Based Computing) (2001)
2. Jakubowski, M.H., Steiglitz, K., Squier, R.K.: When Can Solitons Compute? Complex Systems 10(1) (1996)
3. Bakaoukas, A.G., Edwards, J.: Computing in the 3NLS Domain using First Order Solitons. International Journal of Unconventional Computing (IJUC) 5(6) (2009) ISSN: 15487199



# Quantum Random Active Element Machine

Michael Stephen Fiske

Aemea Institute, San Francisco, CA  
mf@aemea.org

In [4], a computational procedure (Procedure 2) – combining quantum randomness and the active element machine (AEM) [5] – executes a universal Turing machine with Turing incomputable firing patterns. The procedure emulates any digital computer program so its computational steps are incomprehensible to an external observer. This procedure’s purpose is to hinder malware authors.

An AEM consists of computational primitives called *active elements* that simultaneously transmit and receive pulses to and from other active elements. Each pulse has an amplitude and a width, indicating how long the pulse amplitude lasts as input to the element receiving the pulse. If element  $E_i$  simultaneously receives pulses with amplitudes summing to a value greater than  $E_i$ ’s threshold and  $E_i$ ’s refractory period has expired, then  $E_i$  fires. If  $E_i$  fires at time  $t$  and a non-zero connection exists from  $E_i$  to  $E_k$ , a pulse reaches element  $E_k$  at time  $t + \tau_{ik}$ , where  $\tau_{ik}$  is the transmission time. AEM programs are built from `element`, `connection`, `fire`, `program` and `meta` commands. A command explicitly specifies its execution time. Multiple commands can simultaneously execute. During AEM program execution, the `meta` command can self-modify the AEM.

These constructions are physically realizable; the AEM model and a quantum random number generator (QRNG) device [8] act as a single computational entity. The quantum randomness and the `meta` command can non-deterministically modify the AEM’s program. A theory of ideal QRNGs in [1] strives to certify the behavior of actual QRNG devices [2]. Given an ideal QRNG that never stops measuring 0’s and 1’s, the theory in [1] implies that the binary sequence  $x_0x_1\dots$  is bi-immune. Set  $\mathcal{A}$  corresponds to  $x_0x_1\dots$ , where  $k \in \mathcal{A}$  if and only if  $x_k = 1$ .

Set  $\mathcal{A} \subset \mathbb{N}$  is *immune* if  $\mathcal{A}$  is infinite and  $\forall \mathcal{B} \subset \mathbb{N}$ , [ $\mathcal{B}$  is infinite and computably enumerable]  $\implies \mathcal{B} \cap \overline{\mathcal{A}} \neq \emptyset$ .  $\mathcal{A}$  is *bi-immune* if both  $\mathcal{A}$  and  $\overline{\mathcal{A}}$  are immune. The following lemma helps prove theorem 1: Let  $\mathcal{A} \oplus \mathcal{B} = (\mathcal{A} - \mathcal{B}) \cup (\mathcal{B} - \mathcal{A})$ . If  $\mathcal{R}$  is computably enumerable and  $\mathcal{A}$  is bi-immune, then  $\mathcal{A} \oplus \mathcal{R}$  is bi-immune.

**Theorem 1.** *Suppose the measurement, noncontextuality, eigenstate and elements of physical reality assumptions in [1] hold. Thus, in [4], the quantum random sequence used in procedure 2 is bi-immune. Hence, in procedure 2, the active element firing pattern (definition 3, page 79) – emulating the computation of a non-halting universal Turing machine – is a bi-immune sequence.*

**Theorem 2.** *If  $\mathcal{A}$  is a bi-immune set, created by a QRNG, and  $\mathcal{R}$  is Turing computable, then a quantum random AEM can compute bi-immune  $\mathcal{A} \oplus \mathcal{R}$ .*

Our constructions are motivated by the observations that a Gödel numbering is a special type of interpretation and a Turing machine is a discrete, autonomous, dynamical system. In [7], pages 26–29 describe an implicit *interpretation as-*

*sumption* in computability theory: e.g., a fixed Gödel numbering for the partial recursive functions is a Turing computable coding from sets of instructions to the integers. This assumption puts an unnecessary constraint on computation, illustrated in [4] as an incomputable firing interpretation to an external observer.

Let states  $Q = \{q_1, \dots, q_{|Q|}\}$ , alphabet  $A = \{a_1, \dots, a_{|A|}\}$ , halt state  $h$  and program  $\eta : Q \times A \rightarrow Q \cup \{h\} \times A \times \{-1, +1\}$  be a Turing machine. Define a 1–1 mapping  $\phi$  from  $\eta$  to a finite set of affine functions. Set  $B = |A| + |Q| + 1$ . Set  $\nu(h) = 0$ ,  $\nu(a_i) = i$  and  $\nu(q_i) = i + |A|$ .  $\phi$  maps right computational step  $\eta(q, T_k) = (r, \alpha, +1)$  to affine  $f(x, y) = (Bx - B^2\nu(T_k), \frac{1}{B}y + B\nu(r) + \nu(\alpha) - \nu(q))$ . State  $q$  moves to  $r$ ;  $\alpha \in A$  replaces  $T_k$  on tape square  $k$ .  $\phi$  maps left step  $\eta(q, T_k) = (r, \alpha, -1)$  to  $g(x, y) = (\frac{1}{B}x + B\nu(T_{k-1}) + \nu(\alpha) - \nu(T_k), By + B\nu(r) - B^2\nu(q) - B\nu(T_{k-1}))$ .  $\phi$  maps configuration  $(q, k, T) \in Q \times \mathbb{Z} \times A^{\mathbb{Z}}$  to  $\phi(q, k, T) = (\sum_{j=-1}^{\infty} \nu(T_{k+j+1})B^{-j}, B\nu(q) + \sum_{j=0}^{\infty} \nu(T_{k-j-1})B^{-j})$  in the  $x$ - $y$  plane.

Dynamical system  $\frac{dx}{dt} = F(x, y)$ ,  $\frac{dy}{dt} = G(x, y)$  is *autonomous* if the independent variable  $t$  does not appear in  $F$  and  $G$ . The map  $H(x, y) = (1 + y - \frac{7}{5}x^2, \frac{3}{10}x)$  is discrete and autonomous. Executing a Turing machine corresponds to iterating a discrete, autonomous system in the  $x$ - $y$  plane, consisting of a finite number of affine functions, whose domains lie in distinct unit squares. If configuration  $(q, k, T)$  halts after  $n$  computational steps, then the orbit of  $\phi(q, k, T)$  exits one of the unit squares on the  $n$ th iteration. If configuration  $(r, j, S)$  is immortal, then the orbit of  $\phi(r, j, S)$  remains in these unit squares forever.

From these observations, a proof of the *Turing unsolvability of the halting problem* is reexamined. On pages 9–10 of [3], a proof by contradiction is used to define a “total, Turing computable”  $g(x) = \begin{cases} 1 & \text{if } \Phi_x(x) \text{ does not halt} \\ \Phi_x(x) + 1 & \text{if } \Phi_x(x) \text{ halts} \end{cases}$  where  $\Phi_x(y)$  represents a universal Turing machine. The existence of  $y$  with  $g = \Phi_y$  and the resulting contradiction  $g(y) = \Phi_y(y) + 1 = g(y) + 1$  depend upon the interpretation assumption as  $\Phi_x(y)$  acts as an interpreter in the proof. No contradiction is necessarily reached from a Turing incomputable interpretation. Since the `meta` command uses quantum randomness to modify the AEM program, this can create a non-autonomous system. Non-autonomous systems exhibit dynamical behaviors that autonomous systems cannot produce [6].

## References

1. Abbott, A.A., Calude, C.S., Conder, J., Svozil, K.: Strong Kochen-Specker theorem and incomputability of quantum randomness. *Phys. Rev. A* 86, 062109, 1–11 (2012)
2. Calude, C.S., Dinneen, M.J., Dumitrescu, M., Svozil, K.: Experimental Evidence of Quantum Randomness Incomputability. *Phys. Rev. A* 82, 022102, 1–8 (2010)
3. Downey, R., Hirschfeldt, D.: *Algorithmic Randomness and Complexity*. Springer (2010)
4. Fiske, M.S.: Turing Incomputable Computation. In: *Turing-100 Proceedings*. Alan Turing Centenary. EasyChair, vol. 10, pp. 66–91 (2012), <http://www.aemea.org/Turing100>
5. Fiske, M.S.: The Active Element Machine. In: Unger, H., Kyamakya, K., Kacprzyk, J. (eds.) *Autonomous Systems: Developments and Trends*. SCI, vol. 391, pp. 69–96. Springer, Heidelberg (2011)

6. Fiske, M.S.: Non-autonomous Dynamical Systems Applicable to Neural Computation. Northwestern University (1996)
7. Rogers Jr., H.: Theory of Recursive Functions and Effective Computability. MIT Press (1987)
8. Stefanov, A., Gisin, N., Guinnard, O., Guinnard, L., Zbinden, H.: Optical quantum random number generator. *Journal of Modern Optics* 47, 595–598 (2000)

# Simulating Metabolic Processes Using an Architecture Based on Networks of Bio-inspired Processors

Sandra Gómez Canaval, José Ramón Sánchez, and Fernando Arroyo

Department of Languages, Projects and Computer Systems,  
University College of Computer Science,  
Technical University of Madrid, Crta. de Valencia km. 7 - 28031 Madrid, Spain  
{sgomez,jcouso,farroyo}@eui.upm.es

In this work, we propose the Networks of Evolutionary Processors (NEP) [2] as a computational model to solve problems related with biological phenomena. In our first approximation, we simulate biological processes related with cellular signaling and their implications in the metabolism, by using an architecture based on NEP (NEP architecture) and their specializations: Networks of Polarized Evolutionary Processors (NPEP) [1] and NEP Transducers (NEPT) [3]. In particular, we use this architecture to simulate the interplay between cellular processes related with the metabolism as the Krebs cycle and the malate-aspartate shuttle pathway (MAS) both being altered by signaling by calcium.

NEP is complete and efficient from the computational point of view (i.e. is able to solve hard problems NP complete given linear time solutions). This model consists of several processors, each of one is placed in a node of a virtual graph. Each processor acts on local data in accordance with some predefined rules (evolutionary operations simulating point mutations of nucleotides over DNA sequences) and communicates the results using a filtering strategy. This strategy may require satisfy some conditions that are imposed by processors, when sending, receiving or both. The processors can communicate the resulting data with the rest of the processors connected with it. A processor node can be viewed as a cell that carries out only one specific evolutionary operation (substitution, elimination or insertion). NEPT uses these operations in order to generate recursively enumerable languages recognized by other NEP (without filtering strategy). On the other hand, NPEP uses these operations together with a valuation mapping (from strings to integers) to generate strings labeled with an electrical polarization. Each node has their own polarization, then the filtering strategy consists in let pass those strings with their same polarization.

NEP architecture sees a biological process like a web, that is, a network of several NEP working in a collaborative way. This architecture consist in three layers each one representing one block of computing: *selection layer* (implemented by a NEPT), represents the reception of an extracellular signal molecule arriving at the cellular membrane, and its alteration (transduction); *control layer* (implemented by a NPEP), realizes the monitoring functionality of signaling pathway, either activates or inactivates the target proteins (effectors); and finally the *processing layer* (implemented by one or several NEP), represents the target activity which alters the metabolism.

A very well known metabolic process is the *Krebs cycle* that is critical in cellular respiration. Signaling by calcium helps to activate metabolic pathways, such as the *malate-aspartate shuttle* pathway (MAS). Krebs cycle and MAS are linked through shared substrates as the  $\alpha$ -ketoglutarate ( $\alpha$ KG). The interplay between these processes, sharing and competing for  $\alpha$ KG, as well as being altered by calcium is an important study of the brain stimulation *in vivo*. In order to model the interplay between these biological processes, we define a NEPT in the selection layer, which is able to translate strings representing chemical compounds into new specific strings representing some receptor proteins, enzymes and metabolites (i.e. piruvate, malate, aspartate, calcium, etc). These strings are collected in the output node and are sent to the input node of NPEP in the control layer. NPEP recognizes these strings and generates new polarized strings that are collected in their output node: a negative charge means an inhibitor substance for MAS, a neutral charge represents a promoter for MAS and positive charge represents a promoter for Krebs cycle. Resulting polarized strings with a negative or neutral polarization are sent to the input node of the NEP representing the MAS shuttle and the string with a positive charge are sent to the NEP for Krebs cycle. Finally in the NEPs, each node receive only the necessary substances (filtered by the input strategy), transforming them (using substitution, elimination or insertion rules representing the corresponding chemical transformation) and only the necessary substances for the linked nodes (filtered by the output strategy) are sent as can be seen in Fig. 1 and 2.

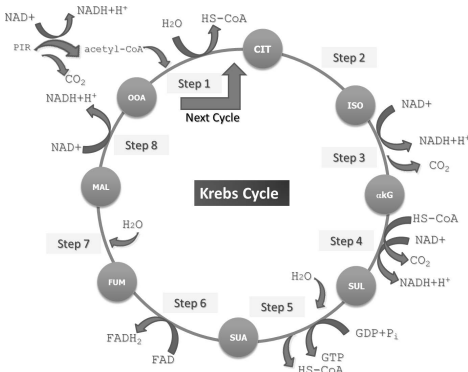


Fig. 1. Krebs cycle

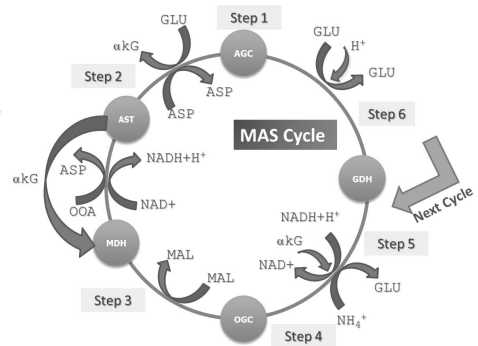


Fig. 2. MAS shuttle pathway

## References

1. Alarcón, P.P., Arroyo, F., Mitrana, V.: Networks of Polarized Evolutionary Processors as Problem Solvers. In: Advances in Knowledge-Based and Intelligent Information and Engineering Systems, pp. 807–815 (2012)
2. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Networks of evolutionary processors. Acta Informática 39, 517–529 (2003)
3. Gómez Canaval, S., Mitrana, V., Sánchez Couso, J.R.: Transducers Based on Networks of Evolutionary Processors (submitted)

# On String Reading Stateless Multicounter 5' → 3' Watson-Crick Automata (Extended Abstract)

László Hegedüs and Benedek Nagy

Department of Computer Science, Faculty of Informatics,  
University of Debrecen, Debrecen, Hungary  
{hegedus.laszlo,nbenedek}@inf.unideb.hu

Counter machines are finite state automata equipped with a fixed, finite number of counters. The machine can check whether a counter is zero or not. In each step, a counter's value can be increased by one, decreased by one, or left unchanged. Counter machines with two counters are Turing universal [5]. 5' → 3' Watson-Crick automata are two-head finite automata whose reading heads start from the two opposite ends of the input in the beginning of the computation [3,4]. Stateless multicounter 5' → 3' Watson-Crick automata were defined in [1,2]. The following is a more general definition, allowing to read strings in a transition.

**Definition 1.** We denote by  $M = (r, m, \Sigma, \delta, \mathfrak{t}, \$)$  a nondeterministic stateless multicounter 5' → 3' WK-automaton with  $m$  counters and radius  $r$ , where

- $m \in \mathbb{N}_0$  is the number of counters,
- $r \in \mathbb{N}, r \geq 2$  is the radius of the machine, (the case  $r = 2$  was already analysed in [1,2])
- $\Sigma$  is a nonempty input alphabet,
- $\delta : V \times \{1, 0\}^m \times \{0, 1, 2, \dots, r, \infty\} \rightarrow 2^{\{\{0,1\} \times \{0,1\} \times \{0,+,-\}^m\}}$  is the transition function (the heads may step through on the read string or stay and the counters may be increased, decreased or unchanged depending on the read strings, the actual signs – that is 0 or positive (1) – of the counters, and on the distance of the heads if it is less than or equal to  $r$ ),
- $\mathfrak{t}, \$ \notin \Sigma$  are two special symbols called end-markers.

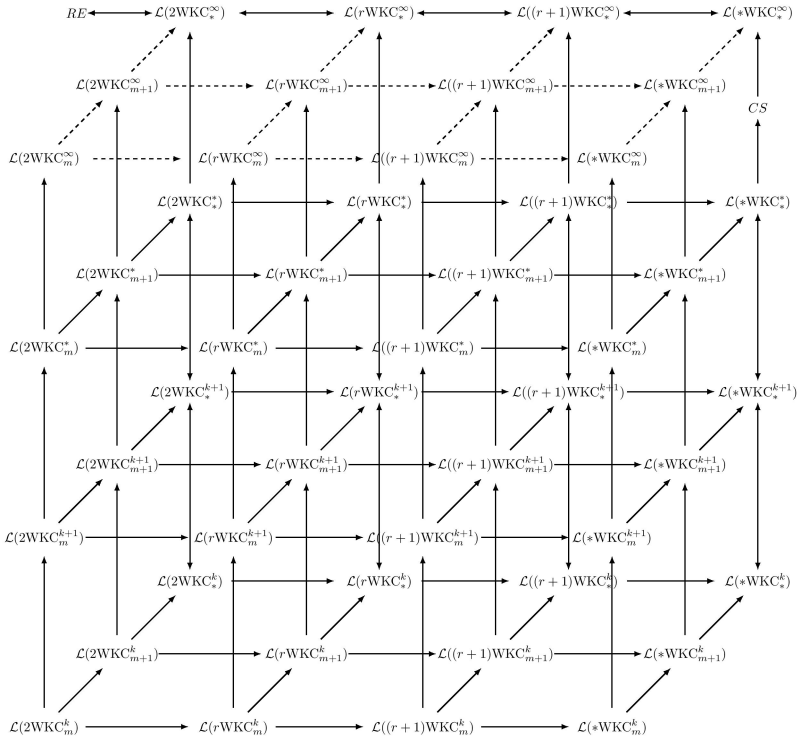
Here,  $V = \{(u, v) \mid u \in \{\mathfrak{t}, \varepsilon\}\Sigma^*, v \in \Sigma^*\{\$, \varepsilon\}, |u| > 0, |v| > 0, \text{ and } |uv| \leq r\}$ .

The set of (string reading)  $k$ -reversal deterministic stateless multicounter 5' → 3' WK-automata with  $m$  counters and radius  $r$  is denoted by  $r\text{WKC}_m^k$ . Automata that are not  $k$ -reversal for any  $k \in \mathbb{N}$  are denoted by  $\text{WKC}^\infty$ . Further,

$$\begin{aligned} r\text{WKC}_m^* &= \bigcup_{k=0}^\infty r\text{WKC}_m^k & \text{and} & & r\text{WKC}_*^k &= \bigcup_{m=0}^\infty r\text{WKC}_m^k \\ r\text{WKC}_*^* &= \bigcup_{k=0}^\infty r\text{WKC}_*^k = \bigcup_{m=0}^\infty r\text{WKC}_m^* = \bigcup_{k=0}^\infty \bigcup_{m=0}^\infty r\text{WKC}_m^k \\ * \text{WKC}_m^k &= \bigcup_{r=2}^\infty r\text{WKC}_m^k. \end{aligned}$$

Let  $\mathfrak{t}w\$ = xyz$  be the input of an arbitrary, stateless multicounter 5' → 3' WK-automaton.  $x[z]y$  is used to denote that the left and right heads have read  $x$  and  $y$  respectively and subword  $z$  is still to be processed. A configuration

of a machine  $M$  is denoted by  $(x[z]y, (C_1, \dots, C_m))$ . The *start* configuration is  $([\$w\$], (0, \dots, 0))$ , while the *accepting* configurations are  $(x][y, (0, \dots, 0))$ , where  $\$w\$ = xy$ . The set of accepted words form the accepted language. The hierarchy of the classes of accepted languages is represented in Fig. 1. One-way arrows denote the proper subset relation, while two-way arrows represent equalities. Dashed arrows denote the subset relation, but it is still an open question whether those classes are proper subsets of each-other or they are equal.



**Fig. 1.** Hierarchy on the number of counters, reversals and the size of the radius (here  $r > 2, k \geq 1, m \geq 1$ ) for deterministic automata.  $\mathcal{L}(x)$  represents the language family accepted by the automata type  $x$ ;  $CS$  and  $RE$  stand for the classes of context-sensitive and recursively enumerable languages, respectively.

**References**

1. Egecioglu, Ö., Hegedüs, L., Nagy, B.: Stateless multicounter  $5' \rightarrow 3'$  Watson-Crick automata. In: IEEE Fifth Int. Conf. BIC-TA, pp. 1599–1606 (2010)
2. Hegedüs, L., Nagy, B., Egecioglu, Ö.: Stateless multicounter  $5' \rightarrow 3'$  Watson-Crick automata: the deterministic case. Natural Computing 11, 361–368 (2012)
3. Nagy, B.: On  $5' \rightarrow 3'$  sensing Watson-Crick finite automata. In: Garzon, M.H., Yan, H. (eds.) DNA 2007. LNCS, vol. 4848, pp. 256–262. Springer, Heidelberg (2008)
4. Freund, R., Păun, G., Rozenberg, G., Salomaa, A.: Watson-Crick Finite Automata. In: Third Ann. DIMACS Symp. on DNA Based Computers, pp. 535–546 (1994)
5. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., Upper Saddle River (1967)

# Relating Transition P Systems and Spiking Neural P Systems (Extended Abstract)

Richelle Ann B. Juayong, Nestine Hope S. Hernandez,  
Francis George C. Cabarle, and Henry N. Adorna

Algorithms & Complexity Lab, Department of Computer Science  
University of the Philippines Diliman, 1101 Quezon City, Philippines  
{rbjuayong, nshernandez, fccabarle, hnadorna}@up.edu.ph

**Abstract.** In this work we provide a relationship between Transition P systems with noncooperative rules or nTP systems and Spiking Neural P systems with weighted synapses or wSNP systems. In particular we define a reasonable operating mode of nTP systems we refer to as  $k$ -restricted object-minimal region-maximal parallelism or  $k$ - $O_{min}, R_{max}$  mode. We show that there exists a simulation of nTP systems operating in  $1$ - $O_{min}, R_{max}$  mode with wSNP systems.

**Keywords:** Membrane Computing, wSNP systems, nTP systems, simulation.

Cell-like and neural-like P systems differ not only in their structures but also in their modes of operation. In this work we provide a relationship between (cell-like) Transition P systems with noncooperative rules or nTP systems and (neural-like) Spiking Neural P systems with weighted synapses or wSNP systems. The goal is to use this work to realize further relations of cell- and neural- like systems. Readers are assumed to be familiar with Membrane Computing basics [1] and formal language theory, including notations and definitions for an nTP system without dissolution [3] and for a wSNP system [5].

In most P system models, rules are applied in a nondeterministic and maximally parallel manner. We impose a restriction on this manner of applying rules in which we require that maximal parallelism only applies to regions while objects per region are  $k$ -restricted minimally parallel (similar to its description in [2]) in the objects per region. This is described in the following definition.

**Definition 1.** *An nTP system without dissolution  $\Pi_{nTP}$  that runs in  $k$ -restricted, object-minimal, region-maximally parallel mode (denoted by  $k$ - $O_{min}, R_{max}$  mode) requires that per region, all objects that can evolve must evolve (region-maximally parallel). If an object is applicable in a region, at most  $k$  copies of an object can evolve in a time step ( $k$ -restricted minimally parallel).*

We now define a notion of simulation in the context of P systems, as adapted from [4]. For any P system  $\Pi$ , we denote a transition from configuration  $C_{\Pi}^i$  at

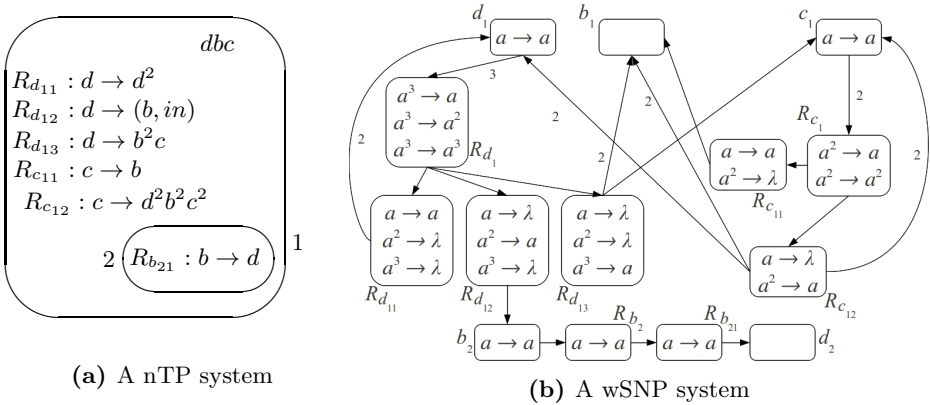


time  $i$  to  $C_{\Pi}^{i+1}$  at time  $i+1$  as  $C_{\Pi}^i \Rightarrow C_{\Pi}^{i+1}$ . We then relate nTP systems without dissolution to wSNP systems using this simulation notion.

**Definition 2.** Let  $\Pi$  and  $\Pi'$  be two P systems and let  $S$  be a binary relation over configurations in  $\Pi$  and  $\Pi'$ .  $S$  is a simulation over  $\Pi$  and  $\Pi'$  if whenever  $C_{\Pi}^i S C_{\Pi'}^j$ , then if  $C_{\Pi}^i \Rightarrow C_{\Pi}^{i+1}$ , there exists  $C_{\Pi'}^{j'}$  such that  $C_{\Pi'}^j \xRightarrow{*} C_{\Pi'}^{j'}$ ,  $j < j'$  and  $C_{\Pi}^{i+1} S C_{\Pi'}^{j'}$ . We say  $C_{\Pi'}^j$  simulates  $C_{\Pi}^i$  and  $\Pi'$  simulates  $\Pi$ .

**Theorem 1.** For every nTP system  $\Pi_{nTP}$  without dissolution that runs in a nondeterministic and  $1-O_{min}, R_{max}$  mode, there exists a wSNP system  $\Pi_{wSNP}$  that simulates  $\Pi_{nTP}$ .

*Proof idea.* Given an  $\Pi_{nTP}$  without dissolution we construct a  $\Pi_{wSNP}$  such that there exists a simulation relation  $S$  over configurations in  $\Pi_{wSNP}$  and  $\Pi_{nTP}$ . In particular we have  $(C_{nTP}^i, C_{wSNP}^{3i}), (C_{nTP}^i, C_{wSNP}^{3i+1}), (C_{nTP}^i, C_{wSNP}^{3i+2}) \in S$ . Figure 1 provides an illustration of this simulation.



**Fig. 1.** Illustration of Theorem 1: (b) is a wSNP system simulating the nTP system without dissolution in (a).

## References

1. Ciobanu, G., Păun, G., Pérez-Jiménez, M.J.: Chapter 1 Introduction to Membrane Computing. Applications of Membrane Computing. Springer (2006)
2. Freund, R., Verlan, S.: (Tissue) P systems working in the  $k$ -restricted minimally or maximally parallel transition mode. Natural Computing 10, 821–833 (2011)
3. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J.: Computing Backwards with P systems. In: WMC 2010, Curtea de Argeş, Romania, pp. 282–295 (2009)
4. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press (1999)
5. Wang, J., Hoogetboom, H.J., Pan, L., Păun, G., Pérez-Jiménez, M.J.: Spiking Neural P Systems with Weights. Neural Computation 22, 2615–2646 (2010)

# Author Index

- Adamatzky, Andrew 79  
Adorna, Henry N. 259  
Ahmed, Saif 244  
Alam Anik, Md. Tanvir 244  
Alhazov, Artiom 246  
Allerdißen, Merle 232  
Aman, Bogdan 248  
Arroyo, Fernando 255  
Asiedu, Isaac Kobina 162
- Bakaoukas, Anastasios G. 250  
Beggs, Edwin 6  
Bianchi, Maria Paola 19  
Boian, Elena 246  
Bournez, Olivier 31  
Bringsjord, Selmer 102
- Cabarle, Francis George C. 259  
Calude, Cristian S. 43  
Ciobanu, Gabriel 248  
Cojocaru, Svetlana 246  
Colesnicov, Alexandru 246  
Costa, José Félix 6  
Csuhaj-Varjú, Erzsébet 55
- de Lacy Costello, Ben 79
- Ehrenfeucht, Andrzej 3
- Fernau, Henning 67  
Fiske, Michael Stephen 252  
Formenti, Enrico 1  
Foughmand-Araabi, Mohammad-Hadi 90  
Freund, Rudolf 67
- Gale, Ella 79  
Goliaei, Sama 90  
Gómez Canaval, Sandra 255  
Govindarajulu, Naveen Sundar 102  
Greiner, Rinaldo 232  
Grigoriev, Dima 113
- Hegedüs, László 257  
Hernandez, Nestine Hope S. 259
- Ivanov, Sergiu 67
- Juayong, Richelle Ann B. 259
- Kari, Lila 125  
Kopecki, Steffen 125  
Krithivasan, Kamala 186
- Lefèvre, Jonas 31  
Licato, John 102
- Makowiec, Danuta 138  
Malahov, Ludmila 246  
Manzoni, Luca 150  
Mereghetti, Carlo 19  
Mizuki, Takaaki 162
- Nagy, Benedek 257
- Padilla, Jennifer E. 174  
Palano, Beatrice 19  
Patitz, Matthew J. 174  
Pena, Raul 174  
Petic, Mircea 246  
Poças, Diogo 6  
Porreca, Antonio E. 150
- Ramanujan, Ajeesh 186  
Richter, Andreas 232  
Rogozhin, Yurii 246  
Rozenberg, Grzegorz 3  
Russell, Benjamin 198, 209
- Sánchez, José Ramón 255  
Schmid, Markus L. 67  
Schweller, Robert T. 174  
Seeman, Nadrian C. 174  
Seki, Shinnosuke 220  
Sheline, Robert 174  
Shpilrain, Vladimir 113  
Simjour, Amirhossein 125  
Sone, Hideaki 162  
Stepney, Susan 198, 209

Subramanian, K.G. 67  
Summers, Scott M. 174

Tadaki, Kohtaro 43  
Tucker, John V. 6

Vaszi, György 55  
Voigt, Andreas 232

Zhong, Xingsi 174