LNCS 3699

Cristian S. Calude
Michael J. Dinneen
Gheorghe Păun
Mario J. Pérez-Jiménez
Grzegorz Rozenberg (Eds.)

Unconventional Computation

4th International Conference, UC 2005
Sevilla, Spain, October 2005
Proceedings

Springer

Cristian S. Calude   Michael J. Dinneen
Gheorghe Păun   Mario J. Pérez-Jiménez
Grzegorz Rozenberg (Eds.)

Unconventional Computation

4th International Conference, UC 2005
Sevilla, Spain, October 3 – 7, 2005
Proceedings

Springer

Volume Editors

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

Gheorghe Păun
Mario J. Pérez-Jiménez
Sevilla University, Department of Computer Science and Artificial Intelligence, Spain
E-mail: {gpaun, marper}@us.es

Grzegorz Rozenberg
Leiden University
Leiden Center for Natural Computing
Leiden, The Netherlands
E-mail: grozenberg@gmail.com

Library of Congress Control Number: 2005932252

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

ISSN	0302-9743
ISBN-10	3-540-29100-8 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-29100-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11560319 06/3142 5 4 3 2 1 0

Preface

The Fourth International Conference on **Unconventional Computation, UC 2005**, organized under the auspices of EATCS by the Centre for Discrete Mathematics and Theoretical Computer Science and the Department of Computer Science and Artificial Intelligence of the University of Seville, was held in Seville, October 3–7, 2005.

Seville, one of the most beautiful cities in Spain, is at its best in October. An explosion of colour and contrast: flamenco, bullfighting, and a lively atmosphere in the streets due to the open and friendly nature of its people. The river Guadalquivir, the Cathedral and the Golden Tower are all places full of magic where the visitor can feel the spirit of a city which is eternally romantic.

The series of International Conferences **Unconventional Computation (UC)**, `https://www.cs.auckland.ac.nz/CDMTCS/conferences/uc/` is devoted to all aspects of unconventional computation, theory as well as experiments and applications. Typical, but not exclusive, topics are: natural computing including quantum, cellular, molecular, neural and evolutionary computing; chaos and dynamical systems based computing; and various proposals for computations that go beyond the Turing model.

The first venue of the Unconventional Computation Conference (formerly called Unconventional Models of Computation) was Auckland, New Zealand in 1998; subsequent sites of the conference were Brussels, Belgium in 2000 and Kobe, Japan in 2002.

The titles of the proceedings volumes from past UC Conferences are as follows:

1. C.S. Calude, J. Casti, M.J. Dinneen (eds.). *Unconventional Models of Computation*, Springer-Verlag, Singapore, 1998, viii + 426 pp. ISBN: 981-3083-69-7.
2. I. Antoniou, C.S. Calude, M.J. Dinneen (eds.). *Unconventional Models of Computation, UMC'2K*, Springer-Verlag, London, December 2000, xi + 301 pp. ISBN 1-85233-417-0.
3. C.S. Calude, M.J. Dinneen, F. Peper (eds.). *Third International Conference, UMC 2002*, Lecture Notes in Computer Science, Vol. 2509, Springer-Verlag, Heidelberg, 2002, vii + 331 pp. ISBN: 3-540-44311-8.

The Steering Committee of the series of International Conferences **Unconventional Computation** includes T. Bäck (Leiden, The Netherlands), C.S. Calude (Auckland, NZ, co-chair), L.K. Grover (Murray Hill, NJ, USA), J. van Leeuwen (Utrecht, The Netherlands), S. Lloyd (Cambridge, MA, USA), Gh. Păun (Seville, Spain and Bucharest, Romania), T. Toffoli (Boston, MA, USA), C. Torras (Barcelona, Spain), G. Rozenberg (Leiden, The Netherlands and Boulder, Colorado, USA, co-chair), A. Salomaa (Turku, Finland).

The five invited speakers of the conference were:

T. Bäck (Leiden, The Netherlands): *Using genetic algorithms to evolve behaviour in cellular automata*
L. Grover (Murray Hill, USA): *Quantum searching amidst uncertainty*
S. Istrail (Providence, USA): *Logic functions of the genomic cis-regulatory code*
N. Seeman (New York, USA): *Structural DNA nanotechnology: Molecular constructions and computations*
C. Torras (Barcelona, Spain): *Natural inspiration for artificial adaptivity: Some neurocomputing experiences in robotics*

UC 2005 included the following three tutorials:

S. Istrail (Providence, USA): *Logic of networks*
I. Petre (Turku, Finland) and G. Rozenberg (Leiden, The Netherland): *Computing with living cells*
Gh. Păun (Seville, Spain): *Elementary aspects of membrane computing*

The Programme Committee thanks the much appreciated work done by the paper reviewers for the conference. These experts were:

S. Basu	D. Karig	M. Margenstern	M. A. Stay
H.-G. Beyer	T. D. Kieu	G. Mauri	K. Svozil
M. Burgin	S. Kobayashi	K. Morita	H. Umeo
C. S. Calude	O. H. Ibarra	M. Mozer	H. T. Wareham
G. Csardi	V. V. Ivanov	B. Ömer	R. Weiss
M. J. Dinneen	N. Jonoska	Gh. Păun	T. Yokomori
P. Érdi	N. Krasnogor	M. J. Pérez-Jiménez	
A. Ekert	J. van Leeuwen	I. Petre	
M. P. Frank	N. Lucas	U. Speidel	

The Programme Committee consisting of L. Accardi (Rome, Italy), H.-G. Beyer (Dornbirn, Austria), M. Burgin (Los Angeles, USA) C. S. Calude (chair; Auckland, NZ), M. J. Dinneen (secretary; Auckland, NZ), P. Érdi (Kalamazoo, USA), A. Ekert (Cambridge, UK), M. P. Frank (Tallahassee, USA), V. V. Ivanov (Dubna, Russia), N. Jonoska (Tampa, USA), N. Krasnogor (Nottingham, UK), J. van Leeuwen (Utrecht, The Netherlands), K. Morita (Hiroshima, Japan), M. Mozer (Boulder, USA), Gh. Păun (Seville, Spain), I. Petre, (Turku, Finland), H. Umeo (Osaka, Japan), R. Weiss (Princeton, USA), T. Yokomori (Tokyo, Japan), selected 18 papers (out of 29) to be presented as regular contributions.

We extend our thanks to all members of the Conference Committee, particularly to M. Cavaliere, C. Graciani Díaz, M. A. Gutiérrez Naranjo, A. Nepomuceno Fernández, Gh. Păun, M. J. Pérez Jiménez (chair), F. J. Romero

Campero, A. Riscos Núñez, A. Romero Jiménez, F. Sancho Caparrini, D. Sburlan, and U. Speidel (registration), for their invaluable organizational work.

We thank the University of Seville and the Centre for Discrete Mathematics of the University of Auckland for their technical support. The hospitality of our hosts, the Department of Computer Science and Artificial Intelligence of the University of Seville, is much appreciated. The conference was partially supported by the project TIN2004-23021-E of the Ministerio de Educación y Ciencia of Spain, by the II Plan Propio of the University of Seville, and by the Acción Coordinada IMUS 2003 of the Junta de Andalucía: we extend to all our gratitude.

It is a great pleasure to acknowledge the fine co-operation with the Lecture Notes in Computer Science team of Springer for producing this volume in time for the conference.

July 2005

C. S. Calude
M. J. Dinneen
Gh. Păun
M. J. Pérez Jiménez
G. Rozenberg

Table of Contents

Using Genetic Algorithms to Evolve Behavior in Cellular Automata

Thomas Bäck[1,2] and Ron Breukelaar[1,⋆]

[1] Universiteit Leiden, LIACS, P.O. Box 9512, 2300 RA Leiden, The Netherlands
{baeck, rbreukel}@liacs.nl
[2] NuTech Solutions GmbH, Martin Schmeißer Weg 15, 44227 Dortmund, Germany
{baeck}@nutechsolutions.de

Abstract. It is an unconventional computation approach to evolve solutions instead of calculating them. Although using evolutionary computation in computer science dates back to the 1960s, using an evolutionary approach to program other algorithms is not that well known. In this paper a genetic algorithm is used to evolve behavior in cellular automata. It shows how this approach works for different topologies and neighborhood shapes. Some different one dimensional neighborhood shapes are investigated with the genetic algorithm and yield surprisingly good results.

1 Introduction

Evolutionary Algorithms is the name for the algorithms in the field of Evolutionary Computation which is a subfield of Natural Computing and already exists more than 40 years. It was born from the idea to use principles of natural evolution as a paradigm for solving search and optimization problem in high-dimensional combinatorial or continuous search spaces. The most widely known instances are genetic algorithms [9,10,11], genetic programming [12,13], evolution strategies [16,17,18,19], and evolutionary programming [7,6]. A detailed introduction to all these algorithms can be found e.g. in the Handbook of Evolutionary Computation [1].

Today the Evolutionary Computation field is very active. It involves fundamental research as well as a variety of applications in areas ranging from data analysis and machine learning to business processes, logistics and scheduling, technical engineering, and others. Across all these fields, evolutionary algorithms have convinced practitioners by the results obtained on hard problems that they are very powerful algorithms for such applications. The general working principle of all instances of evolutionary algorithms is based on a program loop that involves simplified implementations of the operators mutation, recombination, selection, and fitness evaluation on a set of candidate solutions (often called a

⋆ Part of the research was funded by the Foundation for Fundamental Research on Matter (FOM), Utrecht, The Netherlands, *project: "An evolutionary approach to many-parameter physics"*.

population of individuals) for a given problem. In this general setting, mutation corresponds to a modification of a single candidate solution, typically with a preference for small variations over large variations. Recombination corresponds to an exchange of components between two or more candidate solutions. Selection drives the evolutionary process towards populations of increasing average fitness by preferring better candidate solutions to proliferate with higher probability to the next generation than worse candidate solutions. By fitness evaluation, the calculation of a measure of goodness associated with candidate solutions is meant, i.e., the fitness function corresponds to the objective function of the optimization problem at hand.

No attempt will be made to give a complete introduction or overview of evolutionary algorithms, as there are many good introductory books on the topic available, e.g. [1], instead an example of how to use evolutionary algorithms to parameterize other algorithms will be given.

Evolving parameters for complex algorithms could also be viewed as an inverse design problem, i.e. a problem where the target design (behavior of the algorithm to be parameterized) is known, but the way to achieve this is unknown. The inverse design of cellular automata (CA) is such a problem. Cellular automata are used in many fields to generate a global behavior with local rules. Finding the rules that display a desired behavior can be a hard task especially when it comes to real world problems. This paper uses a genetic algorithm to generate the transition rules for cellular automata, thus evolving global behavior with local rules using a genetic base.

2 Cellular Automata

According to [20] cellular automata (CA) are mathematical idealizations of physical systems in which space and time are discrete, and physical quantities take on a finite set of discrete values. The simplest CA is one dimensional and can be viewed as an array of ones and zeros of width N, where the first position of the array is linked to the last position. In other words, defining a row of positions $C = \{a_1, a_2, ..., a_N\}$ where C is a CA of width N and a_N is adjacent to a_1.

The neighborhood s_n of a_n is defined as the local set of positions with a distance to a_n along the connected chain which is no more than a certain radius (r). This for instance means that $s_2 = \{a_{148}, a_{149}, a_1, a_2, a_3, a_4, a_5\}$ for $r = 3$ and $N = 149$. Note that for one dimensional CA the size of the neighborhood is always equal to $2r + 1$.

The values in a CA can be altered all at the same time (synchronous) or at different times (asynchronous). Only synchronous CA are considered in this paper. In the synchronous approach at every time step (t) every cell state in the CA is recalculated according to the states of the neighborhood using a certain transition rule $\Theta : \{0,1\}^{2r+1} \rightarrow \{0,1\}, s_i \rightarrow \Theta(s_i)$. This rule basically is a one-to-one mapping that defines an output value for every possible set of input values, the input values being the 'state' of a neighborhood. The state of a_n at time t is written as a_n^t, the state of s_n at time t as s_n^t and the state of the entire CA C

at time t as C^t so that C^0 is the initial state and $\forall n = 1, \ldots, N$ $a_n^{t+1} = \Theta(s_n^t)$. Given $C^t = \{a_1^t, ..., a_N^t\}$, C^{t+1} can be defined as $\{\Theta(s_1^t), ..., \Theta(s_N^t)\}$.

Because $a_n \in \{0, 1\}$ the number of possible states of s_n equals 2^{2r+1}. Because all possible binary representations of m where $0 \leq m < 2^{2r+1}$ can be mapped to a unique state of the neighborhood, Θ can be written as a row of ones and zeros $R = \{b_1, b_2, ..., b_{2^{2r+1}}\}$ where b_m is the output value of the rule for the input state that maps to the binary representation of $m - 1$. A rule therefore has a length that equals 2^{2r+1} and so there are $2^{2^{2r+1}}$ possible rules for a binary one dimensional CA. This is a huge number of possible rules (if $r = 3$ this sums up to about $3,4 \times 10^{28}$) each with a different behavior.

One of the interesting things about these and other CA is that certain rules tend to exhibit organizational behavior, independently of the initial state of the CA. This behavior also demonstrates there is some form of communication going on in the CA over longer distances than the neighborhood allows directly. In [14] the authors examine if these simple CA are able to perform tasks that need positions in a CA to work together and use some form of communication. One problem where such a communication seems required in order to give a good answer is the Majority Problem (as described in section 3). A genetic algorithm is used to evolve rules for one dimensional CA that do a good job of solving the Majority Problem [14] and it is shown how these rules seem to send "particles" and communicate by using these particles [15]. These results imply that even very simple cells in one dimensional cellular automata can communicate and work together to form more complex and powerful behavior.

Previous work [4] suggested that using multi dimensional CA works a lot better than using one dimensional CA. This paper will try to shed light on what effect a different topology or neighborhood shape can have, and a measure will be given to classify these different CA.

3 Majority Problem

One of the best known global problems that is (partly) solvable with local rules if the Majority Problem. The Majority Problem can be defined as follows:

Given a set $A = \{a_1, ..., a_n\}$ with n odd and $a_m \in \{0, 1\}$ for all $1 \leq m \leq n$, answer the question: 'Are there more ones than zeros in A?'.

The Majority Problem first does not seem to be a very difficult problem to solve. It seems only a matter of counting the ones in the set and then comparing them to the number of zeros. Yet when this problem has to be solved within the framework of a CA it becomes a lot more difficult. This is because the rule in a CA does not let a position look past its neighborhood and that is why the cells all have to work together and use some form of communication.

Given that the relative number of ones in C^0 is written as λ, in a simple binary CA the Majority Problem can be defined as:

Find a transition rule that, given an initial state of a CA with N odd and a finite number of iterations to run (I), will result in an 'all zero' state if $\lambda < 0.5$ and an 'all one' state otherwise. The 'all zero' state being the state in which

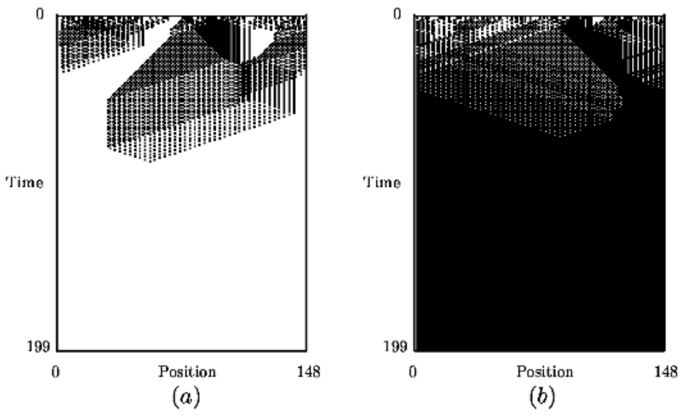


Fig. 1. These are examples of majority problem classification by the rule found by David, Forrest and Koza. [5]. Both are correct classifications (*a*) with 74 ones in the initial state, (*b*) with 75.

every cell in the CA is zero and the 'all one' state being a the state in which every cell is one.

Evaluating a transition rule for this problem is done by iterating M randomly generated initial states and calculating the relative number of correct classification. The fitness of a transition rule is denoted with $F_{N,M}$ where N is the width of the CA. The fitness can be calculated with different distributions over the number of ones in the initial state, but the default is a binomial distribution (denoted with $F^B_{N,M}$) where every cell in the CA has a 50% chance of being initiated with a one for every initial state.

The first intuitive rule to come up with is the 'majority rule'. This being the rule where the output value is 1, if the number of ones in the neighborhood is more than the number of zeros, and a zero otherwise. Surprising as it may seem this does not at all solve the problem. The majority rule gets stuck on the problem that on the boundary thick line in the time plot the cell can't "agree" on the global answer. The cell just left of such a thick line is zero and because all other cells left of it in the neighborhood are also zero, it "decides" to stay that way. Yet its neighboring cell to the right is one and sees only ones on its right and therefore decides to stay one. This way the information fails to propagate through the CA and classification fails.

Researchers in the field of cellular automata have published many different rules to solve this problem, one such rule is the GKL rule after Gacs, Kurdyumov and Levin [8]. This rule is pretty good at classifying the majority problem and does it for 81.6% of the test cases with a width of 149 cells. For 17 years this was the best rule and then L. Davis found a better one in 1995 which did 81.8%. In the same year R. Das found a rule that did 82.178%. Then in 1996 David, Forrest and Koza found a rule by cleverly using genetic programming that was able to classify 82.326% correctly [5].

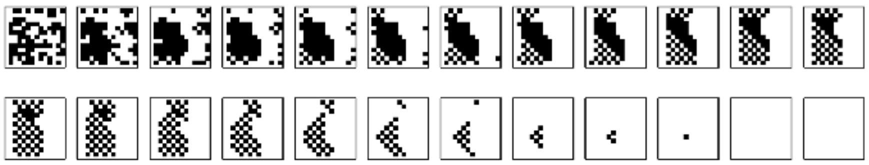


Fig. 2. This figure shows a correct classification of the Majority Problem by a two dimensional CA with both width and height equal to 13 and $\lambda = 84/169$. The transition rule was one of the best tested in the experiment and scored $F_{169,10^3} = 0.715$.

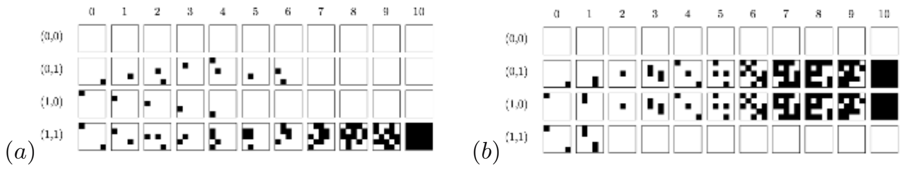


Fig. 3. This figure shows two iteration runs of transition rules for two dimensional CA that were evolved using a GA. (a) shows how a CA can behave like an AND-port and (b) shows how it can behave like an XOR-port.

Although these rules are very impressive it is believed that there is no definite solution for the problem as long as the neighborhood is smaller than the size of the CA. It is already a big accomplishment for a CA to get 70 percent of all random initial states correct, for this shows there is some kind of communication going on; some kind of emerging behavior.

4 Inverse Design

Nature has some remarkable examples of local rules that exhibit global behavior. Ant colonies are a good example. An individual ant does not seem to be very intelligent and does not seem to know what is doing and why, but a colony of ants seem very organized and purposeful. This did not happen over night, but evolved over millions of generations.

The global behavior of a CA can also be evolved in a similar way by evolving the local behavior. This "inverse design" of the behavior is done by using a genetic algorithm to evolve the transition rule that described the behavior. The fitness function of the genetic algorithm then defines the desired behavior.

M. Mitchell, J. P. Crutchfield and P. T. Hraber have shown [14,15] that using a simple GA to evolve transition rules for the majority problem (explained in Section 3) can already give surprisingly good results. About half of the rules that were found performed better than the most trivial rule and about 7 rules out of 300 rules were found that seemed to use some primitive form of communication that worked for more than 70% of the classifications. This is not better than rules that are made by hand, but it does show how a GA can evolve global behavior based on local rules.

In [3] this idea was extended to two dimensional CA and it was shown that two dimensional CA can match the performance of the one dimensional rules even though the neighborhood was two cells smaller. It was also shown that different kinds of problems can be solved using this approach (AND-, XOR-problem and pattern generation) if it is extended to two dimensions. Figure 3 shows how a two dimensional CA can be evolved to behave like an AND or XOR port.

A generalization was defined in [2,4] so that the approach could work for n-dimensional CA and different sizes of neighborhoods. The GA was altered to make it more robust and some experiments were done to compare results for different dimensionality's on identical problems. The way the transition rules were tested was also altered to more accurately describe the desired behavior. The results in [4] suggested that it is a lot easier to find rules for a three dimensional CA than it is for a one dimensional CA. Even though the number of cells in the neighborhoods are identical.

This article will more strongly support the claim that the topology of the CA and its neighborhood is very important for the performance of that CA. Four different neighborhood shapes are chosen to show this relation and to give indication in what shape of neighborhood might have the best performance and why.

5 Distance Measure

A normal one dimensional CA (as described in Section 2) has a very simple defined neighborhood, being "all the cells within a certain radius r of the center cell". Although this seems to be the most logical neighborhood CA can have many different shapes.

Every iteration step in a synchronous CA the state of every cell is updated using the information in the cells of the neighborhood of that cell. That means the way the information moves ("travels") through the CA is defined by the shape of the neighborhood. Note that because the standard one dimensional neighborhood is symmetrical, if information travels from cell a_i to cell a_j it will also travel from cell a_j to cell a_i. The number of iterations it takes for information to travel from cell a_i to cell a_j can be called the "distance between a_i and a_j". If the shape of the neighborhood is different, the distance between cells in the CA will be changed too. Not only will this change the behavior of the CA, but it might also change the possible behaviors of the CA. That means that two neighborhood shapes might performance different on the same problem.

Intuitively, to solve a global problem with local rules information from every cell in the CA has to travel to every other cell in the neighborhood to be combined in a way that suits the problem. Because combining information results in new information and the space in a CA is fixed, there will be information loss. Therefore combining the information as fast as possible seems to be good way to counter this information loss and solve the problem in the best possible way. This then would mean that the distance between two cells in the CA need to be minimized so that the information can be combined faster.

To measure the rate at which information is combined in a CA two metrics are defined. The “maximum distance between two cells” and the “average distance between two cells” in a CA. Because every cell has the same neighborhood shape every cell has the same maximum distance to a cell and the same average distance to all the cells in the CA. Note that these metrics are very dependent on the shape of the neighborhood and the size and topology of the CA.

If it is true that minimizing the distance in a CA will improve the performance of the CA then this might explain why multi dimensional CA seem to be more powerful than the one dimensional CA [3,2]. The topology of multi dimensional CA supports information travel in multiple directions and this decreases the distance between cells in the CA. An experiment was conducted to compare different neighborhood shapes in combination with the well known Majority Problem (see Section 3) and measure what the impact of the different distance measures is on the performance. In order to test the performance of a neighborhood shape a GA was used to search for a good transition rule that solves the Majority Problem like was done in [2,3,4,14,15].

6 The Genetic Algorithm

The GA in this paper is the same as used in [4]. This GA evolves a pool of 100 transition rules to find the one with the best performance. It uses tournament selection as defined in [1] to select which individuals to keep alive. This involves running ‘tournaments’ on the population in order to determine the next generation. Every tournament q individuals are selected at random from generation t and the one with the highest fitness is then copied to generation $t + 1$. This is repeated until generation $t + 1$ has the same number of individuals as generation t. In this experiment $q = 20$.

After selection recombination is applied to generation $t + 1$. Recombination is done by using single-point crossover on a subset of the population and mutating the resulting individuals using probabilistic bit flip. The relative number of individuals that are used in the crossover is denoted by the crossover-rate c. Mutation is done by flipping every bit in the individual with a probability m. In this experiment $c = 0.9$ and $m = 2/l = 1/64 = 0.15625$.

All the individuals in the pool are initialized at random with a normal distribution over the number of ones in an individuals bit string. This means that the number of individuals with a certain number of ones will be roughly equal to the number of individuals with any other number of ones. This prevents the algorithm from specializing in a particular area of the search space at the beginning of the algorithm. The evolution ends after D generations and the best individual of the last generation is considered to be the best answer. Note that this does not need to be the case if the fitness function used in the algorithm is probabilistic. In this experiment $D = 100$.

Table 1. This table shows the physical layout of the four different neighborhoods used in this experiment. The maximum and average distance (as described in Section 5) are calculated on a CA with 149 cells.

Name	Physical distance	Max. distance	Avg. distance
Normal	-3 -2 -1 0 1 2 3	25	12.79
Exponents of 2	-4 -2 -1 0 1 2 3	19	9.97
Exponents of 3	-9 -3 -1 0 1 3 9	10	5.48
Exponents of 5	-25 -5 -1 0 1 5 25	6	3.84

For the fitness evaluation a one dimensional cellular automata is used with a width of 149 cells. Every cell has 7 cells in the neighborhood, but the shape of this neighborhood will be different for every test. The transition rule will therefore consist of $2^7 = 128$ bits and there will be $2^{128} \approx 3.4 \cdot 10^{38}$ different rule each with a different behavior. Every generation 100 initial states will be generated that are used to calculate the fitness. The percentage that a rule classifies correctly of these 100 initial states is the fitness of that rule. A state is correctly classified if the iteration of the CA stops changing before 320 steps in an "all ones state" or an "all zeros state" corresponding to $\lambda > 0.5$ and $\lambda < 0.5$ (see Section 3).

7 Experimental Results

In this experiment four different neighborhoods will be tested. The first one will be the standard one dimensional neighborhood including cells 1, 2 and 3 on both sides of the center cell. The other three are chosen to minimize the distance between cells in the neighborhood and include the cells with a physical distance of an integer to the power of 0, 1 and 2. The three integers chosen for this are 2, 3 and 5. Table 1 gives an overview of the layout of the four neighborhoods.

For every neighborhood 100 runs are calculated with the GA as described in Section 6. Unsurprisingly the results of the normal neighborhood matched that of results in [4] and are only a little bit better than the results in [14,15]. The rest of the results are a lot less trivial though. The neighborhood with "exponent of 2" is performing slightly better than the normal neighborhood, whereas "exponents of 3" performs a lot better with more then half of the rule topping 0.7 and about 10% over 0.75. And "exponents of 5" is even better than all the rest with all the rules found being above 0.7 and some even topping 0.8 with the best at 0.813 coming very close to the best rule found on Majority Problem. Furthermore the average number of iterations that a rule needs to classify an initial state gets smaller the higher you make the exponent. Figure 4 shows the best rule from all the runs.

These results support the claim that the shape of the neighborhood is very important for the performance of the CA and that decreasing the distance between cells in the CA increases the performance and decreases the duration.

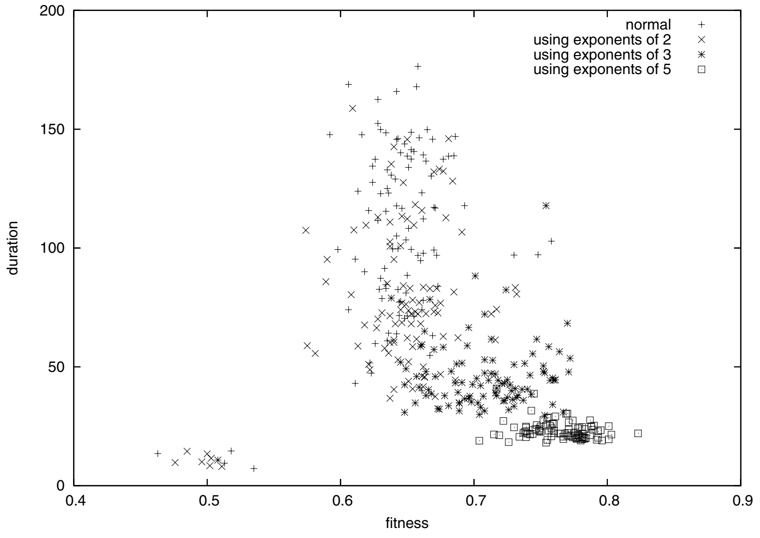


Fig. 4. This figure shows all the rules found with the four different neighborhoods. Note how the fitness goes up and the duration goes down if the neighborhood is wider. The fitness is calculated using $F_{149,10^3}$.

Further research seems needed to determine whether this is the reason that multi dimensional CA have a better performance and a smaller duration [4], but the results suggest that changing the neighborhood or topology of the CA can change the behavior in a very drastic way. Taking this into account it does not seem very surprising that multi dimensional CA have a very different performance than one dimensional CA. It seems intuitive that the way information travels through a CA is an important factor in this and that therefore the metrics of distance (as described in Section 5) seem a very nice way to determine what neighborhood will have a high fitness and a low duration.

8 Conclusions

This article gives an example of how a genetic algorithm can be used to program an algorithm (in this case the cellular automata) and improve the results. Although there are probably better rules than the ones found in this experiment, the search space of transition rules for these cellular automata is so big that improving results is already a big achievement.

It also shows how using a GA it can be concluded that the shape of the neighborhood of a CA can have a big effect on the performance. Although the distance between the cells in a neighborhood is not proven to be the biggest factor in this, it does seem to be an indicator that can be used to devise the right shape of neighborhood.

References

1. Th. Bäck, D. B. Fogel, and editors Michalewicz, Z., editors. *Handbook of Evolutionary Computation.* Oxford University Press and Institute of Physics Publishing, Bristol/New York, 1997.
2. Th. Bäck, Breukelaar R., and Willmes L. Inverse design of cellular automata by genetic algorithms: an unconventional programming paradigm. *UPP proceedings in the 'Hot Topics' subline of LNCS*, 2005.
3. R. Breukelaar and Th. Bäck. Evolving transition rules for multi dimensional cellular automata. In *6th International Conference on Cellular Automata for Research and Industry, ACRI*, Amsterdam, The Netherlands, 2004. Springer.
4. R. Breukelaar and Th. Bäck. Using a genetic algorithm to evolve behavior in multi dimensional cellular automata. In *(to be published) GECCO proceedings*, 2005.
5. A. David, B. Forest, and H. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. 1996.
6. D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, New York, 1995.
7. L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley and Sons, 1966.
8. P. Gacs, G. L. Kurdyumov, and L. A. Levin. One dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii*, 1978.
9. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.
10. D. E. Goldberg. *The Design of Invocation:Lessons from and for Competent Genetic Algorithms.* Kluwer Academic Publishers, 2002.
11. J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.
12. J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT Press, Cambridge, MA, 1992.
13. J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, 2003.
14. M. Mitchell and J.P. Crutchfield. The evolution of emergent computation. Technical report, Proceedings of the National Academy of Sciences, SFI Technical Report 94-03-012, 1994.
15. M. Mitchell, J.P. Crutchfield, and P.T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.
16. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Fromman-Holzboog Verlag, Stuttgart, 1973.
17. I. Rechenberg. *Evolutionsstrategie '94.* Fromman-Holzboog Verlag, Stuttgart, 1994.
18. H. P. Schwefel. Numerische optimierung von computer-modellen mittels der evolutionsstrategie. *Interdisciplinary Systems Research*, 26, 1977.
19. H. P. Schwefel. *Evolution and Optimum Seeking.* Wiley, New York, 1995.
20. S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55, 1983.

Quantum Searching Amidst Uncertainty

Lov K. Grover[1]

Bell Laboratories, Lucent Technologies, 600-700 Mountain Avenue,
Murray Hill, NJ 07974, USA
lkgrover@bell-labs.com

Abstract. Consider a database most of whose entries are marked but the precise fraction of marked entries is not known. What is known is that the fraction of marked entries is $1-\epsilon$, where ϵ is a random variable that is uniformly distributed in the range $(0,\epsilon_0)$ .The problem is to try to select a marked item from the database in a single query. If the algorithm selects a marked item, it succeeds, else if it selects an unmarked item, it makes an error.

How low can we make the probability of error? The best possible classical algorithm can lower the probability of error to $O\left(\epsilon_0^2\right)$. The best known quantum algorithms for this problem could also only lower the probability of error to $O\left(\epsilon_0^2\right)$. Using a recently invented quantum search technique, this paper gives an algorithm that reduces the probability of error to $O\left(\epsilon_0^3\right)$. The algorithm is asymptotically optimal.

1 Introduction

Classical search algorithms are robust. If we reduce the problem size, the algorithm has fewer items to search and the performance of the algorithm will almost always improve. Quantum search algorithms depend on delicate interference effects, any change in parameters leads to significantly different results. For example, if we reduce the number of states in the database by a factor of four, a quantum search algorithm that would have previously succeeded, will now fail with certainty. Quantum algorithms usually exhibit oscillatory behavior in their performance characteristics. As a result of this, if there is some quantity that we want to maximize, e.g. the probability of the system being in marked states, we will need very precise knowledge of the problem parameters. Given such knowledge, it is easy to fine-tune the algorithm so that it achieves a probability of success of unity. What happens if we do not have this knowledge? This can be a serious handicap for a quantum search algorithm. This paper describes a way around this problem.

The original quantum search algorithm [1] considered the problem of finding a marked item in a large unsorted database with minimum queries to the database. For this type of problem it is usually enough to be able to obtain the correct answer with a constant probability since the procedure can be repeated a logarithmic number of times to drive the probability exponentially close to unity.

[1] Research was partly supported by NSA & ARO under contract DAAG55-98-C-0040.

In that problem a logarithmic factor was not significant since the quantum search algorithm gave a square-root asymptotic improvement. However there are other important problems where the additional queries create a significant overhead and need to be minimized, e.g. when we are limited to a single query and have to find the answer with a probability approaching unity. One field in which this type of problem occurs is in pattern recognition and image analysis where each query requires a lot of signal processing and the consequences of making an error are catastrophic.

2 The Problem

Consider the situation where a large fraction of the items in a database are marked, but the precise fraction of marked items is not known. The goal is to find a single marked item with as high a probability as possible in a single query to the database. For concreteness, say some unknown fraction $(1-\epsilon)$ of the items are marked, with ϵ uniformly distributed in the range $(0,\epsilon_0)$ with equal probability. The search algorithm returns an item, if it is a marked item the algorithm is said to succeed, else if it is unmarked, the algorithm is said to fail.

Classically the smallest error that can be obtained is $O\left(\epsilon_0^2\right)$. To achieve this, uniformly guess an input and evaluate the function. If the function outputs 1, you're done, otherwise guess again and cross your fingers (because you have no more evaluations left). One cannot do better than $O\left(\epsilon_0^2\right)$ classically. In this paper we show that the probability of failure for the new scheme is $O\left(\epsilon_0^3\right)$, whereas that of the best (possible) classical scheme and that of the best known quantum schemes are both $O\left(\epsilon_0^2\right)$.

Before considering the specific problem mentioned above, let us describe a general framework. Consider the following transformation

$$UR_sU^\dagger R_tU\left|s\right\rangle \tag{1}$$

$$R_s = I-\left(1-\exp\left(i\frac{\pi}{3}\right)\right)\left|s\right\rangle\left\langle s\right|, \quad R_t = I-\left(1-\exp\left(i\frac{\pi}{3}\right)\right)\left|t\right\rangle\left\langle t\right|$$

U is an arbitrary unitary transformation, R_t & R_s denote selective phase shifts of the respective state(s) by $\frac{\pi}{3}$. Note that if we were to change these phase shifts from $\frac{\pi}{3}$ to π, we would get one iteration of the amplitude amplification algorithm [2], [3].

The next section shows that if U drives the state vector from a source (s) to a target (t) state with a probability of $(1-\epsilon)$, i.e. $\left\|U_{ts}\right\|^2=(1-\epsilon)$, then the transformation (1) drives the state vector from the source to the same target state with a probability of $\left(1-\epsilon^3\right)$. The deviation from the t state has hence fallen from ϵ to ϵ^3. Note that this is different from the amplitude amplification framework where the *amplitudes* were getting amplified; over here it is more convenient to present the results in terms of the *probabilities*.

The striking aspect of this result is that it holds for *any* kind of deviation from the t state. Unlike the standard search (or amplitude amplification) algorithm which would greatly overshoot the target state when ϵ is small (Figure 1); the new algorithm will always move towards the target. As shown in Section 5, this feature of the framework can be used to develop algorithms that are more robust to variations in the problem parameters.

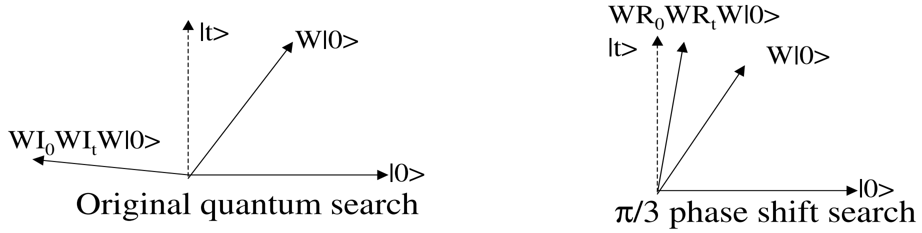


Fig. 1. The original quantum search algorithm is very sensitive to the fraction of marked states. For example, when the fraction is about 3/4, the algorithm fails, the algorithm of this paper will always produce an improvement.

3 Analysis

We analyze the effect of the transformation $UR_sU^\dagger R_tU$ when it is applied to the $|s\rangle$ state. As mentioned in the previous section, R_t & R_s denote selective phase shifts of the respective state(s) by $\frac{\pi}{3}$ (t for target, s for source). We show that if $\|U_{ts}\|^2 = (1-\epsilon)$, then

$$\left\| \langle t| \, UR_sU^\dagger R_tU \, |s\rangle \right\|^2 = \left(1-\epsilon^3\right). \tag{2}$$

In the rest of this section, the greek alphabet θ will be used to denote $\frac{\pi}{3}$. Start with $|s\rangle$ and apply the operations $U, R_s, U^\dagger, R_t$ & U. If we analyze the effect of the operations, one by one, just as in the original quantum search algorithm [1], we find that it leads to the following superposition (this calculation is carried out in the appendix):

$$U|s\rangle \left(e^{i\theta} + \|U_{ts}\|^2 \left(e^{i\theta}-1\right)^2\right) + |t\rangle \, U_{ts} \left(e^{i\theta}-1\right).$$

To calculate the deviation of this superposition from $|t\rangle$, consider the amplitude of the above superposition in non-target states. The probability is given by the absolute square of the corresponding amplitude:

$$\left(1-\|U_{ts}\|^2\right) \left\| \left(e^{i\theta} + \|U_{ts}\|^2 \left(e^{i\theta}-1\right)^2\right) \right\|^2.$$

Substituting $\|U_{ts}\|^2 = (1-\epsilon)$, the above quantity becomes:

$$\begin{aligned} &\epsilon \left\| \left(e^{i\theta} + (1-\epsilon)\left(e^{i\theta}-1\right)^2\right) \right\|^2 \\ &= \epsilon \left\| \left(-e^{i\theta} + e^{2i\theta} + 1\right) - \epsilon \left(e^{i\theta}-1\right)^2 \right\|^2 \\ &= \epsilon^3. \end{aligned}$$

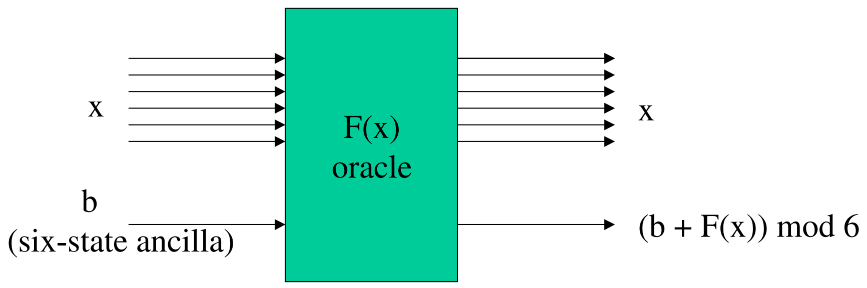


Fig. 2. By setting the six-state ancilla, b, to the superposition $\frac{1}{\sqrt{6}}\left(|0\rangle + |1\rangle\,\omega + |2\rangle\,\omega^2 + |3\rangle\,\omega^3 + |4\rangle\,\omega^4 + |5\rangle\,\omega^5\right)$ where $\omega = \exp\left(-\frac{i\pi}{3}\right)$, we get a $\frac{\pi}{3}$ phase-shift of the states for which $F(x) = 1$ relative to those for which $F(x) = 0$. [10] gives a technique for implementing this with qubits.

4 Existing Algorithms

4.1 Classical Algorithm

The best classical algorithm is to select a random state and see if it is a t state (one query). If yes, return this state; if not, pick another random state and return that without any querying. Note that this algorithm requires a single query, not two. The probability of failure is equal to that of not getting a single t state in two random picks since if either of the two states is a marked state, the algorithm will succeed. This probability is equal to ϵ^2. Since ϵ is uniformly distributed in the range $(0, \epsilon_0)$. The overall failure probability becomes $\frac{\int_0^{\epsilon_0} \epsilon^2 d\epsilon}{\int_0^{\epsilon_0} d\epsilon} = \frac{1}{3}\epsilon_0^2$.

4.2 Quantum Searching

Boyer, Brassard, Høyer and Tapp [4] first described in detail an algorithm that succeeds with probability approaching 1, regardless of the number of solutions (it's a classical algorithm that uses quantum searching as a subroutine; of course, it can be made fully quantum.) The first quantum algorithm to be able to search in a single query with a success probability approaching 1 was given by Mosca [5].

Mosca observed that the quantum counting algorithm of [4] (based on the original searching algorithm) produces a solution with probability converging to 1/2. One easily converts this to an algorithm with probability of success converging to 1/4. Thus by using this algorithm as a sub-routine in another quantum search, we get success probability converging to 1. (This appears to be based on the observation in [4] that an algorithm that succeeds with probability exactly 1/4 can be amplified to one with success probability exactly 1 using only one quantum search iterate). In other words, the technique Mosca uses is to take a search algorithm that succeeds with probability $1/4 - X$ and then use one quantum search iteration to map it to an algorithm that succeeds with probability $\left(1-12X^2-16X^3\right)$. Using this scheme, if the fraction of marked states of the database is $1-\epsilon$, one can easily obtain a marked state with a probability of error of $1-\frac{3}{4}\epsilon^2-\frac{1}{4}\epsilon^3$ by means of a single quantum query. The overall failure probability in this case becomes $\frac{\int_0^{\epsilon_0}\left(\frac{3}{4}\epsilon^2+\frac{1}{4}\epsilon^3\right)d\epsilon}{\int_0^{\epsilon_0}d\epsilon}=\frac{1}{4}\epsilon_0^2+\frac{1}{16}\epsilon_0^3$.

A recent quantum search based algorithm for this problem is by Younes et al [6] (actually it is somewhat unfair to compare it to the other algorithms since this was specifically designed to perform well when the success probability was close to $\frac{1}{2}$, not close to 1). This finds a solution with a probability of $(1-\cos\theta)\left(\frac{\sin^2(q+1)\theta}{\sin^2\theta}+\frac{\sin^2 q\theta}{\sin^2\theta}\right)$, where q = number of queries and $\theta=\arccos\epsilon$ ((59) from [6]). When $q=1$, the success probability becomes: $(1-\epsilon)\left(1+4\epsilon^2\right)$, hence the probability of error becomes $\epsilon-4\epsilon^2+4\epsilon^3$. The overall failure probability becomes $\frac{\int_0^{\epsilon_0}\left(\epsilon-4\epsilon^2+4\epsilon^3\right)d\epsilon}{\int_0^{\epsilon_0}d\epsilon}=\left(\frac{1}{2}\epsilon_0-\frac{4}{3}\epsilon_0^2+\epsilon_0^3\right)$.

5 New Algorithm

As in the quantum search algorithm, encode the N items in the database in terms of $\log_2 N$ qubits. The algorithm consists of applying the transformation $WR_{\overline{0}}WR_tW$ to $\left|\overline{0}\right\rangle$. W is the Walsh-Hadamard Transformation and $\overline{0}$ is the state with all qubits in the 0 state. After this an observation is made which makes the system collapse into a basis state.

In order to analyze the performance of this algorithm, note that the algorithm is merely the phase shift transformation $UR_sU^\dagger R_tU$ applied to $|s\rangle$ which has already been analyzed in section 3. U is the W-H transform (W) and the state s is the $\overline{0}$ state (state with all qubits in the 0 state), then $\left\|U_{ts}\right\|^2=1-\epsilon$, where ϵ lies in the range $(0,\epsilon_0)$. Therefore after applying the transformation $WR_{\overline{0}}WR_tW$ to $\left|\overline{0}\right\rangle$, the probability of being in a non-t state becomes ϵ^3, i.e. the overall failure probability becomes $\frac{\int_0^{\epsilon_0}\epsilon^3 d\epsilon}{\int_0^{\epsilon_0}d\epsilon}=\frac{1}{4}\epsilon_0^3$.

The performance of the algorithm is graphically illustrated in Figure 3.

6 Extensions

6.1 Multi-query Searching

In practice, a database search would use multiple queries. The technique discussed above extends neatly to the multi-query situation. As described in [7], the multi-query algorithm based on the $\frac{\pi}{3}$ phase-shift transformation is able to reduce the probability of error to $\frac{\epsilon_0^{2q+1}}{2q+2}$ after q queries to the database. A classical algorithm reduces the probability of error to $\frac{\epsilon_0^{q+1}}{q+2}$. Note that $q=1$ gives the same results as in section 5.

6.2 Optimality

Both the single query, as well as the multi-query algorithms are asymptotically optimal in the limit of small ϵ_0. This is separately proved in [9].

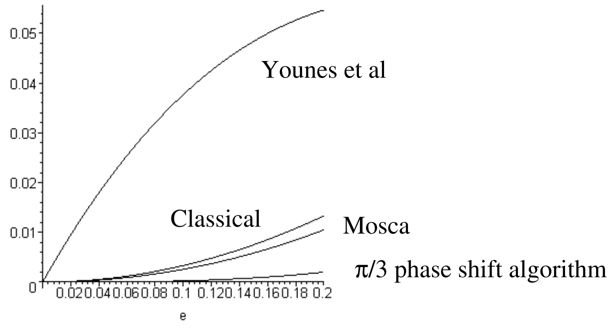


Fig. 3. Comparison of the failure probability of the $\pi/3$ phase shift algorithm with a classical algorithm, [6], and [5], when the fraction of unmarked states ϵ, varies between 0 & 0.2

6.3 Quantum Control and Error Correction

Connections to control and error correction might be evident. Let us say that we are trying to drive a system from an s to a t state/subspace. The transformation that we have available for this is U which drives it from s to t with a probability $\|U_{ts}\|^2$ of $(1-\epsilon)$, i.e. the probability of error in this transformation is ϵ. Then the composite transformation $UR_sU^\dagger R_tU$ will reduce the error to ϵ^3.

This technique is applicable whenever the transformations U, $U^\dagger, R_s$ & R_t can be implemented. This will be the case when errors are systematic errors or slowly varying errors, e.g. due to environmental degradation of some component. This would not apply to errors that come about as a result of sudden disturbances from the environment. It is further assumed that the transformation U can be inverted with exactly the same error. Traditionally quantum error correction is carried out at the single qubit level where individual errors are corrected, each error being corrected in a separate way. With the machinery of this paper, errors can be corrected without ever needing to identify the error syndrome. This is discussed in [7] and [8].

References

1. L. K. Grover, "Quantum Mechanics helps in searching for a needle in a haystack", *Phys. Rev. Letters*, 78(2), 325, 1997.
2. G. Brassard and P. Hoyer, "An exact quantum polynomial-time algorithm for Simon's problem", Proceedings of Fifth Israeli Symposium on Theory of Computing and Systems (ISTCS'97), Ramat-Gan, Israel, June 1997, 12–23, quant-ph/9704027.
3. L. K. Grover, "Quantum computers can search rapidly by using almost any transformation", Phys. Rev. Letters, 80(19), 1998, 4329-4332.
4. Boyer et al, quant-ph/9605034, PhysComp 96, and Fortsch.Phys. 46 (1998) 493-506.

5. M. Mosca, Theoretical Computer Science, 264 (2001) pages 139-153.
6. Ahmed Younes, Jon Rowe, Julian Miller, "Quantum Search Algorithm with more Reliable Behavior using Partial Diffusion," quant-ph/0312022.
7. Lov K. Grover, "A different kind of quantum search", quant-ph/0503205.
8. B. Reichardt & L.K. Grover, "Quantum error correction of systematic errors using a quantum search framework," quant-ph/0506242 .
9. Sourav Chakraborty, Jaikumar Radhakrishnan, Nanda Raghunathan, "The optimality of Grover's recent quantum search algorithm," manuscript in preparation.
10. Tathagat Tulsi, Lov Grover, Apoorva Patel, "A new algorithm for directed quantum search," quant-ph/0505007.

Appendix

We analyze the effect of the transformation (3) of section 3 when it is applied to the $|s\rangle$ state. As in section 3, θ will denote $\frac{\pi}{3}$. We show that if $\|U_{ts}\|^2 = (1-\epsilon)$, then

$$\left\| \langle t| U R_s U^{\dagger} R_t U |s\rangle \right\|^2 = \left(1-\epsilon^3\right) \tag{3}$$

starting state $|s\rangle$
after R_tU

$$R_tU|s\rangle = U_{ts}\left(e^{i\theta}-1\right)|t\rangle + U|s\rangle$$

after $U^{\dagger}$:

$$U^{\dagger} R_tU|s\rangle = |s\rangle + U_{ts}\left(e^{i\theta}-1\right)U^{\dagger}|t\rangle$$

after R_s :

$$\begin{aligned} R_sU^{\dagger}R_tU|s\rangle &= \left(|s\rangle + U_{ts}\left(e^{i\theta}-1\right)U^{\dagger}|t\rangle\right) \\ &+ \left(e^{i\theta}-1\right)\left(|s\rangle + U_{ts}\left(e^{i\theta}-1\right)|s\rangle\langle s|U^{\dagger}|t\rangle\right) \\ &= |s\rangle\left(e^{i\theta} + \|U_{ts}\|^2\left(e^{i\theta}-1\right)^2\right) + U_{ts}\left(e^{i\theta}-1\right)U^{\dagger}|t\rangle \end{aligned}$$

after the final U**:**

$$UR_sU^{\dagger}R_tU|s\rangle = U|s\rangle\left(e^{i\theta} + \|U_{ts}\|^2\left(e^{i\theta}-1\right)^2\right) + U_{ts}\left(e^{i\theta}-1\right)|t\rangle$$

Calculating the error. To estimate the error, consider the amplitude of the above superposition in non-target states (this is due to the portion

$$U|s\rangle\left(e^{i\theta} + \|U_{ts}\|^2\left(e^{i\theta}-1\right)^2\right),$$

the other portion $U_{ts}\left(e^{i\theta}-1\right)|t\rangle$ is clearly in the target state). The magnitude of the error probability is given by the absolute square of this error amplitude:

$$\left(1-\|U_{ts}\|^2\right)\left\|\left(e^{i\theta} + \|U_{ts}\|^2\left(e^{i\theta}-1\right)^2\right)\right\|^2$$

Assume $\|U_{ts}\|^2$ to be $(1-\epsilon)$, the above quantity becomes:

$$\begin{aligned}
&\epsilon \left\| \left(e^{i\theta} + (1-\epsilon)\left(e^{i\theta}-1\right)^2 \right) \right\|^2 \\
&= \epsilon \left\| \left(-e^{i\theta} + e^{2i\theta} + 1\right) - \epsilon \left(e^{i\theta}-1\right)^2 \right\|^2 \\
&= \epsilon^3
\end{aligned}$$

Logic Functions of the Genomic Cis-regulatory Code

Eric Davidson[1] and Sorin Istrail[2]

[1] Division of Biology, California Institute of Technology, USA
[2] Center for Computational Molecular Biology and Computer Science Department, Brown University, USA
sorin@cs.brown.edu

Cis-regulatory modules that control developmental gene expression process the regulatory inputs provided by the transcription factors for which they contain specific target sites. A prominent class of cis-regulatory processing functions can be modeled as logic operations. Many of these are combinatorial because they are mediated by multiple sites, although others are unitary. In this work, we illustrate the repertoire of cis-regulatory logic operations, as an approach toward a functional interpretation of the genomic regulatory code.

Structural DNA Nanotechnology: Molecular Construction and Computation

Ruojie Sha, Xiaoping Zhang, Shiping Liao, Pamela E. Constantinou, Baoquan Ding, Tong Wang, Alejandra V. Garibotti, Hong Zhong, Lisa B. Israel, Xing Wang, Gang Wu, Banani Chakraborty, Junghuei Chen, Yuwen Zhang, Hao Yan, Zhiyong Shen, Wanqiu Shen, Phiset Sa-Ardyen, Jens Kopatsch, Jiwen Zheng, Philip S. Lukeman, William B. Sherman, Chengde Mao[1], Natasha Jonoska[2], and Nadrian C. Seeman

Department of Chemistry, New York University, New York, NY 10003, USA
ned.seeman@nyu.edu
[2] Department of Mathematics, University of South Florida, Tampa, FL 33620, USA
jonoska@math.usf.edu
[1] Department of Chemistry, Purdue University, West Lafayette, IN 47907, USA
mao@purdue.edu

Abstract. Structural DNA nanotechnology entails the construction of objects, lattices and devices from branched DNA molecules. Branched DNA molecules open the way for the construction of a variety of N-connected motifs. These motifs can be joined by cohesive interactions to produce larger constructs in a bottom-up approach to nanoconstruction. The first objects produced by this approach were stick polyhedra and topological targets, such as knots and Borromean rings. These were followed by periodic arrays with programmable patterns. It is possible to exploit DNA structural transitions and sequence-specific binding to produce a variety of DNA nanomechanical devices, which include a bipedal walker and a machine that emulates the translational capabilities of the ribosome. Much of the promise of this methodology involves the use of DNA to scaffold other materials, such as biological macromolecules, nanoelectronic components, and polymers. These systems are designed to lead to improvements in crystallography, computation and the production of diverse and exotic materials. Branched DNA can be used to emulate Wang tiles, and it can be used to construct arbitrary irregular graphs and to address their colorability.

Keywords: Unusual DNA Motifs, Bottom-Up Nanoscale Construction, DNA Sequence Design, Nanoscale DNA Objects, Nanorobotics, Nanoscale Pattern Design.

1 Introduction

The double helical structure of DNA is well-known. It is the repository of genetic information for all organisms. It is able to function in this role because of the complementarity nature of the pairing between the bases: Adenine (A) pairs specifically with thymine (T) and guanine (G) pairs specifically with cytosine (C). The biological consequence of complementarity is that the cellular machinery can replicate the information contained in each strand *via* a semi-conservative mechanism, leading to the production of daughter cells containing the same genetic complement. Pairing between short intermolecular overhangs, called 'sticky ends' has been used by genetic engineers for over 30 years to organize DNA fragments [1]. This is a unique intermolecular interaction in chemistry, not only because it is programmable (the base sequences of sticky ends can be changed to a wide variety of different complementary pairs), but because the local product structure is known to be similar to the classic B-DNA structure that is seen everywhere as a cultural icon throughout the world [2].

Of course, from the perspective of nanotechnology, genetic engineering does not provide the structural consequences that one would want from a bottom-up approach to the control of the structure of matter. Although sticky-ended cohesion can be used to order a series of DNA fragments, say on a plasmid, the end product is simply a double helical molecule that flanks a linear helix axis, although that helix may be cyclic, knotted or catenated. The missing element to make DNA an interesting nanotechnological building block is a branch point. Branches enable the construction of N-connected objects and lattices. They are also inherent to most applications of DNA to nanomechanical devices. Sticky ends and their application to forming N-connected DNA objects are illustrated in Figure 1.

In this article, we describe how to design branched DNA motifs and how to select sequences for those motifs. We then describe some of the key steps taken by the field of structural DNA nanotechnology to reach the point where today it is a discipline practiced by a significant number of investigators. We describe the construction of the earliest N-connected objects, and of periodic arrays with programmable patterns. We introduce the way in which structural DNA nanotechnology interfaces with computation. We also describe some of the most exciting developments in the area of nanomechanical devices, leading to the development of DNA-based nanorobotics.

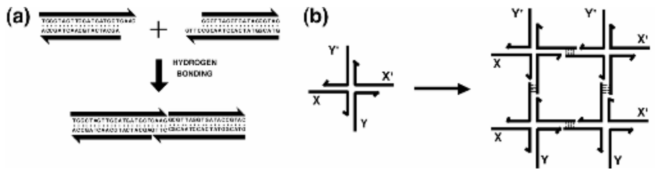


Fig. 1. *Basics of Structural DNA Nanotechnology.* (a) Sticky-ended cohesion. (b) Self-assembly of branched DNA molecules with complementary sticky end pairs.

2 Motif and Sequence Design

The essential difference between conventional linear DNA and the motifs useful in structural DNA nanotechnology is the presence of branching. This notion is thought of most easily as a crossover or strand exchange between two adjacent double helices. Reciprocal exchange is the protocol that can be used to produce branched motifs.

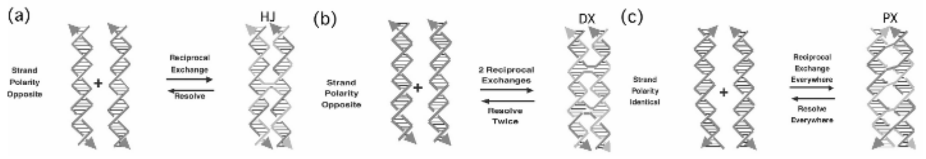


Fig. 2. *Reciprocal Exchange leads to branched structures.* (a) A single exchange leads to a Holliday junction. (b) Two exchanges lead to the DX molecule. (c) Exchange at every possible position generates the PX molecule. Note that the exchanges in (b) are between strands of the opposite polarity, but those in (c) are between strands of the same polarity. The exchange in (a) is shown to be between strands of opposite polarity, but the polarity of the strands does not matter in this case.

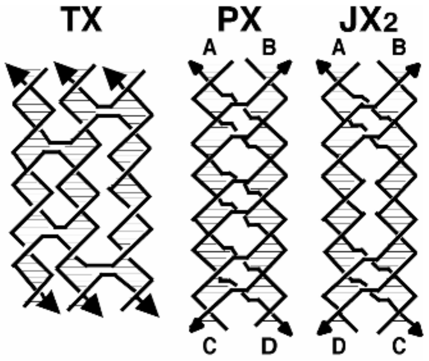


Fig. 3. *Branched DNA Motifs.* On the left is a TX molecule generated by adding a helical domain to a DX by two reciprocal exchanges between opposite polarity stands. The JX_2 molecule on the right can be compared readily to its topoisomer, the PX molecule. It contains the same arrangement of strands, but it lacks two crossovers, leading to a molecule in which the top end is twisted relative to the bottom by a half-turn less than in the PX. This feature is highlighted by the alphabetic labels on the ends of the double helices.

Figure 2 illustrates how this method is used to produce a singly-branched Holliday junction (an intermediate in genetic recombination), a stiff double crossover (DX) molecule and a paranemic crossover (PX) molecule. A number of other useful motifs generated in this way are illustrated in Figure 3.

These include the TX molecule and the JX_2 molecule, a topoisomer of the PX molecule whose relationships are illustrated by the letters flanking their helices. All of these motifs have found application in structural DNA nanotechnology. As noted in Figure 2, it is clear that forming each motif entails fusing DNA double helices. The two strands of the conventional double helical DNA molecule run in opposite directions. Thus, fusion between double helices can occur between strands of the same polarity or of opposite polarity. For the Holliday junction, with a single crossover, this difference has no topological consequences. However, DX molecules can be made with either polarity in its linkages, and the TX molecule can be made with mixed polarities. The DX and TX molecules shown in Figures 2 and 3 have opposite polarities. The PX molecule, and its topoisomer, JX_2, have crossovers between strands of the same polarity[3].

The assignment of sequences to these motifs seeks to maximize the probability that the target structure will form at equilibrium as a consequence of mixing the strands and cooling them. This is done by using the assumption that Watson-Crick base pairing is the preferred form of interaction between strands of DNA. Designing a crossover into the structure introduces a fault into this structure, corresponding to an excited state. The key point is to make sure that the excited state that results is the one sought, rather than one that is easier for the system to accommodate. To ensure that the target is what results, sequence symmetry is minimized throughout the structure [4]. An example of how this is done is shown in Figure 4a, which shows a single-crossover Holliday junction-like molecule in a crossroads representation. The strands of the target molecule shown contain sixteen nucleotides. Each strand has been broken up into a series of thirteen overlapping tetramers, such as the CGCA or GCAA that are boxed at the top of strand 1. Sequence symmetry is minimized by insisting that each tetramer be unique. Thus, competition with the target octamers in each arm can come only from trimers, such as the ATG sequences boxed on strands 2 and 3; in principle, they could go to the wrong places, but both the thermodynamics of pairing and the cooperativity of double helix formation work against this eventuality. In addition, an element of negative control is introduced by forbidding the complement of any tetramer that flanks a branch point, such as the CTGA that is boxed at the corner of strand 1; the absence of a TCAG anywhere in the sequence means that the sequence designed to go around the corner does not have the opportunity to form double helical DNA.

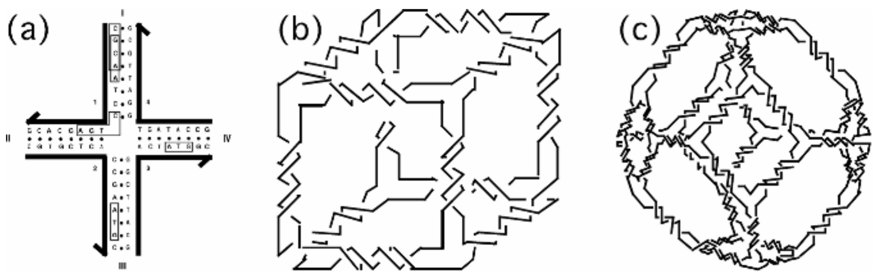


Fig. 4. *Design of Sequences for Branched Junctions and Polyhedral Catenanes.* (a) Design of a branched junction. (b) DNA cube and (c) a DNA truncated octahedron.

3 Structural and Topological Constructions

Given our understanding of branched motif generation and sequence assignment, it is easy to see how DNA can be used to prepare topological and structural targets. Following the notion summarized in Figure 1b, it has been possible to construct DNA polyhedra, molecules whose edges consist of double helical DNA and whose vertices correspond to the branch points of DNA branched junctions. We have reported a cube [5] and a truncated octahedron [6] built from DNA. Schematic versions of these constructs are shown in Figures 4b and 4c. One can think of these polyhedra simply as 3-connected graphs; we have also reported the construction of an irregular graph containing both 3-connected and 4-connected vertices (see below). The connectivity of targets is limited by the number of double helical arms that flank the junctions in the structure. We have reported junctions flanked by up to six arms [7], and actually have made junctions with as many as twelve arms (XW and NCS, in preparation). There are few interesting individual targets with large connectivities, but lattices built from Platonic and Archimedean solids can have connectivities this high. For example, the stick version of cubic close packing is 12-connected. It is also noteworthy that the plectonemic nature of the DNA double helix makes it an extremely convenient synthon for topological targets; this is true because a half-turn of DNA corresponds to a unit-tangle in a knotted or catenated structure [8]. In this context, the cube is a hexacatenane, and the truncated octahedron is a 14-catenane. In addition, an elusive topo-synthetic target, known as Borromean Rings, was synthesized first from DNA by using this same principle.

The construction of periodic matter is one of the key goals of structural DNA nanotechnology. Periodic matter requires stiff components: The relationship between periodicity and cyclization so prominent in the Fourier analysis of crystals also applies to DNA nanoconstruction; components that can combine to form periodic matter can also form cyclic matter, thereby poisoning the assembly. The best defense against this problem is to use stiff components. Simple branched junctions are somewhat floppy, so they did not seem to be the best molecules to use to construct periodic matter. We found that DX molecules and their relatives, such as TX molecules are very good for this purpose. We have constructed arrays with patterns that can be both programmed and modified from DX molecules [9,10]. We have also built 2D arrays from parallelograms [11], and from triangular arrangements of DX molecules [12]. An example of the latter is shown in Figure 5.

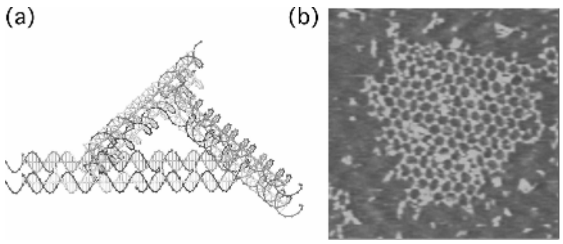


Fig. 5. *Formation of a Trigonal DNA Array.* (a) The robust DX triangle motif. (b) An AFM image of a trigonal array built from two different molecules of this motif.

4 Applications of Structural DNA Nanotechnology in Molecular Computation

Experimental DNA-based computation was founded by Leonard Adleman in 1994 [13]. Adleman combined the information in DNA molecules themselves, using standard biotechnological operations (ligation, PCR, gel electrophoresis and sequence-specific binding) to solve a Hamiltonian path problem. The idea is that there exist computational problems for which the parallelism of molecular assembly overcomes the slow speed of the required macroscopic manipulations. Many varieties of DNA-based computation have been proposed, and a number of them have been executed experimentally for relatively small cases. Space does not permit discussion of all of them.

However, there are two approaches to DNA-based computation that are relevant to our discussion of structural DNA nanotechnology. The first was a method suggested by Winfree, who noticed that the system described above, branched junctions with sticky ends, could be a way to implement computation by 'Wang tiles' on the molecular scale [14]. This is a system of tiles whose edges may contain one or more different markings; the tiles self-assemble into a mosaic according to the local rule that all edges in the mosaic are flanked by the same marking. Such a form of assembly can be shown to emulate the operation of a Turing machine, a general-purpose computer. The relationship between the sticky ends of a branched junction and the markings on a Wang tile is shown in Figure 6a.

This form of DNA-based computation has been prototyped successfully in a 4-bit cumulative XOR calculation [15]. The XOR calculation yields a 1 if the two inputs are different, and a 0 if they are the same. Figure 6b shows the components of this calculation. Each component is a TX molecule, schematized as three rectangles with geometrical shapes on their ends to represent complementarity. The input bits are '**x**' tiles (upper left), and the output bits are '**y**' tiles (bottom), and there are two initiator tiles, **C1** and **C2**, as well (upper right). The upper left corner of Figure 6c shows the strand structure of the TX tiles; each strand contains a 'reporter strand' (drawn with a thicker line); the value of **x** and **y** tiles is set to 0 or 1 depending whether it contains a Pvu II or EcoR V restriction site, respectively. The $\mathbf{y_i}$ tiles perform the gating function; there are four of them, corresponding to the four possible combinations of 0 and 1 inputs. The input involves the bottom domain (Figure 6b). The assembly of periodic arrays discussed above entails competition between correct and incorrect tiles for particular positions; by contrast, the competition here is between correct and partially correct tiles. For example, the $\mathbf{y_{i-1}} = 0$ sticky end on the leftmost tile is the same as the $\mathbf{y_{i-1}}$ sticky end on the rightmost tile. In the cumulative XOR calculation, $\mathbf{y_i} = \mathbf{XOR}\ (\mathbf{x_i},\mathbf{y_{i-1}})$. The implementation of this formula is shown in Figure 6c. The $\mathbf{x_i}$ tiles and the initiators are given longer sticky ends than the $\mathbf{y_i}$ tiles, so they assemble a template first when the tiles are cooled. This creates a double site where the $\mathbf{y_1}$ tile can bind. This binding creates the double site where the $\mathbf{y_2}$ tile can bind, and so on. When the assembly is complete, the reporter strands are ligated together, creating a long strand that connects the input to the output through the initiator tiles. Partial restriction analysis of the resulting strand reveals that the correct answer is obtained almost exclusively.

The second approach is due to Jonoska and colleagues [16], and applies to the problem of coloring graphs. If a graph is 3-colorable, one can color its nodes with three colors, so that in no cases is one edge flanked by the same color. This problem can be converted to terms of structural DNA nanotechnology, if one represents the edges with double helices and the vertices with branched junctions as shown in Figure 7. Each edge has six different representations, corresponding to six different combinations of different colors that can flank it. The colors are coded by sticky

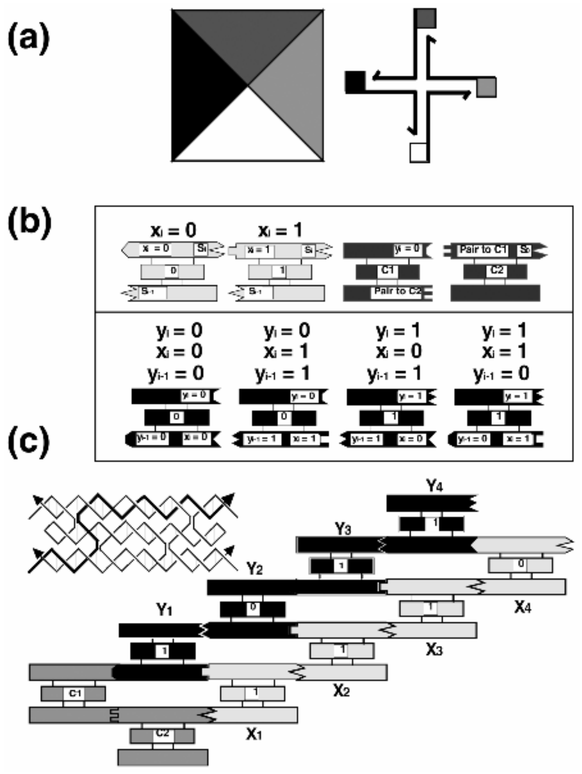


Fig. 6. *A Cumulative XOR Computation.* (a) The Relationship between Wang Tiles and Branched Junctions. The shadings are the same in both the tile and the sticky ends of the junction, indicating that the sticky ends on a branched junction can emulate a Wang tile. (b) The Components of a Cumulative XOR Calculation. TX tiles are shown as rectangles ending in sticky ends represented geometrically. The input **x** tiles are shown at the upper left; and the value of the tile is shown in the central domain. Initiator tiles **C1** and **C2** are shown in the upper right and the four possible **y** tiles are shown in the bottom row. The inputs of the **y** tiles is shown on their bottom domains. (c) The Self-Assembled Tiles. The strand structure of the TX tiles is illustrated on the upper left, with the reporter strand drawn with a thicker line. The assembly of tiles in a prototype calculation is shown, using the components illustrated in (b). The input 1, 1, 1, 0 produces an output of 1, 0, 1, 1 by successive binding of **y** tiles into the double sites created as the array assembles.

ends. Similarly, three different copies of each vertex are present in solution, except for an initiating pair. If a solution to the problem exists, it should be possible to ligate the components to form a closed cyclic DNA molecule. Such a molecule can be detected either by its mobility on a 2D polyacrylamide gel, or by its resistance to exonucleases.

If one is to perform such a calculation, it is necessary to make sure that it is possible to build the graph out of real molecules. The edge of a graph has no thickness, but the edges of a DNA representation of a graph are 2 nm thick. The length of an edge in a graph can be arbitrary, but specific lengths must be assigned to a DNA graph, and the lengths and placements of the edges must be compatible with the twisting and bending stiffness of the DNA double helical components. Thus, we have prototyped a three-colorability calculation in a 'monochromatic' experiment [17]. In addition to the issue of the feasibility of building the molecules with DNA components, this type of assembly also needs to be done all at once in solution. The regular graphs previously built from DNA, the cube [5] and the truncated octahedron [6], we built in specific steps, so that the chemistry was much simpler. Thus, two issues were at stake in this self-assembly. Following the simultaneous ligation of all components, the products were run on a 2D gel, and bands indicating complete formation of a cyclic molecule were found. Restriction analysis demonstrated that all components were present, and were in the right order. About 2% of the products indicated errors.

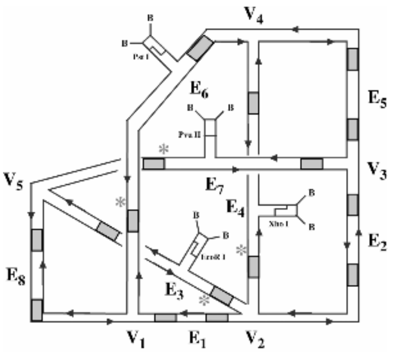


Fig. 7. *An Irregular Graph.* The graph shown has explicit edges (E1-E8) as well as vertices (V1-V5). The sites of the sticky ends are indicated by patches of gray. Restriction sites are indicated by the names of the restriction enzymes, biotin groups are labeled as 'B' and radioactive labeling sites are shown by asterisks. Arrows indicate strand polarities. Note that this is a knot, not a catenane.

5 Nanomechanical Devices

In addition to being an outstanding medium for the construction of objects and arrays, DNA is also convenient for the construction of nanomechanical devices. The first robust device constructed from DNA was based on the B-Z structural transition [18]. Conventional DNA, known as B-DNA, is a right-handed double helical molecule whose repeat unit is the nucleotide pair. However, appropriate sequences, typically $(CG)_n$ can adopt the Z-DNA conformation which is a left-handed double helix whose repeat unit is a dinucleotide pair. In the absence of Z-DNA-promoting conditions, Z-forming sequences remain in the B-DNA conformation. The device (Figure 8a) consists of two DX molecules connected by a shaft that contains a Z-forming sequence. In the absence of the Z-promoting reagent $Co(NH_3)_6^{3+}$, both DX molecules have their extra domain on the same side of the shaft; adding $Co(NH_3)_6^{3+}$ converts the Z-forming sequence to the Z conformation, placing one of the extra domains on the other side of the shaft.

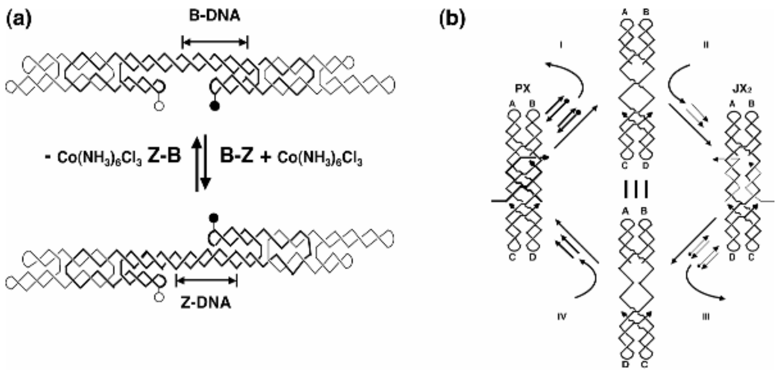


Fig. 8. *DNA Nanomechanical Devices.* (a) A device based on the B-Z transition of DNA. Addition of $Co(NH_3)_6Cl_3$ converts a short segment from B-DNA to Z-DNA. (b) The machine cycle of a PX-JX_2 device. Removal of PX set strands (I) leaves a naked frame to which JX_2 set strands (II) can be added to change state. The PX state can be restored by the same procedures (III) and (IV).

Figure 8b illustrates a robust sequence-dependent device that is capable of using the sequence specificity of DNA, so that a variety of devices can coexist in the same environment, yet be addressed individually [19]. The device can adopt two different states, termed PX and JX_2, that differ from each other by a half-turn. The states are set by DNA strands that are termed 'set strands'. One can make a number of different devices by changing the sequences to which the set strands bind. In the drawing, the set strands for the PX conformation are drawn with thick lines, and the set strands for the JX_2 conformation are drawn with thin lines. The set strands contain single-stranded extensions that are not paired with any other strands. Yurke and his colleagues [20] have shown that when the complete complement to a set strand [termed an 'unset strand'] is added to solution, it will remove the set strand, leaving a naked device frame. Thus, the machine cycle shown in Figure 8b starts on the left with the PX state, moves clockwise through the addition of unset strands to leave a naked frame. Continuing onwards, the set strands for the JX_2 state are then added, and that state is formed. Adding the unset strands for the JX_2 state again produces the naked frame, and the addition of the set strands for the PX state completes the cycle.

Two of these devices have been used to produce a nanomachine capable of translation [21]. The devices are used to couple DNA constructs in this machine, shown in Figure 8a. From left to right, the machine contains a DNA diamond, a PX-JX_2 device, a DNA double diamond, a distinct PX-JX_2 device, and a second DNA double diamond. The numbers represent sticky ends. The gap above the first PX-JX_2 device is flanked by 1 and 2, and the gap above the second PX-JX_2 device is flanked by 4 and 6. If the state of the left PX-JX_2 device is changed from PX to JX_2, and the other PX-JX_2 device left untouched, the first gap will be flanked by 1 and 3, and the second gap will be flanked by 5 and 7. There are four different states to the machine, and each of them places different sticky end pairs on the two ends of the gaps. The solution contains six DX molecules whose bottom domain is complementary to one of these pairs. Once DX molecules are in place above the gaps, they are ligated to each

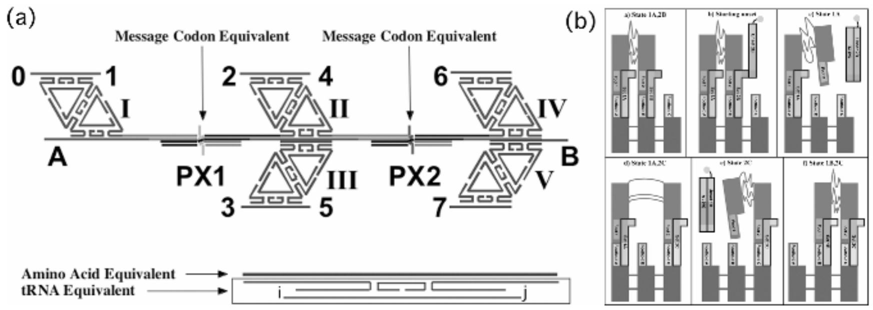


Fig. 9. Advanced Devices Built from DNA. (a) A translation device. The diamond on the left and the double diamonds in the center and on the right are separated by two PX-JX_2 devices. The Arabic numerals indicated sticky ends. The sticky ends at the top will bind a DX molecule complementary to the numbers. As shown, a DX with sticky ends complementary to 1 and 2 and another with sticky ends complementary to 4 and 6 will bind. If the device on the right is switched to the JX_2 state, a DX complementary to 4 and 7 will bind. This arrangement allows for positional ligation of the DX molecules bound there. The set strands correspond to an mRNA codon, and the bottom sticky ends (i and j) of the DX are the equivalent of the anticodon. The top strand of the DX is equivalent to the amino acid of an aminoacyl tRNA. (b) A bipedal walker. The parts of a walk are shown. The unset strand removes the set strand of the right foot, and then another set strand fastens it to a new position on the sidewalk. The unset strand of the left foot then frees it and it is fastened by a new set strand to the position where the right foot was bound.

other and to an initiator DX that binds to site 0. The product is determined by the state of the machine, and there is no transcriptional relationship between the set strands and the produce. The analogy to the components of protein synthesis is indicated in Figure 9a: The DX molecules are analogous to aminoacyl tRNA molecules, their bottom domain to the anticodon, and their top strand to the amino acid. The set strands are analogous to codons.

Another sequence-dependent device is illustrated in Figure 9b. This is a bipedal walker that walks on a sidewalk [22]. It consists of two double helical domains attached to each other by a flexible linker. The bottom end of each domain is a single-stranded segment. Similarly, the top end of each part of the sidewalk (a TX molecule) is single-stranded. The walker is positioned onto the sidewalk by set strands, similar to those in the PX-JX_2 device. Removal of a set strand by adding an unset strand frees a double helical domain of the walker; addition of a set strand for a different position of the sidewalk locks it down to the new position. The drawing shows how the right leg of the walker is freed from the sidewalk, and then it is tied down to a new position. The left leg is then freed from the sidewalk, and is tied down to the position originally occupied by the right leg. Thus, a step has occurred. The device can walk in both directions. By itself this one-dimensional example is only a prototype. However, the same device or a triped can walk in two directions on a two-dimensional lattice. Thus it is programmable in an interesting fashion by the input strands; similarly, it also could be programmed by the output of logical processes, and could be used to keep track of the state to which a system had arrived.

6 Conclusions, Applications and Challenges

We have demonstrated the facility and versatility of DNA as a bottom-up construction medium on the nanoscale. Objects, arrays and devices are all readily within the scope of this technology. Where can this approach be utilized? The system was originally devised to facilitate the crystallization of biological macromolecules. The idea is to use the DNA lattice as a macromolecular-scale host for macromolecular guests; if all components are well ordered, crystallographic analysis can be performed on both the host and the guest. An example is shown in Figure 10a. However, if one can conceive of organizing biological systems, one also can imagine that nanoelectronic components can be organized [23]. Thus, by using the superb architectural properties of DNA described above, we may be able to organize molecular electronics in an efficient self-assembled fashion. This notion is shown in Figure 10b. We expect that the various devices noted above will be of use in producing materials that previously were inaccessible.

Many technical challenges remain to be overcome before all of the potential of this approach can be realized. We must extend our 2D capabilities to 3D, with high order. We must be able to incorporate both devices and hetero-species into these lattices. Both hierarchical and functional lattices remain to be developed, and sophistication of the chemistry must be achieved to make this a biomimetic, rather than a biokleptic system. Nevertheless, we are at the very beginning of structural DNA nanotechnology, and the future appears to be boundless!

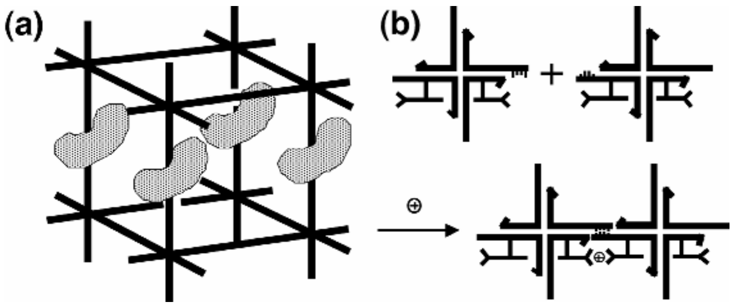


Fig. 10. *Applications of Structural DNA Nanotechnology.* (a) A crystallographic application. A DNA box is shown with sticky ends in each direction. The sticky ends can associate to make a host lattice. Macromolecular guests are shown ordered within the host lattice, enabling their structure determination. (b) Ordering nanoelectronic components. Two branched junctions are shown, and their cohesion organizes molecular wires that are pendent from them.

References

1. S.N. Cohen, A.C.Y., Chang, H.W., Boyer & R.B. Helling, Construction of Biologically Functional Bacterial Plasmids in vitro, Proc. Nat. Acad. Sci. (USA) **70**, 3240-3244 (1973).
2. H. Qiu, J.C. Dewan and N.C. Seeman, A DNA Decamer with a Sticky End: The Crystal Structure of d-CGACGATCGT. J. Mol. Biol. **267**, 881-898 (1997).
3. N.C. Seeman, DNA Nicks and Nodes and Nanotechnology, NanoLetts. **1**, 22-26 (2001).
4. N.C. Seeman, Nucleic Acid Junctions and Lattices. J. Theor. Biol. **99**, 237-247 (1982).

5. J. Chen & N.C. Seeman, The Synthesis from DNA of a Molecule with the Connectivity of a Cube, Nature **350**, 631-633 (1991).
6. Y. Zhang & N.C. Seeman, The Construction of a DNA Truncated Octahedron, J. Am. Chem. Soc. **116**, 1661-1669 (1994).
7. Y. Wang, J.E. Mueller, B. Kemper, & N.C. Seeman, The Assembly and Characterization of 5-Arm and 6-Arm DNA Junctions, Biochem. **30**, 5667-5674 (1991).
8. N.C. Seeman, The Design of Single-Stranded Nucleic Acid Knots, Molec. Eng. **2**, 297-307 (1992).
9. E. Winfree, F. Liu, L. A. Wenzler, & N.C. Seeman, Design and Self-Assembly of Two-Dimensional DNA Crystals, Nature **394**, 539-544 (1998).
10. F. Liu, R. Sha & N.C. Seeman, Modifying the Surface Features of Two-Dimensional DNA Crystals, J. Am. Chem. Soc. **121**, 917-922 (1999).
11. C. Mao, W. Sun & N.C. Seeman, Designed Two-Dimensional DNA Holliday Junction Arrays Visualized by Atomic Force Microscopy, J. Am. Chem. Soc. **121**, 5437-5443 (1999).
12. B. Ding, R. Sha & N.C. Seeman, Pseudohexagonal 2D DNA Crystals from Double Crossover Cohesion, J. Am. Chem. Soc. **126**, 10230-10231 (2004).
13. L. Adleman, Molecular Computation of Solutions to Combinatorial Problems. Science **266**, 1021-1024 (1994).
14. E. Winfree,. In DNA Based Computers, Proceedings of a DIMACS Workshop, April 4, 1995, Princeton University, (eds. Lipton, R.J & Baum, E.B.), Am. Math. Soc., Providence, On the computational power of DNA annealing and ligation. pp. 199-219 (1996).
15. C. Mao, T. LaBean, J.H. Reif & N.C. Seeman, Logical Computation Using Algorithmic Self-Assembly of DNA Triple Crossover Molecules, Nature **407**, 493-496 (2000).
16. N. Jonoska, S.A. Karl & M. Saito, 3D DNA Structures in Computing, Biosystems **52**, 143-153 (1999).
17. P. Sa-Ardyen, N. Jonoska and N.C. Seeman, The Construction of Graphs Whose Edges are DNA Helix Axes, J. Am. Chem. Soc., **126**, 6648-6657 (2004).
18. C. Mao, W. Sun, Z. Shen & N.C. Seeman, A DNA Nanomechanical Device Based on the B-Z Transition, Nature **397**, 144-146 (1999).
19. H. Yan, X. Zhang, Z. Shen & N.C. Seeman, A Robust DNA Mechanical Device Controlled by Hybridization Topology, Nature **415**, 62-65 (2002).
20. B. Yurke, A.J., Turberfield, A.P. Mills, Jr., F.C. Simmel & J.L. Newmann, A DNA-fuelled molecular machine made of DNA, Nature **406**, 605-608 (2000).
21. S. Liao & N.C. Seeman, Translation of DNA Signals into Polymer Assembly Instructions, Science **306**, 2072-2074 (2004).
22. W.B. Sherman & N.C. Seeman, A Precisely Controlled DNA Bipedal Walking Device, NanoLetts. **4**, 1203-1207 (2004).
23. B.H. Robinson & N.C. Seeman, The Design of a Biochip: A Self-Assembling Molecular-Scale Memory Device. Protein Eng. **1**, 295-300 (1987).

Natural Inspiration for Artificial Adaptivity: Some Neurocomputing Experiences in Robotics*

Carme Torras

Institut de Robòtica i Informàtica Industrial (CSIC-UPC),
Llorens i Artigas 4-6, 08028-Barcelona, Spain
`torras@iri.upc.edu`
`http://www-iri.upc.es/people/torras`

Abstract. The biological world offers a full range of adaptive mechanisms, from which technology researchers try to get inspiration. Among the several disciplines attempting to reproduce these mechanisms artificially, this paper concentrates on the field of Neural Networks and its contributions to attain sensorimotor adaptivity in robots. Essentially this type of adaptivity requires tuning nonlinear mappings on the basis of input-output information. Several experimental robotic systems are described, which rely on inverse kinematics and visuomotor mappings. Finally, the main trends in the evolution of neural computing are highlighted, followed by some remarks drawn from the surveyed robotic applications.

1 Introduction

Why the use of robots is not as widespread as some envisaged they would be by now? At the risk of oversimplification, I would say that it is due to their lack of adaptivity, at all levels. This capability is dispensable in well-engineered environments, and thus we have very performant robots in manufacturing lines, but it is a *sine qua non* when tasks are to be carried out in non-predefined worlds.

In this sense, the biological world – where adaptivity is crucial for survival – constitutes a very good source of inspiration for robotics researchers, since it provides existence proofs of many adaptive mechanisms that do function. However, caution must be taken, because the best natural solution may not be the best artificial one [40]. Wheels, wings and calculators have often been mentioned as examples of artificial solutions considerably different from their natural counterparts, and more performant according to certain criteria. The resources available to engineering design depart a lot from those in nature, and not just when it comes to materials, but also in the number of instances and spendable time.

With this note of caution in mind, i.e., accepting that biological plausibility in itself adds no special value from an engineering viewpoint, it is safe to look into natural adaptivity to get seed ideas that can be instantiated in a different way by artificial means.

* A more detailed version of this review, although less up to date, can be found in [48].

2 Natural and Artificial Adaptivity

We refer to adaptivity as the capability of self-modification that some agents have, which allows them to maintain a level of performance when facing environmental changes, or to improve performance when confronted repeatedly with the same situation. The term 'agent' above stands for a single cell, an organ, an individual or even a whole society, because, in the biological world, adaptivity occurs at several levels, each having a possible counterpart in the design of autonomous robots [42,14,53].

At the cell level, several chemical and electrical mechanisms of **plasticity** have been discovered, some of which have been modelled and analysed within the Neural Modelling field, and later applied to adjust the parameters of robot sensors and actuators. See the chapters on 'neural plasticity' in [3].

When referring to individuals, adaptation is usually called **learning** and it takes two rather different forms depending on whether it occurs at the sensorimotor or cognitive levels. Sensorimotor adaptation consists in building relevant associations between stimuli and responses, while cognitive learning entails constructing symbolic representations to guide decision-making. Two disciplines have tried to mimick these two capacities. Neural Networks, closer to Biology, has proven adequate to handle the massively-parallel tasks of perception and control of action, while Artificial Intelligence, steeming from Computer Science and Cognitive Psychology, has developed the necessary data structures and procedures to tackle symbolic learning [46,47]. Results in both disciplines have been applied to Robotics, the former to attain adaptive robot sensorimotor mappings [24], while the latter have led to so called learning robots [49,9,22,23].

Finally, at the species level, adaptation is attained through **evolution**. Genetic algorithms [11,17] and evolutionary computation [4,5] are starting to be used to tailor robot genotypes to given tasks and environments [15].

Table 1 summarizes the different adaptation levels and the involved disciplines.

In this paper we concentrate on adaptivity at the individual sensorimotor level, i.e. both the robot morphology and its components are assumed to be fixed and what may change with experience is the functional relationship between sensors and actuators.

Table 1. Levels of adaptation and related disciplines

ADAPTATION LEVEL	TYPE OF ADAPTIVITY	"ARTIFICIAL" DISCIPLINE
Cell	Plasticity	Neural Modelling
Sensorimotor	Associative learning	Neural Networks
Cognitive	Symbolic learning	Artificial Intelligence
Species	Evolution	Evolutionary Algorithms

3 Natural Inspiration for Artificial Neural Adaptivity

Neural networks are essentially procedures for approximating nonlinear mappings given a set of inputs and some information on the corresponding outputs. The approximation is attained by iteratively tuning the weights of the connections between neurons. This iterative process is referred to as neural adaptivity, leading to the desired input-output behaviour of the network.

The approaches to adaptivity pursued within the Neural Networks field have their roots in the learning paradigms developed in the domain of Behavioural Psychology (refer to Figure 1). This is the reason why the rules to attain neural adaptivity are usually called learning rules. The role of the animal in the behavioural learning experiments is played here by the neuron. It is worth noting that, although inspiration comes from the biological world, the artificial techniques are here applied not only to a different physical substrate, but also at a different level (neuron instead of animal).

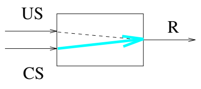
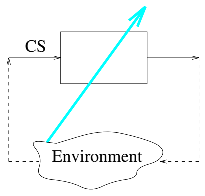
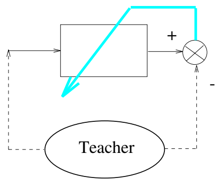
TYPE OF LEARNING	DEGREE OF FEEDBACK	DIAGRAM	KEYWORDS	NEURAL RULES
CLASSICAL CONDITIONING	none		Pavlov Unsupervised Open-loop	**CORRELATIONAL**
INSTRUMENTAL CONDITIONING	qualitative		Skinner Trial-and-error Optimal control	**REINFORCEMENT**
INPUT-OUTPUT TEACHING	quantitative		Supervised Reference-model control Closed-loop	**ERROR-MINIMIZATION**

Fig. 1. Learning paradigms inspiring neural adaptivity. In the diagrams, US stands for unconditioned stimulus, CS for conditioned stimulus and R for response.

The most basic learning paradigm is **classical conditioning**, as introduced by Pavlov [26], which consists of repeatedly presenting to an animal (e.g. a dog) an initially meaningless stimulus (e.g. the sound of a bell) together with an unconditioned stimulus (e.g. food) that triggers a reflex response (e.g. salivation). As a result of such paired presentations, the animal builds up an association so that, when presented with the conditioned stimulus (e.g. the sound of a bell) alone, it produces the same response as before (e.g. salivation). This type of learning is completely open-loop, in the sense that it entails no feedback. The neural learning rules mimicking this type of conditioning come from the classical Hebbian rule [13], in which a connection weight is adjusted according to the correlation between the activation of the two neurons connected, this being the reason why they are called *correlational rules*. Neural models incorporating these type of rules are Self-Organizing feature Maps (SOM) [16], the Cerebellar Model Articulation Controller (CMAC) [1], and Adaptive Resonance Theory (ART) [12].

Instrumental conditioning was introduced by Skinner [41] and requires that the animal under experimentation performs an arbitrary action (e.g. pressing a lever, walking around) in the presence of an initially meaningless stimulus (e.g. a flickering light). If the action is "appropriate" to the given stimular situation, the animal receives a reward (e.g. food). Otherwise, it receives nothing or punishment, depending on the particular experimental design being applied. Thus, this learning paradigm strongly relies on providing the animal with a reinforcement signal dependent on the action performed. This can be conceptualized as a qualitative feedback. The neural learning rules implementing this type of conditioning at the neuron level are known as *reinforcement-based rules* [43].

Note that the natural progression in the degree of feedback supplied suggests the use of a quantitative error signal to guide learning. This is represented in the third row of Figure 1 under the name of **input-output teaching**. Here, after presenting an input to the system and observing the emitted response, a teacher supplies the desired output whose difference with the emitted one provides the error signal which is fed back to the system. This is an entirely closed-loop learning process that requires perfect knowledge of the input-output pairs to be associated. The most widely used neural learning rules follow this scheme and we call them *error-minimisation rules*. They are mostly variants of the well-known backpropagation procedure [19,39] aimed at palliating its main drawbacks, namely catastrophic forgetting, overfitting, and a slow convergence rate.

Several techniques to prevent catastrophic forgetting [30] by explicitly minimising degradation of the previously learned patterns while encoding a new one [25,28] and by introducing noise [2,35] have been devised. Overfitting, i.e., the problem of learning a function too tailored to the samples and thus yielding a high generalisation error, is usually addressed by using methods for model complexity control [6,7] and, in particular, regularisation. An interesting observation is that many such methods lead to functional invariance [31,32,36], i.e., they converge to the same function irrespective of network size for fixed regularisation parameters.

4 Robot Sensorimotor Mappings

Motion control, both in biological and technological systems, relies strongly on sensorimotor mappings. These mappings vary considerably [45], depending not only on the nature of the involved sensors and actuators, but also on the goal pursued.

Tasks to be carried out by robots are usually specified in world coordinates (or, alternatively, in terms of sensor readings), while robot moves are governed by their actuator's variables. For instance, a sealing task may be specified as a given curve in 3D space or as following a prespecified visual pattern, but it has ultimately to be translated into currents sent to the motors governing the different joints. Therefore, robot control critically depends on the availability of accurate mappings from physical space or sensor space to joint space or motor space. The discussion in what follows is centered on mappings required for robot arms to work, but similar arguments apply to the case of mobile robots [20,21].

For a gripper to reach a desired position and orientation in space, the robot controller must access a mapping relating workspace coordinates to joint coordinates. This is called *inverse kinematics mapping*, because the natural (direct) map is that relating the values of the joint coordinates defining an arm configuration to the position and orientation of its end-effector (hand, gripper,...) in the workspace.

If a desired end-effector trajectory is specified instead, then the controller should resort to an *inverse dynamics mapping* relating such trajectory to the forces and torques that need to be exerted at the different joints to realize it. Note that this mapping, which is again called inverse for the same reason above, cannot be characterized uniquely in terms of inputs and outputs, it being instead dependent on state variables (or the short-term history of inputs) as it is usually the case with dynamic systems.

For tasks entailing the achievement of a goal using sensory feedback, programming even in terms of the coordinates of the workspace can be very complex. An example of this is the visual inspection of large objects that cannot be precisely placed (e.g., aircraft wings), since devising a detailed vision-based control strategy that moves the camera to the same relative position with respect to the object in all possible situations, and subject to real-world conditions of uncertainty and noise, is extremely difficult. What is needed to accomplish this type of tasks is an appropriate *sensorimotor mapping* relating sensory patterns to motor commands.

The diversity of the aforementioned mappings sometimes hides what they have in common: an underlying highly nonlinear relation between a continuous (often hard to interpret) input domain and a continuous motor domain; a relation that is very difficult (when not impossible) to derive analytically. Furthermore, because of environmental changes or robot tear-and-wear, the mappings may vary in time and then one would like the controller to adapt to these variations, without any human intervention if possible. Therefore, a way of learning (or tuning) these mappings automatically while robots move is highly desirable. Since, as we have mentioned, neural networks are essentially procedures for approximating nonlinear mappings, they constitute a good tool to attain the desired adaptivity.

In what follows we will describe some experiences related to the learning of two of the mappings mentioned above, namely inverse kinematics and visuomotor mappings.

5 Adaptive Inverse Kinematics

Making robots adaptive to changes in their own geometry (e.g., link bending, encoder shifting and other wear-and-tear deformations occasioned by regular functioning) would certainly widen their range of application. Since these geometric changes affect the robot inverse kinematics, the interest of using neural networks to approximate such mapping has been widely recognized. Especially when the operation conditions of the robot (in space, underwater, etc.) make it very hard to recalibrate it.

Along this line DASA (Daimler-Benz Aerospace S.A.), in the framework of the Advanced Servicing Robot project targeted at unmanned space stations, proposed an application of maintenance of electronic equipment that required the automatic recalibration of a 6-dof robot in-situ, since recalibration through teleoperation from earth is a very time-consuming task due to communication delays. After reviewing previous approaches to the learning of inverse kinematics, we will present the solution implemented in the REIS robot included in the space-station mock-up located at DASA's R&D laboratory, in Bremen, Germany (see Figure 2).

The conclusion reached after extensive experimentation with feedforward networks using backpropagation [18,46] is that a coarse mapping can be obtained quickly, but an accurate representation of the true mapping is often not feasible or extremely difficult. The reason for this seems to be the global character

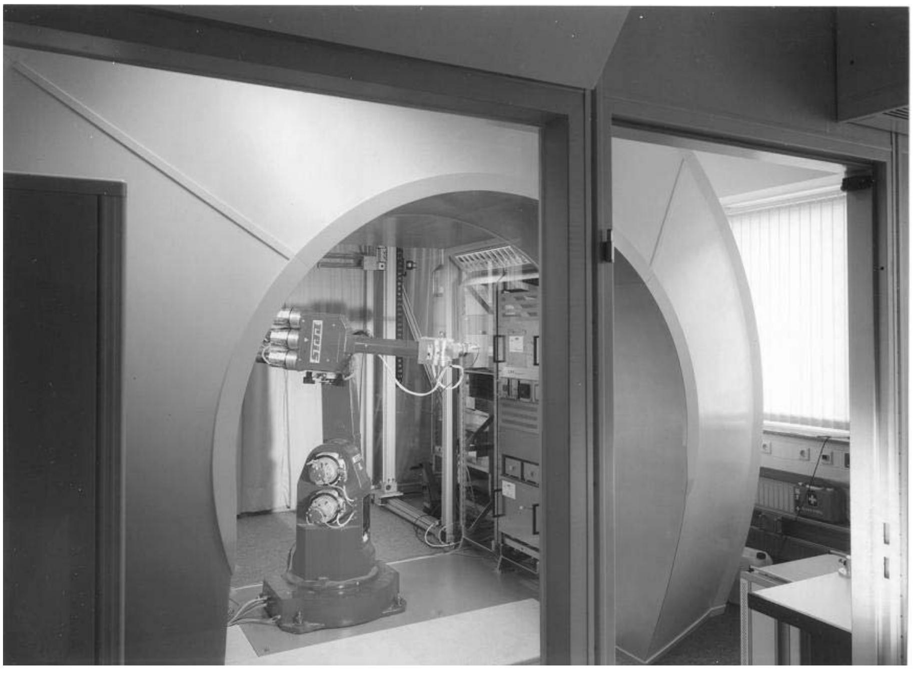


Fig. 2. Space station mock-up at Daimler-Benz Aerospace, Bremen

of the approximation obtained with this type of networks using sigmoid units: every connection weight has a *global* effect on the final approximation that is obtained [18].

An obvious way to avoid this global effect is of course using local representations, so that every part of the network is responsible for a small subspace of the total input space. Thus, Ritter et al. [27] have used a self-organizing map (SOM) together with an error-minimisation rule to learn the visuomotor mapping of a robot arm with three degrees of freedom (dof) in 3D space. The target position of the end-effector is defined as a spot registered by two cameras looking at the workspace from two different vantage points. Neurons are arranged in a 3D lattice to match the dimensionality of physical space. It is expected that learning will make this lattice converge to a discrete representation of the workspace.

Extensive experimentation by Ritter et al. [27] and other groups has shown that the network self-organizes into a reasonable representation of the workspace in about 30.000 learning cycles. This should be taken as an experimental demonstration of the powerful learning capabilities of this approach, because the conditions in which it is made to operate are the worst possible ones: no a priori knowledge of the robot model, random weight initialization, and random sampling of the workspace during training.

This basic model has been extended in three directions to cope with higher-dof robots. First, a hierarchical version, consisting of a 3D SOM whose nodes have associated a 2D SOM each, was applied to a 5-dof robot. The 3D net encodes the workspace as before, while each 2D subnet approximates the end-effector orientation space at the corresponding position [27].

Ruiz de Angulo and Torras [29] adapted this hierarchical model to suit a practical setting. Thus, instead of learning the kinematics from scratch, only the deviations from the nominal kinematics embedded in the original robot controller are learnt. This, together with informed initialization and sampling, as well as several modifications in the learning algorithm aimed at improving the cooperation between neurons, led to a speed-up of two orders of magnitude with respect to the original model.

The resulting algorithm constitutes the core of the recalibration system that was installed in the REIS robot included in the space-station mock-up located at DASA, as mentioned above. Figure 2 is a photograph of such a set-up, where the different racks containing the electronic cards that the robot should maintain are shown. The robot must reach the handles of the racks with enough precision to be able to pull them out and, afterwards, extract a faulty card in order to replace it by another one. Although testing in this set-up has been constrained by the need to preserve robot integrity, the system has proven able to correct large miscalibrations of the robot: 95% of the decalibration was corrected with the first 25 movements, this percentage raising to 98% after 100 movements [29]. Moreover, other desirable features in stand-alone applications, such as parameter stability, are guaranteed.

The third extension of the basic model relies on the generalisation of SOMs to parameterized SOMs (called PSOMs). The idea is to turn the discrete representation into a continuous one by associating a basis function to each neuron, so that a parameterized mapping manifold is obtained. Moreover, PSOMs make no distinction between inputs and outputs, thus encoding bidirectional mappings. The PSOM reduces considerably the number of training samples required to attain a given precision as compared to the SOM [50], allowing the learning of the full inverse kinematics of a 6-dof robot with less than 800 movements.

The main drawback of using neural networks to approximate the inverse kinematics of robot arms is precisely the high number of training samples (i.e., robot movements) required to attain an acceptable precision. A trick has been proposed [33,37], valid for most industrial robots, that greatly reduces the number of movements needed to learn or relearn the mapping to a given accuracy. This trick consists in expressing inverse kinematics as a composition of learnable functions, each having half the dimensionality of the original mapping. A training scheme to learn these component functions has also been proposed. Experimental results obtained by using PSOMs, with and without the decomposition, show that the time savings granted by the proposed scheme grow polynomically with the precision required.

The aforementioned trick assumes that the last three robot joints cross at a point, a condition satisfied by some classic robot architectures, but not by other more innovative ones. Therefore, a more general decomposition technique applicable to any serial robot has recently been developed [38], which still retains the main advantage of the trick above: The input dimensionality of each of the tasks resulting from the decomposition is half that of the original one. Thus, for a given desired accuracy, if the number of training samples required to learn inverse kinematics directly is $O(n^d)$, through the decomposition it reduces to $O(n^{d/2})$. This yields an enormous reduction in the number of samples required for high-precision applications.

The development of humanoid robots has recently raised the interest in learning inverse kinematics. Due to the many dof's involved, the aim is no longer learning the mapping for the whole workspace, but focussed on a specific trajectory. Following the trend of using localized representations, D'Souza et al. [8] have applied a supervised algorithm –locally weighted projection regression– in this context, with promising results.

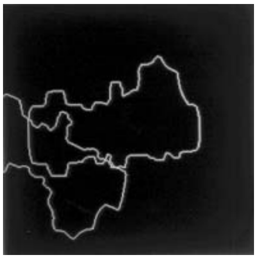
6 Adaptive Visuomotor Mappings

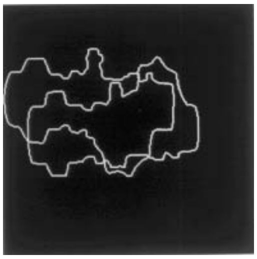
Depending on the task to be performed and the camera-robot arrangement, visuomotor mappings take different forms. Thus, in eye-hand coordination, where cameras external to the robot are used to monitor the pose (position and orientation) of its end-effector, a mapping from the camera coordinates of a desired end-effector pose to the joint angles that permit attaining that pose is sought. This mapping is closely related to the inverse kinematics one, especially if the camera coordinates of selected points in the end-effector uniquely characterize its pose. Therefore, the same models used to learn inverse kinematics have been applied to the learning of the visuomotor mapping underlying eye-hand cooordination [27].

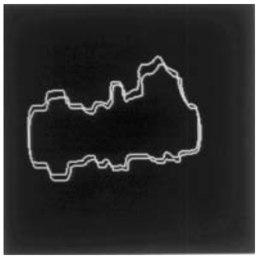
a) Robot and camera

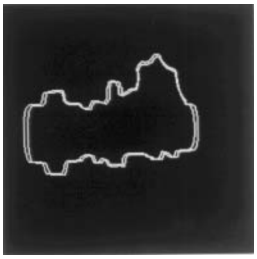
b) Reference image

c) Initial image


d) Initial contours


e) After 1 movement


f) After 5 movements


g) After 7 movements

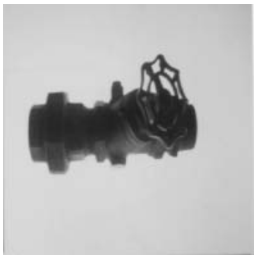

h) Final image

Fig. 3. Visual positioning system developed in collaboration with Thomson Broadcast Systems

A camera mounted on a robot arm is used in tasks such as visual positioning and object tracking. The goal of these tasks is to move the camera so that the image captured matches a given reference pattern. The target is thus no longer a position of the robot in space but a desired image pattern, and the desired visuomotor mapping needs to relate offsets w.r.t. that pattern with appropriate movements to cancel them. In visual positioning, the scene is assumed to be static and the main issue is to attain high precision. Applications include inspection and grasping of parts that cannot be precisely placed (e.g., in underwater or space settings). The aim of object tracking is to maintain a moving object within the field of view, speed being here the critical parameter instead of precision.

The classical way of tackling these tasks consists of defining a set of image features (corners, holes, etc.) and then deriving an interaction matrix relating 2D shifts of these features in the image to 3D movements of the camera [10]. The advantages of applying neural networks to this task are the direct learning of the interaction matrix, as well as avoiding the costly matching of features in the current and reference images.

The latter approach has been used in an application developed for Thomson Broadcast Systems [52] for the inspection of large objects (e.g., ship hulls, airplane wings, etc.). Since these objects are difficult to move, it is the camera that has to travel to attain a pre-specified position and orientation with respect to the object. The developed camera control system consists of a feedforward network trained with backpropagation. The training procedure consists of moving the camera from the reference position to random positions and then using the displacement in image features together with the motion performed as input-output pairs. In operation, the robot is commanded to execute the inverse of the motion that the network has associated to the given input.

The key option in this work is the use of global image descriptors, which permits avoiding the costly matching of local geometric features in the current and reference images. By using a statistical measure of variable interdependence (the mutual information criterion), sets of global descriptors as variant as possible with each robot dof are selected from a battery of features, including geometric moments, eigenvectors, pose-image covariance vectors and local feature analysis vectors [51]. The results obtained with a 6-dof show that, after 10.000 learning cycles, translation and rotation errors are lower than $2mm$ and 0.1 degrees, respectively. Figure 3 shows the robot-mounted camera and the reference image of an object to be inspected (a water valve), together with several snapshots along the visual positioning process. In this case, the silhouette of the object could be readily extracted and 32 Fourier descriptors coding it were used as image features. It can be observed that, after 7 movements, the captured image is practically registered with the reference one.

7 Conclusions

This paper has reviewed the ways in which neural computing may help to increase sensorimotor adaptivity in robots. The mechanisms of neural adaptivity have been inspired in the learning paradigms of Behavioural Psychology (classical and instrumental conditioning), and fall into three classes that require progressively more feedback: correlational, reinforcement and error-minimisation rules.

Some trends in the development of these learning rules deserve notice, since they have parallels in other disciplines dealing with adaptivity at diferent scales, such as Evolutionary Computation and Artificial Intelligence. The first trend is that of progressing from binary variables to continuous ones. This entails moving from discrete search spaces and classification tasks to manifold representations and function approximation. Then, issues such as model complexity control [7] and functional invariance [36] become very important.

A second trend is that of progressively replacing local feedback for more global one, this globalisation taking place both spatially and temporally. The first learning rules proposed required feedback to be supplied to each single neuron. Backpropagation made a big step forward in allowing feedback to be supplied at the overall network level (spatial globalisation). Reinforcement learning has greatly contributed to dealing with deferred feedback (temporal globalisation).

The dichotomy between locality and globality appears also in the state space representation. Correlational rules are often incorporated into network models that build localised representations (such as SOM, ART and CMAC), while the strength of most models based on error-minimisation and reinforcement rules lies precisely in the distributed (global) way in which information is represented across all the network weights. In the localised representations, appropriately tuning the neighbourhood size is a key issue.

Moving from off-line to on-line learning is another trend observed in neural computing. Initial learning procedures were designed to work in a batch mode (with all training samples supplied at the same time), while a later challenge was to incorporate new samples into an already trained network. Sequential learning addresses this challenge by explicitly seeking to avoid catastrophic forgetting [34].

Finally, let us mention the important role that randomness plays in learning. This has been widely acknowledged in many contexts, but specifically in neural computing noisy inputs and weights have proven useful for regularisation (a complexity control method), and randomness is of course a key ingredient of reinforcement learning.

After the overview of neural adaptivity, the paper has focused on its application to robot control. This basically entails the learning of nonlinear mappings relating stimuli to responses. Several robotics applications have been surveyed, classified according to the underlying mapping that needs to be approximated: inverse kinematics and visuomotor mappings.

The learning of inverse kinematics makes robot arms adaptive to changes in their own geometry (e.g., link bendings, encoder shiftings, etc.), while learning of visuomotor mappings allows robots to cope with changing environments (e.g., different loads, moving objects, etc.).

A first remark stemming from the survey of robotic applications is that in the case of mappings that can be easily sampled, it seems sufficient to apply a plain error-minimisation procedure. Some simple inverse kinematics mappings and visuomotor mappings used for visual positioning have been learned in this way. If the input space is complex, then many researchers have resorted to a combination of correlational rules for the efficient coding of that space, with error-minimisation rules to build the appropriate association with the outputs. The use of SOMs to encode the robot workspace or the sensor space falls into this category. Then, an error-minimisation rule is used to build the appropriate input-output mapping: inverse kinematics in this case. Finally, when the task is specified as a goal to be reached using sensory feedback, without making explicit the movements necessary to reach it, then the only possibility is to resort to reinforcement learning schemes, which depend just on the availability of a measure of success rather than an error measure.

The number of learning cycles required ranges widely in the applications described, depending on the complexity of the mapping to be learned as well as on the accuracy required. Note that learning time is directly related to the number of training samples, each of which entails at least one robot movement. And robots are slow as compared to computers. Therefore, minimising the number of training samples is of paramount importance in the application of neural networks to robotics, and many efforts are currently oriented in this direction (e.g., adaptive sampling, function decomposition) [37,38].

Acknowledgments

This work has been partially supported by the Catalan Research Commission, through the "Robotics and Control" group. The author thanks Bernd Maediger, from Daimler-Benz Aerospace, and Christophe Venaille, from Thomson Broadcast Systems, for providing the specifications and the robot set-ups for the applications described in Sections 5 and 6 in the framework of the former Esprit project CONNY, and also for supplying the photographs in Figures 2 and 3, respectively.

References

1. Albus JS (1975) A new approach to manipulator control: The cerebellar model articulation controller (CMAC). Transactions of the ASME, Journal of Dynamic Systems, Measurement and Control 97: 220-227
2. An G (1996) The effects of adding noise during backpropagation training on generalization performance. Neural Computation 8: 643-674
3. Arbib MA (1995) Handbook of Brain Theory and Neural Networks. MIT Press, Cambridge, MA (An updated second edition will appear in November 2002)
4. Bäck T, Fogel DB and Michalewicz Z (eds) (1997) Handbook of Evolutionary Computation. Oxford University Press, New York, and Institute of Physics Publishing, Bristol
5. Beyer H-G and Schwefel H-P (2002) Evolution strategies - A comprehensive introduction. Natural Computing 1(1): 3-52
6. Bishop C (1995) Neural Networks for Pattern Recognition. Oxford University Press
7. Cherkassky V (2002) Model complexity control and statistical learning theory. Natural Computing 1(1): 109-133

8. D'Souza A, Vijayakumar S and Schaal S (2001) Learning inverse kinematics. Proc. IEEE/RSJ Conf. on Intel. Robots and Systems, Maui, Hawaii, USA, pp. 298-303
9. Dorigo M (ed) (1996) Special issue on 'Learning Autonomous Robots'. IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics 26(3)
10. Espiau B, Chaumette F and Rives P (1992) A new approach to visual servoing in robotics. IEEE Trans. on Robotics and Automation 8(3): 313-326
11. Goldberg DE (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA
12. Grossberg S (1987) Competitive learning: from interactive activation to adaptive resonance. Cognitive Science 11: 23-63
13. Hebb DO (1949) The Organization of Behavior. Wiley, New York
14. Higuchi T, Iwata M and Liu W (eds) (1997) Evolvable Systems: From Biology to Hardware. Springer-Verlag, Berlin Heidelberg New York
15. Husbands P and Meyer JA (1998) Proc. 1st European Workshop on Evolutionary Robotics (EvoRobot'98). Springer-Verlag, Berlin Heidelberg New York
16. Kohonen T (2001) Self-Organizing Maps (third edition). Series in Information Sciences 30, Springer-Verlag, Berlin Heidelberg New York Tokyo
17. Koza JR (1992) Genetic Programming: On Programming Computers by means of Natural Selection. MIT Press, Cambridge, MA
18. Kröse BJA and van der Smagt PP (1993) An Introduction to Neural Networks (5th edition). University of Amsterdam
19. LeCun Y (1985) Une procedure d'aprentissage pour reseau au seuil assymetrique. Proc. of COGNITIVA, pp. 599-604
20. Millán JR and Torras C (1992) A reinforcement learning connectionist approach to robot path finding in non-maze-like environments. Machine Learning 8(3/4): 363-395
21. Millán JR and Torras C (1999) Learning sensor-based navigation. In: Morik K, Kaiser M and Klingspor V (eds) Making Robots Smarter: Combining Sensing and Action through Robot Learning. Kluwer Academic Publishers, Boston, MA
22. Mitchell T, Franklin J and Thrun S (1996) Recent Advances in Robot Learning. Kluwer Academic Publishers, Boston, MA
23. Morik K, Kaiser M and Klingspor V (1999) Making Robots Smarter: Combining Sensing and Action through Robot Learning. Kluwer Academic Publishers, Boston, MA
24. Omidvar O and van der Smagt P (1997) Neural Systems for Robotics. Academic Press, San Diego, CA
25. Park DC, El-Sharkawi MA and Marks II RJ (1991) An adaptively trained neural network. IEEE Trans. on Neural Networks 2(3): 334-345
26. Pavlov IP (1927) Conditioned Reflexes. Oxford University Press
27. Ritter H, Martinetz T and Schulten K (1992) Neural Computation and Self-Organizing Maps. Addison Wesley, New York
28. Ruiz de Angulo V and Torras C (1995) On-line learning with minimum degradation in feedforward networks. IEEE Trans. on Neural Networks 6(3): 657-668
29. Ruiz de Angulo V and Torras C (1997) Self-calibration of a space robot. IEEE Trans. on Neural Networks 8(4): 951-963
30. Ruiz de Angulo V and Torras C (2000) A framework to deal with interference in connectionist systems. AI Communications 13(4): 259-274
31. Ruiz de Angulo V and Torras C (2001) Architecture-independent approximation of functions. Neural Computation 13(5): 1119-1135
32. Ruiz de Angulo V and Torras C (2001) Neural learning invariant to network size changes. Proc. Intl. Conf. on Artificial Neural Networks (ICANN'01), Vienna, Austria, Lecture Notes in Computer Science 2130: 33-40
33. Ruiz de Angulo V and Torras C (2002) Learning inverse kinematics via cross-point function decomposition. Proc. Intl. Conf. on Artificial Neural Networks (ICANN-2002), Madrid, Spain, Lecture Notes in Computer Science 2415: 856-861

34. Ruiz de Angulo V and Torras C (2002) Sequential learning in feedforward networks: proactive and retroactive interference minimization. Proc. Intl. Conf. on Artificial Neural Networks (ICANN-2002), Madrid, Spain, Lecture Notes in Computer Science 2415: 1339-1344
35. Ruiz de Angulo V and Torras C (2002) A deterministic algorithm that emulates learning with random weights. Neurocomputing 48(1-4): 975-1002
36. Ruiz de Angulo V and Torras C (2004) Neural learning methods yielding functional invariance. Theoretical Computer Science 320(1): 111-121
37. Ruiz de Angulo V and Torras C (2005) Speeding up the learning of robot kinematics through function decomposition. IEEE Trans. on Neural Networks, Nov.
38. Ruiz de Angulo V and Torras C (2005) Using PSOMs to learn inverse kinematics through virtual decomposition of the robot. Proc. 8th Intl. Work-Conf. on Artificial Neural Networks (IWANN'05), Vilanova i la Geltrú, Spain, Lecture Notes in Computer Science 3512: 701-708
39. Rumelhart DE, Hinton GE and Williams RJ (1986) Learning representations by back-propagating errors. Letters to Nature 323: 533-535
40. Simon HA (1969) The Sciences of the Artificial. MIT Press, Cambridge, MA
41. Skinner BF (1938) The Behavior of Organisms: An Experimental Analysis. Appleton Century
42. Steels L (1995) The Biology and Technology of Intelligent Autonomous Agents. NATO ASI Series F, Springer-Verlag, Berlin Heidelberg New York
43. Sutton RS and Barto AG (1998) Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA
44. Torras C (1985) Temporal-Pattern Learning in Neural Models. Lecture Notes in Biomathematics 63, Springer-Verlag, Berlin Heidelberg New York
45. Torras C (1989) Sensorimotor integration in robots. In: Ewert P and Arbib MA (eds) Visuomotor Coordination: Experiments, Comparisons, Models and Robots, Plenum Press, pp. 673-689
46. Torras C (1993) Symbolic planning versus neural control in robots. In: Rudomín P, Arbib MA, Cervantes-Pérez F and Romo R (eds.) Neuroscience: From Neural Networks to Artificial Intelligence, Research Notes in Neural Computing 4: 509-523, Springer-Verlag: Berlin Heidelberg New-York
47. Torras C (ed) (2001) Special issue on 'Neural Networks at IJCAI'01'. Intl. Journal of Computational Intelligence and Applications 1(4)
48. Torras C (2002) Neural computing increases robot adaptivity. Natural Computing 1(4): 391-425
49. van de Velde W (ed) (1993) Special issue on 'Towards Learning Robots'. Robotics and Autonomous Systems 8(1-2)
50. Walter J and Ritter H (1996) Rapid learning with parametrized self-organizing maps. Neurocomputing 12: 131-153
51. Wells G and Torras C (2001) Assessing image features for vision-based robot positioning. Journal of Intelligent and Robotic Systems 30(1): 95-118
52. Wells G, Venaille Ch and Torras C (1996) Vision-based robot positioning using neural networks. Image and Vision Computing 14: 715-732
53. Ziemke T and Sharkey N (eds) (1998) Special issue on 'Biorobotics'. Connection Science 10(3-4)

On Self-assembly in Population P Systems

Francesco Bernardini[1], Marian Gheorghe[1], Natalio Krasnogor[2], and Jean-Louis Giavitto[3]

[1] Department of Computer Science, The University of Sheffield, Regent Court, Portobello Street, Sheffield S1 4DP, UK
{F.Bernardini, M.Gheorghe}@dcs.shef.ac.uk

[2] Automated Scheduling, Optimisation and Planning Research Group, School of Computer Science and Information Technology, University of Nottingham, Jubilee Campus, Nottingham NG8 1BB, UK
Natalio.Krasnogor@nottingham.ac.uk

[3] Laboratoire de Méthodes Informatiques UMR 8042, CNRS-Univesité d'Evry, Tour Evry 2, GENOPOLE 523, Place de terasses de l'agora, 91000, Evry, France
giavitto@lami.univ-evry.fr

Abstract. We introduce a model of self-assembly P systems as devices that use some of the features of population P systems to progressively grow a graph structure by forming new bonds between the existing cells and some new cells which are brought into the system step by step. The new cells are then able to self-assemble locally either at the level of cells or at the level of neighbourhoods of cells by using bond-making rules according to a specific self-assembly model. We describe two self-assembly models, called respectively parallel single-point self-assembly and parallel multi-point self-assembly. Then, we precisely state the problem of programmable self-assembly for P systems as the problem of uniquely generating a given graph by means of self-assembly P systems. In this respect, we show how to define a self-assembly P systems that uniquely generates a complete binary tree by using a "minimal" set of resources.

1 Introduction

Self-assembly is the ubiquitous process by which simple individual components autonomously assemble into intricate complexes, which is now being studied in many different research areas of molecular biology, nanotechnology, robotics, and natural computing. In this respect, a number of (abstract) models for self-assembly have been proposed and the problem of having programmable self-assembly models has been identified as a key issue in self-assembly related research. Programmable self-assembly means defining a formalism that can help the systematic (or, even better, automatic) design of an appropriate set of components and the associated interactions which will make these components autonomously, robustly and efficiently assemble to form a desired shape or pattern [5]. In the existing literature, two main approaches to the study of programmable self-assembly models have been considered: an incremental/generative approach (e.g, tiles [8], amorphous computing [1]) where a shape is generated in an incremental way by progressively adding to the existing structure a certain number of components in correspondence to some specific "growing points"; a distributed approach where the desired shape results from the spatial re-organization of some already existing components [4]. In this paper, we propose an incremental/generative approach for the self-assembly of a graph that is based on P systems, a fairly new computational model which abstracts from the structure and functioning of living cells [6]. In particular, we focus on the population P system variant introduced in [2], which provides a formalism for modelling abstract systems consisting of a population of individual components, called cells, which are linked together to form a graph structure; cells interact each other by means of the existing set of links, which is continuously updated by means of some bond-making rules specifying how to add/remove links between the cells in the system. Here, bond-making rules are used in a self-assembly process to progressively enlarge an existing graph structure by forming new bonds between the existing cells and some new cells which are brought into the system step by step; bond-making rules must be used according to a specific self-assembly model. In this respect, we present two self-assembly models where bond-making rules are restricted to be used "locally" either at the level of cells or at the level of neighbourhood of cells. Then, we precisely state the problem of programmable self-assembly, the problem of defining a self-assembly P system that uniquely generates a given target graph. Finally we show how a complete binary tree can be uniquely generated by a self-assembly P system by using a "minimal" set of resources.

2 Preliminaries

We recall here some basic notions and notations commonly used in membrane computing as well as some formal language concepts we need in the rest of the paper. We refer to [6], [7] for further details.

An alphabet is a finite non-empty set of abstract symbols. Given an alphabet O, we denote by O^* the set of all possible strings over O, including the empty string λ. The length of a string $x \in O^*$ is denoted by $|x|$ and, for each $a \in O$, $|x|_a$ denotes the number of occurrences of the symbol a in x. A multiset over O is a mapping $M : O \longrightarrow \mathbf{N}$ such that, $M(a)$ defines the multiplicity of a in the multiset ($\mathbf{N}$ denotes the set of natural numbers). Such a multiset can be represented by a string $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in O^*$ and by all its permutations with $a_j \in O$, $M(a_j) \neq 0$, $1 \leq j \leq n$. In other words, we can say that each string $x \in O^*$ identifies a finite multiset over O defined by $M_x = \{\, (a, |x|_a) \mid a \in O \,\}$. Moreover, given two strings $x, y \in O^*$, we denote by xy their concatenation, which corresponds to the union of the multisets represented by the string x, y.

A finite undirected graph is a pair $G = (V, E)$ where $V \subseteq \mathbf{N}$ if a finite set of nodes, and $E \subseteq V \times V$ is a finite set of unordered pairs called edges; the edges in the graph G are denoted by using the notation $\{i, j\}$, with $i, j \in V$. We restrict our discussion to finite undirected graphs and therefore we will simply use the term graph. A graph $G = (V, E)$ is said to be cyclic if and only if E contains at least a subset of edges of the form $\{i, i_1\}, \{i_1, i_2\}, \ldots, \{i_{n-1}, i_n\}, \{i_n, i\}$, with $n \geq 1$; a graph $G = (V, E)$ is said to be connected if and only if, for each $i \neq j \in V$, either $\{i, j\} \in E$ or there exist $i_1 \neq i_2 \neq \ldots \neq i_{n-1} \neq i_n \in V$, with $n \geq 1$, such that, $\{i, i_1\} \in E$, $\{i_t, i_{t+1}\} \in E$, for each $1 \leq t \leq n-1$, and $\{i_n, j\} \in E$. A tree is a connected acyclic graph where all the nodes are thought as being descendants of an unique node called root; the depth d of a tree is the length of the longest path from the root to another node different from the root. The nodes placed at depth d are called leaves whereas, the nodes placed at depth p, with $1 \leq p \leq d-1$, are called intermediate nodes. A complete n-ary (binary if $n = 2$) tree of depth $d \geq 0$, with $n \geq 1$, is a tree where, for each $0 \leq p \leq d$, the number of nodes placed at level p is exactly n^p. Finally, given two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, we say G_1 is isomorphic to G_2, and we write $G_1 \approx G_2$ if and only if, there exists a bijective mapping $h : V_1 \longrightarrow V_2$ such that, for each $i, j \in V_1$, $\{i, j\} \in E_1$ iff $\{h(i), h(j)\} \in E_2$.

3 Self-assembly P Systems

We call self-assembly P systems a family of P systems describing a population of cells that self-assemble together to form a graph structure. Cells are the basic functional units of the system and they correspond to nodes in a graph which, at any moment, defines the structure of the system. The edges in such a graph represent links which tightly bond the cells to each other. Such a configuration consisting in a population of cells linked to form a graph structure is called an assembly of cells. Each cell in a given assembly contains a finite multiset of objects which is continuously updated by means of a finite set of transformation rules and communication rules. Transformation rules are used inside the cells to consume some objects in order to produce some new ones; communication rules are instead used to move objects from one cell to the other by using the edges in the graph as if they were communication channels. As well as this, cells in a given assembly can form bonds with some new cells that, step by step, are brought into the system in order to enlarge the current population of cells and form a new graph structure. These bonds are created by some bond-making rules which specify how to connect two cells in the system depending on their respective contents. More precisely, in each step of a self-assembly process, by starting from a given assembly of cells, we first update the content of each cell by using their respective sets of transformation and communication rules; then we introduce into the system some new cells which self-assemble by using bond-making rules to connect themselves to the existing structure according to a chosen self-assembly model.

The definition of self-assembly P systems proposed here is developed alongside the population P system model introduced in [2] where bond-making rules were used for the first time in order to define P systems with a dynamic graph structure.

Definition 1. *A self-assembly P system is a construct*

$$\mathcal{P} = (O, L, \Gamma, \sigma, R, B)$$

where:

- *O is a finite alphabet of symbols called objects;*
- *L is a finite alphabet of symbols called labels;*
- *Γ is a finite set containing t distinct cell templates of the form $C_i = (x_i, l_i, m_i)$ with $1 \leq i \leq t$, $x_i \in O^*$, $l \in L$, and $m_i \geq 1$ the number of bonds that can be formed by that cell;*
- *$\sigma = (w, q, b)$, is the seed cell with $w \in O^*$, $q \in L$, $b \geq 1$ the number of bonds that can be formed by the seed cell;*
- *R is a finite set of rules of the forms:*
  1. *$[x \to y]_l$, with $x \in O^+$, $y \in O^*$, $l \in L$ (transformation rules),*
  2. *$[x; y, in]_l$, with $x, y \in O^*$, $l \in L$ (communication rules);*
- *B is a finite set of bond-making rules of the form $(l, x; y, l')$, with $l, l' \in L$, $x, y \in O^*$ and, for some $1 \leq i \leq t$ and $z \in O^*$, $C_i = (x_i, l_i, m_i) \in \Gamma$, $x_i = yz$, $l_i = l'$.*

The symbols in O are used for the objects that can be contained inside the cells whereas, the symbols in L are instead used for labelling the cells and they are necessary to retrieve the subset of rules from R to be used inside a specific cell.

The set Γ contains a finite number of distinct cell templates $C_1, C_2, \ldots, C_t$, with $t \geq 1$; the templates in Γ can be instantiated by cloning an arbitrary number of copies, which can then be added to a given assembly of cells as to enlarge the current structure of the assembly. At the beginning, the initial assembly of cells is given by the seed cell σ and the graph containing only the node associated with this cell and no edges.

Each cell in a self-assembly P system, as well as each cell template, is characterised by a finite multiset of objects defining its content, by a label from L identifying the rules which can be used inside that cell and by a positive integer providing a bound for the total number of bonds which can be formed by that cell. The value of this bound is decreased by one every time a cell form a new bond and this makes sure that, at any moment, the current value of such a bound corresponds to the number of bonds which can still be formed by that cell. A cell can form a new bond if and only if its value of the bound on the number of bonds is greater or equal to 0. A clone of a cell template is a cell that inherits from a template in Γ the initial information about its content, its label, and its bound on the number of bonds.

A transformation rule $[x \to y]_l$ in R is an usual multiset rewriting rule specifying that, inside a cell with label l, an occurrence of a multiset x can be replaced by an occurrence of a multiset y. A communication rule $[x; y, in]_l$ in R instead specifies that, in presence of a multiset x, a cell with label l can receive an occurrence of the multiset y from one of its neighbouring cells; communication rules are executed non-deterministically; a neighbouring cell is a cell that is directly linked to the cell where the communication rule is applied.

Finally, we have a finite set of bond-making rules in B containing rules of the form $(l, x; y, l')$, with $x, y \in O^*$, $l, l' \in L$. A bond making rule $(l, x; y, l')$ must be read from left to right and it specifies that a cell with label l', containing an occurrence of the multiset x and already present in the current assembly of cells, can form a bond with a new cell which is being added to the current assembly in order to enlarge the existing graph structure if and only if, this cell is a clone of cell template in Γ containing an occurrence of the multiset y and having label l'. Objects are not consumed by bond-making rules but they are rather used as "resources" to be allocated to the bond-making rules in order to determine the number of bonds that can be effectively formed between each cell in the current assembly and the new cells that are being added in order to enlarge the existing graph structure. In particular, the same occurrence of a given multiset of objects can be used only by one bond making rule at a time.

Next, we formally introduce the notion of an assembly of cells in a self-assembly P system $\mathcal{P}$ and clarify the notion of a derivation in such a system.

Definition 2. *Let $\mathcal{P} = (O, L, \Gamma, \sigma, R, B)$ be a self-assembly P system as specified in Definition 1. An assembly of cells in $\mathcal{P}$ is a tuple $\mathcal{A} = (\sigma_1, \sigma_2, \dots, \sigma_n, \gamma)$ where:*

- *$\sigma_i = (w_i, q_i, b_i)$, for each $1 \leq i \leq n$, is a cell with $w_i \in O^*$ and $q_i \in L$, and $b_i \geq 0$ the number of bonds that can be formed by that cell;*
- *$\gamma = (\{1, 2, \dots, n\}, E)$, with $E \subseteq \{\{i, j\} \mid 1 \leq i \neq j \leq n\}$, is a connected graph defining the structure of the assembly.*

We also say that the assembly $\mathcal{S} = (\sigma_1, (\{1\}, \emptyset))$, with $\sigma_1 = \sigma$, is the seed assembly of $\mathcal{P}$.

Now, given a self-assembly P system $\mathcal{P}$, a step of derivation is performed in two separate stages: a stage of evolution-communication and a stage of self-assembly.

1. Evolution-communication: we apply the rules in R inside each cell in the current assembly in a non-deterministical maximal parallel manner. This stage of evolution-communication can be considered as being the same as in [2] where an analogous evolution-communication stage is defined that deals with the same type of rules.
2. Self-assembly: a certain number of clones of the cell templates in Γ are added to the current assembly of cells by connecting them to the existing structure by using the bond-making rules in B according to a specific self-assembly model.

This latter stage of self-assembly corresponds to the stage of bond-making considered in [2] for defining the notion of a computation in a population P system. Here the difference with respect to [2] is that we do not destroy any existing bond but we rather increase the structure by adding new bonds and new cells. Specifically, in our model, a bond, as well as a cell, once introduced in an assembly of cells, can never be removed from in any further step of derivation.

In the next two subsections, we will present two different self-assembly models which are defined by imposing particular restrictions on the use of the bond-making rules.

3.1 Parallel Single-Point Self-assembly

We consider here a self-assembly model where, at each stage of self-assembly process, each cell in the current assembly serves as an acretion point where new cells cloned from the template set can attach. This process occurs simultaneously for each of the cells already assembled and hence the name "parallel single-point" self-assembly. More specifically, for each already assembled cell we non-deterministically select a maximal set of new clones to be connected to that cell forming a bond between the later and each clone in the selected set; this set of new cells must be maximal with respect to both the particular choice of bond-making rules, the current distribution of objects inside the cell, and the number of bonds that can be effectively formed by that cell which, by definition, is bounded by a fixed constant. The main restriction in this self-assembly model is that, after the application of the bond-making rules, each new cell ends-up connected to the graph defining the structure of the assembly only by means of a single bond that is created between this new cell and a specific pre-existing cell. Note that bond cannot be formed between two new cells introduced during the same stage of self-assembly. Moreover, as we want the resulting graph to be connected, we also impose the constraint that a new cell is added if and only if a new bond can be effectively formed.

More formally, let $\mathcal{P} = (O, L, \Gamma, \sigma, R, B)$ be a self-assembly P system as specified in Definition 1 and let $\mathcal{A} = (\sigma_1, \sigma_2, \dots, \sigma_n, \gamma)$, for some $n \geq 1$, be an assembly of cells in $\mathcal{P}$ as specified in Definition 2. We write $A \overset{sa}{\Longrightarrow}_{\mathcal{P}} \mathcal{A}'$, and we say $\mathcal{A}'$ is derived from $\mathcal{A}$ by single-point self-assembly, if and only if $\mathcal{A}'$ is an assembly of cells in $\mathcal{P}$ which is obtained from $\mathcal{A}$ in the following way.

1. For each cell $\sigma_i = (w_i, q_i, b_i)$ in $\mathcal{A}$, with $w_i \in O^*$, $q_i \in L$, $b_i > 0$, $1 \leq i \leq n$, we select a maximal set of bond-making rules from B to be used to link this cell with a maximal number of clones of the cell templates in Γ. This is set of bond-making rules is constructed by assigning in a non-deterministical way the objects in w_i to the rules B as far as it is possible (i.e., all the objects that can be assigned to some bond-making rules must be assigned to some bond-making rule). However, the total number of bond-making rules to be applied must not be greater than b_i. Then, for each bond-making rule $(q_i, x; y, l')$ in B selected to be applied during this self-assembly stage, and for each occurrence of the multiset x in w_i assigned to that rule, we introduce into the new assembly $\mathcal{A}'$ a new cell $\sigma'_h = (w'_h, q'_h, b'_h - 1)$ such that: $h > n$ is a new index which has not yet been used for any other cell in $\mathcal{A}'$, $w'_h = yz = x_j$, $q'_h = l_j$, $b'_h = m_j$, for some $C_j = (x_j, l_j, m_j) \in \Gamma$ and $y, z \in O^*$. At the same time a node h and an edge $\{i, h\}$ are added to the graph γ' in $\mathcal{A}'$. The cell σ_i, is instead replaced in $\mathcal{A}'$ by a cell σ'_i where the value b_i is decreased by the number of bonds formed by this cell in this stage of self-assembly.
2. For each cell σ_i in $\mathcal{A}$, with $1 \leq i \leq n$, which no new cells can be linked to, we add to the assembly $\mathcal{A}'$ a cell $\sigma''_i = \sigma_i$ and a node i in the graph γ'.
3. For each edge $\{i, j\}$ in the graph γ from the assembly $\mathcal{A}$, with $1 \leq i \neq j \leq n$, we add the same edge $\{i, j\}$ to the graph γ'.

4. Finally, we renumber the cells, the nodes, and the edges in $\mathcal{A}'$ in an one-to-one manner with values from $\{1, 2, \ldots, n'\}$, with $n' \geq n \geq 1$ the current number of cells in $\mathcal{A}'$, in such a way to preserve the correspondence between cells, nodes and edges.

Parallel single-point self-assembly has a limited capacity of forming complex graph structures as it works under the assumption that, during a self-assembly stage, a new cell can form only one single bond with a specific cell in the current assembly. Moreover, from that moment on, this new structure will never be altered apart for the introduction of more new cells. Specifically, if we denote by $\Rightarrow_{\mathcal{P}}$ a derivation step in a P system $\mathcal{P}$ which uses single-point self-assembly, and by $\Rightarrow_{\mathcal{P}}^{+}$ its transitive closure, then it is easy to see that the following lemma holds.

Lemma 1. *Let $\mathcal{P} = (O, L, \Gamma, \sigma, R, B)$ be a self-assembly P system as specified in Definition 1. Let $\mathcal{S} = (\sigma_1, (\{1\}, \emptyset))$, with $\sigma_1 = \sigma$, be the initial assembly of cells in $\mathcal{P}$. For each assembly of cell $\mathcal{A}$ such that $\mathcal{S} \Rightarrow^{+} \mathcal{A}$, the graph γ defining the structure of the assembly A is a tree.*

This result is a consequence of the fact that the number of nodes and the number of edges added to the current graph, during a stage of self-assembly, is always equal to the number of edges added at the same time. Moreover, each new node introduced in such a stage results connected to the pre-existing graph by means of at least one edge. This means that, during the self-assembly stage, no new cycles can be created inside the graph defining the structure of the new assembly of cells. Therefore, if we start with an acyclic graph, this property of not containing cycle will be preserved during each step of derivation of a P system that uses parallel single-point self assembly. Thus, in the case of the seed assembly, what we obtain is always a tree which can be thought as being rooted in the seed cell.

3.2 Parallel Multi-point Self-assembly

We pass now to consider a self-assembly model where "growing points" for the current graph structure are represented by "neighbourhood" of cells in the current assembly. This means a new cell, which is added to the current assembly during a self-assembly stage, can form more than one bond with many different pre-existing cells but all these cells must be neighbouring cells of a certain cell being itself connected to the same new cell; that is, a new cell can only be connected to cells which are all reachable in one step from a given starting point. In particular, each new cell is now going to form as many bond as possible with respect to a particular choice of bond making rules, the distribution of objects inside the cells from the chosen neighbourhood, and the number of bonds that can be effectively formed with these cells. Moreover, new cell must result connected to the pre-existing graph by means of at least one edge and bonds cannot be formed between new cells added during the same self-assembly stage. This self-assembly model is called parallel multi-point self-assembly as, in each self-assembly stage, many new cells can be added in parallel at the same by forming many new bonds with many different cells already present in the current assembly.

More formally, let $\mathcal{P} = (O, L, \Gamma, \sigma, R, B)$ be a self-assembly P system as specified in Definition 1 and let $\mathcal{A} = (\sigma_1, \sigma_2, \ldots, \sigma_n, \gamma)$, for some $n \geq 1$, be an assembly of cells in $\mathcal{P}$ as specified in Definition 2. We write $A \overset{sa}{\Longrightarrow}_{\mathcal{P}} \mathcal{A}'$, and we say A' is derived from $\mathcal{A}$ by parallel multi-point self-assembly, if and only if $\mathcal{A}'$ is an assembly of cells in $\mathcal{P}$ which is obtained from $\mathcal{A}$ in the following way:

1. Given a clone σ' of a cell template in Γ, we non-deterministically select a neighbourhood of cells η_i, for some $1 \leq i \leq n$, such that η_i contains cells that are directly linked to cell σ_i and all of them including cell σ_i can form a bond with the clone σ'; this set must be maximal with respect to a particular assignment of the objects contained in all these cells to the bond-making rules in B in the sense that no other cell directly linked with σ_i can form a bond with the clone σ'. The clone σ' is then added to the assembly σ' together with a corresponding node and an edge between this new node and each pre-existing node corresponding to a cell in η_i. At the same time, for each cell involved in this bond making process, we update the bound on the number of bonds that can be formed by that cell so to keep track of the bonds that have been just formed.
2. The previous operation of adding a clone is performed in a non-deterministic maximal parallel manner by inserting into the new assembly $\mathcal{A}'$ as many clones as possible according to the current distribution of objects inside the cells and to a particular assignments of these objects to the bond-making rules in B. In particular the following conditions must be satisfied: the total number of bonds formed by a cell during this self-assembly stage is less or equal to the current number of bonds that can be formed by that cell, if two cells compete for the same occurrence of the same multiset placed inside the same cell then, only the cell forming the greater number of bonds is effectively inserted in $\mathcal{A}'$.
3. For each cell σ_i in $\mathcal{A}$, with $1 \leq i \leq n$, which no new cells can be linked to, we add to the assembly $\mathcal{A}'$ a cell $\sigma_i'' = \sigma_i$ and a node i in the graph γ'.
4. For each edge $\{i, j\}$ in the graph γ from the assembly $\mathcal{A}$, with $1 \leq i \neq j \leq n$, we add the same edge $\{i, j\}$ to the graph γ'.

Now it is easy to see that multi-point self-assembly is less restrictive than single-point self-assembly and it can lead to the formation of cyclic graph. Specifically, if we denote by $\Rightarrow_{\mathcal{P}}$ a derivation step in a P system $\mathcal{P}$ which uses multi-point self-assembly, and its transitive closure by $\Rightarrow_{\mathcal{P}}^{+}$, then the following lemma holds.

Lemma 2. *Let $\mathcal{P} = (O, L, \Gamma, \sigma, R, B)$ be a self-assembly P system as specified in Definition 1. Let $\mathcal{S} = (\sigma_1, (\{1\}, \emptyset))$, with $\sigma_1 = \sigma$, be the initial assembly of cells in $\mathcal{P}$. For each assembly of cell $\mathcal{A}$ such that $\mathcal{S} \Rightarrow^{+} \mathcal{A}$, the graph γ defining the structure of the assembly A may contain some cycles.*

Proof. Consider the self-assembly P system $\mathcal{P} = (\{a, b\}, \{\$, \#\}, \Gamma, \sigma, \emptyset, B)$ with: $\Gamma = \{(a, \$, 3), (b, \$, 3), (ab, \#, 3)\}$, $\sigma = (a, \$, 2)$, and $B = \{(\$, a; b, \$), (\$, a; b, \#)\} \cup$ $\cup \{(\$, b; a, \#), (\#, b; b, \$), (\#, a; a, \$)\}$.

The seed assembly of $\mathcal{P}$ is the assembly $\mathcal{S} = ((a, \$, 2)_1, (\{1\}, \emptyset))$. In the first step of derivation, we can add to cell 1 either a cell $(b, \$, 3)$ by using the bond-making rule $(\$, a; b, \$)$, or a cell $(ab, \#, 3)$ by using the bond-making rule $(\$, a; b, \#)$. Let us suppose the first bond-making rule is used in the first step of derivation so to obtain a new assembly of cells $\mathcal{A}_1$ such that:

$$\mathcal{A}_1 = ((a, \$, 1)_1, (b, \$, 2)_2, (\{1, 2\}, \{\{1, 2\}\})).$$

Now, given the assembly $\mathcal{A}_1$, we can add to cell 1 either a cell $(b, \$, 3)$ by using the bond-making rule $(\$, a; b, \$)$, or a cell $(ab, \#, 3)$ by using the bond-making rule $(\$, a; b, \#)$; we can add to cell 2 a cell $(ab, \#, 3)$ by using the bond-making rule $(\$, b; a, \#)$. Moreover, as we are considering multi-point self-assembly, each clone of the cell $(ab, \#, 3)$ can potentially form two bonds at the same time by using the rule $(\$, a; b, \#)$ in parallel with the $(\$, b; a, \#)$. Specifically, a copy of this cell is the unique cell which is added in the next step of derivation, as this is the cell that can form the greatest number of bonds by preventing any other cells from forming any other bond. In this way we immediately obtain an assembly of cells where the corresponding graph contains the cycle $\{1, 2\}$, $\{2, 3\}$, $\{3, 1\}$. $\square$

4 Uniquely Self-assembly a Graph

In this section we deal with the problem of defining a self-assembly P system which is able to produce as result of its derivations a given target graph; this graph is supposed to be connected and with no loop edges (i.e., edges linking a node with itself). In particular, we want this graph to be uniquely generated by the defined P system, that is, all the possible derivations must always produce, after a finite number of steps, a similar assembly of cells where the corresponding graph is isomorphic to the given target graph. As well as this, all these derivations must "halt" immediately after having produced this particular assembly of cells; halting, in this context, means the self-assembly P system produces an assembly of cells where no more transformation or communication rules can be applied to the objects placed inside the cells and no more bond-making rules can be applied to the current graph structure.

More precisely, let $\Rightarrow_{\mathcal{P}}$ be the notion of derivation step in a self-assembly P system $\mathcal{P}$ as specified in the previous section and let $\Rightarrow^{+}_{\mathcal{P}}$ be its transitive closure. Moreover, given an assembly of cell $\mathcal{A}$, we denote by $\mathcal{A}.\gamma$ the graph defining the structure of the assembly $\mathcal{A}$. Then, we say $\mathcal{P}$ uniquely generates a graph G if and only if:

– there exists an assembly of cells $\mathcal{A}$ such that $\mathcal{S} \Rightarrow^{+}_{\mathcal{P}} \mathcal{A}$ and $\mathcal{A}.\gamma \approx G$;
– for all assembly of cells $\mathcal{A}$ with $\mathcal{A}.\gamma \not\approx G$, if $\mathcal{S} \Rightarrow^{+}_{\mathcal{P}} \mathcal{A}$ then, there exists $\mathcal{A}'$ with $\mathcal{A}'.\gamma \approx G$ such that $\mathcal{A} \Rightarrow^{+}_{\mathcal{P}} \mathcal{A}'$;

– for all assembly of cells $\mathcal{A}$ with $\mathcal{A}.\gamma \approx G$, if $\mathcal{S} \Rightarrow^{+}_{\mathcal{P}} \mathcal{A}$ then, there does not exist $\mathcal{A}'$ such that $\mathcal{A} \Rightarrow^{+}_{\mathcal{P}} \mathcal{A}'$.

This means all the derivations in the self-assembly P system $\mathcal{P}$ halt by always producing a graph structure isomorphic to the graph G; if that is the case then we write $\mathcal{P} \vdash^{u} G$.

Now, we can precisely state the two main problems related to the unique self-assembly of a graph by means of self-assembly P systems.

Problem 1. For every connected graph $G = (\{1, 2 \ldots, n\}, E)$, with E being included in the set $\{\{i, j\} \mid 1 \leq i \neq j \leq n\}$, does exist a self-assembly P system such that $\mathcal{P} \vdash^{u} G$?

It is obvious that the answer and the solution to this problem highly depend on the particular self-assembly model chosen. Specifically, in the case of single-point self-assembly, Lemma 1 provides a negative answer to Problem 1 whereas, in the case of multi-point self-assembly, Problem 1 still remains open.

Let $G = (\{1, 2 \ldots, n\}, E)$, with $E \subseteq \{\{i, j\} \mid 1 \leq i \neq j \leq n\}$, be a connected graph. If a self-assembly P system with at most o objects, at most l labels, at most c cell templates, at most b bond making rules and at most r transformation and communication rules per each cell label, exists such that $\mathcal{P} \vdash^{u} G$ then, it is denoted by $\mathcal{P}_{o,l,c,b,r}$. We say $\mathcal{P}_{o,l,c,b,r}$ is optimal if there does not exist $\mathcal{P}'_{o',l',c',b',r'}$ uniquely generating G with at least one of these primed parameters being less than the corresponding one in the first self-assembly P systems and the others having the same values.

Problem 2. Given a connected graph $G = (\{1, 2 \ldots, n\}, E)$, with E being included in the set $\{\{i, j\} \mid 1 \leq i \neq j \leq n\}$. If a self-assembly P system $\mathcal{P}_{o,l,c,b,r}$ exists such that $\mathcal{P}_{o,l,c,b,r} \vdash^{u} G$ then, is $\mathcal{P}_{o,l,c,b,r}$ optimal?

The following lemma states that a complete binary tree with depth $d \geq 0$ can be uniquely generated by a P system $\mathcal{P}_{2d+2,2,2,2(d-1),d+1}$. We do not know whether this is optimal or not but we claim it is not.

Lemma 3. *Let T be a complete binary tree with depth $d \geq 0$. We can always construct a self-assembly P systems that uniquely generate T by using: $2d + 2$ different objects, 2 different labels, 2 different cell templates, $2(d - 1)$ different bond-making rules, and at most $d + 1$ transformation and communication rules per cell.*

Proof. Let T be a complete binary tree with depth $d \geq 0$. We construct a self-assembly P system $\mathcal{P}$ that uses parallel single-point self-assembly and such that:

$$\mathcal{P} = (O, L, \Gamma, \sigma, R, B),$$

with:

– $O = \{a, b\} \cup \{\$_p, \$'_p \mid 0 \leq p \leq d\}$;
– $K = \{c_1, c_2\}$;

– $\Gamma = \{(a, c_1, 3), (a, c_2, 3)\}$;
– $\sigma = (b, c_1, 2)$;
– $R = \{[\,\$_p \rightarrow \$'_{p+1}\$'_{p+1}\,]_{c_1} \,|\, 0 \leq p \leq d-1\}$
$\cup\{[\,\$'_p \rightarrow \$_{p+1}\$_{p+1}\,]_{c_2} \,|\, 0 \leq p \leq d-1\}$
$\cup\{\,[a; \$_p\$_p, in]_{c_1} \,|\, 0 \leq p \leq d-1\} \cup \{\,[a; \$'_p\$'_p, in]_{c_1} \,|\, 0 \leq p \leq d-1\}$
$\cup\{\,[\,b \rightarrow \$_0\$_0\,]_{c_1}\,\}$;
– $B = \{\,(c_1, \$_p; a, c_2) \,|\, 0 \leq p \leq d-1\}$
$\cup\{\,(c_2, \$'_p; a, c_1) \,|\, 0 \leq p \leq d-1\}$.

The seed assembly of $\mathcal{P}$ is the assembly $\mathcal{S} = ((b, c_1, 2)_1, (\{1\}, \emptyset))$. In the first stage of evolution-communication, we apply inside the seed cell the rule $[\,b \rightarrow \$_0\$_0\,]_{c_1}$ in order to produce inside this cell two copies of the object $\$_0$. Then, in the self-assembly stage, we connect the seed cell with two new cells $\sigma_2 = (a, c_2, 3)$, $\sigma_3 = (a, c_2, 3)$ by using the bond-making rule $(c_1, \$_0; a, c_2)$ twice. Next, inside the seed cell, we apply the rule $[\,\$_0 \rightarrow \$'_1\$'_1\,]_{c_1}$ in order to produce four copies of the object $\$'_1$. Both cell σ_3 and σ_2, in the next step of evolution-communication, receive two copies of this object by using, inside both of them, a rule $[a; \$'_p\$'_p, in]_{c_1}$; these objects are then used both cell σ_3 and σ_2 to attract two new cells labeled by c_1 by using the bond making rule $(c_2, \$_1; a, c_2)$ four times. This process can be then iterated for each level $p \leq d-1$ by adding to the current structure, during each step, exactly 2^p cells; the process halts immediately after having produced inside the new cells objects of the form $\$_d$, which no rules can be applied to these cells. This mean the tree is correctly generated in d steps by starting from the root and adding the leaves in the last step of derivation. □

Notice that the same construction can be applied to any complete n-ary tree of depth $d \geq 0$ by just augmenting the number of objects $\$_p$ that are produced inside the cells placed at level $0 \leq p \leq d-1$.

5 Conclusions

Self-assembly P systems are devices that use some of the features of population P systems [2] to progressively increase a graph structure by forming new bonds between the existing cells and some new cells which are brought into the system step by step. Specifically, with respect to [2], bond-making rules can be used only to increase the number of links in the graph defining the structure of the system and they can never be used to alter the structure of an already formed assembly of cells. As well as this, bond-making rules are restricted to be applied locally in correspondence of a certain neighbourhood of cells where self-assembly can take place between cells that are supposed to be "attracted" in that particular vicinity. Moreover, self-assembly P systems use transformation and communication rules to continuously update the internal configuration of the cells and vary the distribution of objects between various cells in the system. The problem of defining in a self-assembly P systems that, for a given self-assembly model, are able to generate any graph of any form remains an open. In fact, here we have only been able to show how to generate a complete binary tree in an "efficient" way by means of a P system that uses a “limited” number of resources. In particular, this is achieved by exploiting the features of transformation and communication rules which allow cells to update this internal configuration and exchange objects with its neighbouring cells. In general, many other features of P systems may be introduced in self-assembly P systems so to have systems consisting in a finite number of cells which are able to re-organise themselves by means of local and limited interactions in order to produce a desired shape or pattern.

Considering the high sensitivity of the final shape of the graph with respect to the rules, the study of self-assembly P systems would certainly benefit from a simulation tool. This kind of tool can be used to ”play” several variations and to check the result on some cases. We consider the use of the MGS [3] programming language to develop such simulator. As a matter of fact, MGS provides both multiset rewriting and graph rewriting as well as the ability to create and transform, by rewriting rules, a graph of multisets.

Acknowledgements

The research of F.B. and M.G. has been supported by the Engineering and Physical Sciences Research Council of United Kingdom (EPSRC), grant GR/R84221/01. The research of N.K. has been supported by EPSRC grants EP/D021847/1 and EP/D023343/1. The authors wish to thank the anonymous referees for their comments that allow us to improve the content of this paper.

References

1. Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight, F., T., Nagpal, R., Rauch, E., Sussman, J., G., Weiss, R.: Amorphous Computing. *Communication of the ACM* 43 (2000) 74-82
2. Bernardini, F., Gheorghe, M.: Population P Systems. *Journal of Universal Computer Science* 20 (2002) 509-539
3. Giavitto,J.-L., Michel, O.: The Topological Structure of Membrane Computing. *Fundamenta Informaticae* 49 (2002) 107-129
4. Klavins, E.: Automatic Synthesis of Controllers for Assembly and Formation Forming. In: *Proceedings of the International Conference on Robotics and Automation.* (2002)
5. Krasnogor, N., Gustafson, S.: A Family of Conceptual Problems in the Automated Design of Systems Self-Assembly. In: *Proceedings of the 2nd International Conference on the Foundations of Nanoscience: Self-Assembled Architectures and Devices.* (2005)
6. Păun, Gh.: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin Heidelberg New York (2002)
7. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages,* Vol. 1-3 Springer-Verlag, Berlin Heidelberg New York (1997)
8. Winfree, E.: *Algorithmic Self-Assembly of DNA.* PhD Thesis, California Institute of Technology (1998).

A Web-Based P Systems Simulator and Its Parallelization

Cosmin Bonchiş[1], Gabriel Ciobanu[1,3], Cornel Izbaşa[1], and Dana Petcu[1,2]

[1] Research Institute "e-Austria" Timişoara, Romania
[2] Western University of Timişoara, Department of Computer Science
[3] Romanian Academy, Institute of Computer Science, Iaşi

Abstract. In this paper we present WebPS, an open-source web-enabled simulator for P systems, and a P accelerator for parallelization of the existing sequential simulators. The simulator is based on CLIPS, and it is already available as a web application. The P accelerator is used to parallelize the existing sequential simulators of the P systems. We exemplify this tool by using a simple CLIPS simulator. The speedup and the efficiency of the resulting parallel implementation are surprisingly close to the ideal ones. Combining these two ingredients, we get a complex and ready-to-use parallel simulator for P systems; no installation are required, no previous knowledge of operating systems, parallel programming or specific software.

1 Introduction

Transition P systems and deterministic P systems with active membranes (see [7]) are simulated in various programming languages, and some of them are used to solve NP-complete problems as SAT, Subset Sum, Knapsack, and partition problems. P systems with active membranes, input membrane and external output are simulated in CLIPS, and used to solve NP-complete problems (see [8]). New variants of these simulators provide symport-antiport rules, and catalysts. A previous simulator written in Visual C++ for P systems with active membranes and catalytic P systems is presented in [2]. It provides a graphical simulator, interactive definition, visualization of a defined membrane system, a scalable graphical representation of the computation, and step-by-step observations of the membrane system behaviour. A parallel and cluster implementation for transition P systems in C++ and MPI is presented in [2]. The rules are implemented as threads. At the initialization phase, one thread is created for each rule. Since each rule is modelled as a separate thread, it should have the ability to decide its own applicability in a particular round. In order to detect if the P system halts, each membrane must inform the other membranes about its inactivity. It can do so by sending messages to others, and by using a termination detection algorithm.

We present here a new simulator, and its main feature is given by the availability as a Web application. This aspect is actually emphasized by its name: WebPS. As any Web application, WebPS does not require an installation. It can be used from any machine anywhere in the world, without any previous preparation. A simple and easy to use interface allows the user to supply an XML input both as text and a file. A friendly way of describing P systems is given by an interactive JavaScript-based P system designer. The interface provides a high degree of (re)usability during the development and simulation of the P systems. The initial screen offers an example, and the user may find useful documentation about the XML schema, the rules, and the query language. The query language helps the user to select the output of the simulation.

This simulator is based on CLIPS. The previous CLIPS implementations represent P systems rules by CLIPS facts; we get a significantly faster execution by using CLIPS rules to implement P systems rules. In [8], the P system data are inserted directly into code, and so they cannot be easily modified, as opposed to our implementation where a P system is an input data, providing a lot of flexibility. P systems as input data are described by XML documents; this fact provides a standard method to access information, making it easier to use, modify, transmit and display. XML is readable and understandable, it expresses metadata easily in a portable format, just because many applications can process XML on many existent platforms. Moreover, by using XML, it becomes easy to define new features and properties of P systems. XML allows an automated document validation, restricting wrong input data, and warning the user before execution with respect to the possible errors in their P systems description. From a user point of view, all these features facilitate an efficient description and reconfiguration of the P systems.

The simulator is *open-source*, actually *free software*, as it is being offered at `http://psystems.ieat.ro` under the *GNU General Public License*. This allows anyone to contribute with enhancements and error corrections to the code, and possibly develop new interfaces for the C and CLIPS level APIs. These interfaces can be local (graphical or command-line), or yet other Web-based ones.

Section 2 presents the WebPS structure, and argue what are its advantages over the previous P simulators. Several new and interesting examples implemented in the WebPS system are described in Section 3. In Section 4 we discuss on the parallelization of sequential simulators, and present Jess as a starting tool in building a wrapper for parallel systems. Then we describe our P accelerator as a parallel version of Jess, able to work on homogeneous clusters of workstations. We provide some implementation details, and measure the efficiency of our P accelerator by applying it to a simple CLIPS simulator solving a difficult problem. We present the results of our experiments showing a significant reduction of the running time. Conclusion, further work and references end the paper.

2 WebPS Software Architecture

The software structure of the simulator has three distinct levels. The inner level is the CLIPS level. This level integrates part of the knowledge about the theoretical description of P systems, the result being a library of CLIPS functions, meta-rules and templates. Since CLIPS is easily embeddable in C (as its name "C Language Integrated Production System" suggests), we can control the CLIPS level from a C program, and we include some example to illustrate how this is done. The C level is strengthened by introducing a C library for modelling and simulating P systems based on the CLIPS library. The Web application level offers a user-friendly interface to the simulator, and can become, after the addition of debugging and visualization features a powerful P system development tool.

2.1 CLIPS Level

This is the core level of the simulator. We address here some crucial issues, namely how we implement in a sequential context the *maximally parallel* and *nondeterministic execution* of P systems.

Maximally Parallel Execution. The maximally parallel execution requirement relies on constraining our simulation cycle to the following distinct steps:

1. **React** step: where the activated reaction rules are sequentially executed.
2. **Spawn** step: where the new objects created by rules inside their membranes are asserted as object facts; they become visible for a future **React** step.
3. **Communication** step: where the objects injected or ejected by communication rules in different membranes are asserted as objects facts.
4. **Divide** step: it handles possible divisions processes of membranes.
5. **Dissolve** step: it handles dissolving processes of membranes.

By constantly recording the state of the P system after each **React**, **Spawn** and **Communication** steps, we can get a trace of an execution.

Nondeterministic Execution. CLIPS uses RETE algorithm to process rules; RETE is a very efficient mechanism for solving many-to-many matching problem [5]. The nondeterministic execution requirement is fulfilled by the CLIPS *random* mechanism. We have looked closely at the random strategy, and we found it makes the same choice for the same configurations in different executions. By calling the random function of CLIPS, the random mechanism uses the random number generator. The failure is related to the improper seeding of the random number generator. We have corrected this error, and properly seed the random number generator by using */dev/urandom*, the entropy gathering device on GNU/Linux systems. This aspect can be also useful for other CLIPS implementations possibly affected by the same failure.

Data Representation. An important choice regarding an implementation is given by data representation. We decide to represent P system objects and membrane structure as CLIPS facts (with set-fact-duplication option of CLIPS set to on), and the P systems reaction rules as CLIPS rules. This contrasts the previous implementations which have represented reaction rules as CLIPS facts; while their choice might allow general meta-rules for execution, we get a flexible and efficient framework by representing reaction rules as CLIPS rules. The efficiency is gained by making direct use of the CLIPS pattern-matching mechanism, and rule activation capabilities. This choice is also confirmed by the efficiency of the dissolve and divide operations which imply a lot of moving and copying. Initially we think to represent membranes as modules, but later we see that this representation decreases the efficiency and flexibility of the whole system.

It is useful to note that the simulator supports divide, promoters/inhibitors, and symport/antiport rules for membranes. For example, the P system transition rule a+b→c+d(2)+e(0) with priority 11 from membrane 1 is converted into the following CLIPS rule:

```
(defrule MAIN::1_a+b->c+d[2]+e[0]
  (declare (salience 11))
  (do (what react))
  (or (parent-child (parent 1) (child 2))
      (parent-child (parent 2) (child 1)))
  (or (parent-child (parent 1) (child 0))
      (parent-child (parent 0) (child 1)))
  ?a-0 <- (obj (name a) (membrane 1))
  ?b-1 <- (obj (name b) (membrane 1))
  =>
  (assert (newobj (name c) (membrane 1)))
  (assert (inject (name d) (membrane 2)))
  (assert (inject (name e) (membrane 0)))
  (retract ?a-0) (retract ?b-1))
```

The membrane structure is reflected by the parent-child facts. An object ***a*** of membrane ***1*** is represented as a CLIPS fact on line 8: *(obj (name a) (membrane 1))*. While the reactants ***a*** and ***b*** are consumed, the new objects ***c***, ***d***, and ***e*** are created during the **Spawn** and **Communication** steps, respectively. The rule priorities are mapped directly to CLIPS rules salience values, and therefore are restricted to integer values in the interval [-10000, 10000].

2.2 C Level

In the C level we use CLIPS API, although we plan to develop a complete library that encapsulates the C-CLIPS interface, namely a C library which wraps nicely around the CLIPS one.

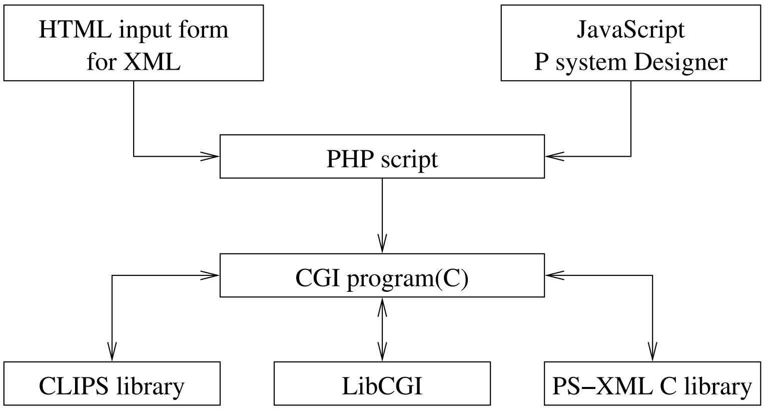
We represent a P system using XML, and we define an XML schema for this kind of document. A special library is developed to handle XML parsing of the input for the CLIPS part of the simulator. In the following example we describe a P system using our XML schema. This example system presents a P system for multiplication of the number of ***a*** objects in membrane ***1*** with the number of ***b*** objects in membrane ***0***, the result being the number of ***d*** objects in membrane ***0***.

```
<?xml version="1.0"?>
  <psystem>
    <membrane name="0">
      <object name="b" count="3" />
      <rule body="b+v->e+v+d" priority="1" />
      <rule body="e+u->b+u+d" priority="1" />
      <rule body="v->u(1)" />
```

```
        <rule body="u->v(1)" />
          <membrane name="1">
            <object name="a" count="4" />
            <object name="v" count="1" />
            <rule body="a+v->v(0)" />
            <rule body="a+u->u(0)" />
          </membrane>
      </membrane>
      <query text="count of (objects from 0 where (objects d))" />
    </psystem>
```

2.3 Web Level

The Web level of the simulator allows to choose between a user-friendly P system designer written in JavaScript and a traditional HTML input form transmitting an XML description by uploading a file or by editing. Aside from the XML P system description editing, the user can specify a number of executions of the P system. Our JavaScript P system Designer aims to facilitate the description of the P system without requiring the user to write XML, but generating it based on the user's interaction with a dynamic interface.



After the user introduces a XML description, it is transmitted to a PHP script which does some further processing, and sent then to a CGI program written in C. The C program uses a specific P system XML library called PS-XML, as well as LibCGI and CLIPS library in order to simulate the evolution of the P system. Finally it returns the results to the user. It is possible to select various information provided as results, and in order to help the user to select the desired information we define a query language called PsQL.

PsQL (P systems Query Language). We define PsQL as an SQL-like language for querying the state of a P system. We developed a CLIPS library for parsing and interpreting this language. At the Web level, the queries can be included in the XML input; these queries are activated after the execution of the specified P system. If it does not exist any query, the P system is simulated, but no output is generated. At the CLIPS level it is possible to specify queries for the P system in a dynamic manner, not just before starting the simulation. At the syntactic level, PsQL is a Lisp-like language, and it is supported by a small CLIPS library of list-handling functions.

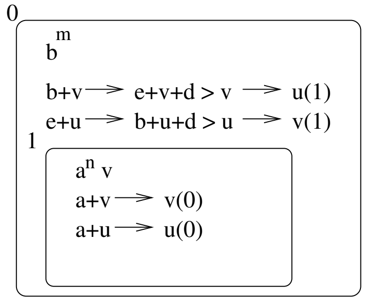
The Backus-Naur description of PsQL is presented in the following lines:

```
<query> ::= <expression>      |
            <count-query>     |
            <membranes-query> |
            <objects-query>
<count-query> ::= "(" count-of <objects-query> |
                               <membranes-query> ")"
<objects-query> ::= "(" objects-from <membranes-spec>
                                [ where <where-spec> ] ")"
<membranes-query> ::= "(" membranes-from <membranes-spec>
                                [ where <where-spec> ] ")"
```

The full description is available at `http://twiki.ieat.ro/twiki/bin/view/Institut/PSystemsQueryLanguage`. We plan to extend PsQL with trace facilities; having queries on the possible traces during an execution represents a step towards an automated verification of the P systems.

3 Examples

The first example is a P system that computes the **multiplication** of two natural numbers. The following figure describes graphically the P system.



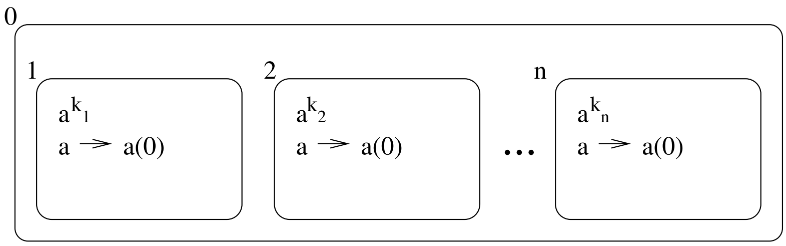
As inputs we consider the number of ***a*** objects in membrane ***1***, and the number of ***b*** objects in membrane ***0***. The result is given by the number of ***d*** objects in membrane ***0***.

This P system differs from other similar ones by that it does not have exponential space complexity, and does not require active membranes. As a particular case, it would be quite easy to compute n^2 by just placing the same number n of ***a*** and ***b*** objects in its membranes.

Another interesting feature is that it may continue computing the multiplication after reaching a certain result. Thus if initially there are m ***b*** objects and

n ***a*** objects, the system evolves and reaches a state with $n \cdot m$ ***d*** objects in membrane ***0***. If the user wish to continue in order to compute $(n+k)\cdot m$, it is enough to inject k ***a*** objects in membrane ***1*** at the current state, and the computation can go on. Therefore this example emphasizes a certain degree of re-usability.

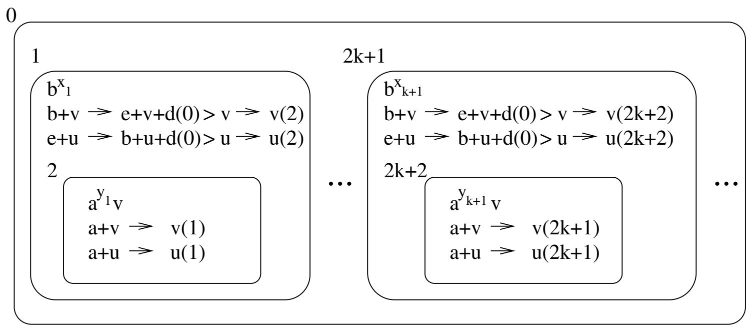
Recursive Sum: The P system described in the next picture computes the recursive sum $\sum_{i=1}^{n} k_i$.



The numbers of ***a*** objects in the membranes 1...n are the addition arguments, and the result of the computation is the number of ***a*** objects in membrane 0. The PsQL query to determine this result is: `(count of (objects from 0))`.

While this example is rather trivial, it illustrates the expressiveness of the query language (PsQL). Using PsQL queries, it is not necessary to apply the rules and execute the specified P system. Given the initial multiset, the same recursive sum is obtained by using the following query: `(count of (objects from (membranes from 0)))`.

Dot Product of Two Vectors: Combining the previous two examples, we can compute the dot (scalar) product $x \cdot y$ of two vectors $x, y \in \mathbb{N}^m$, where $m \in \mathbb{N}$. Let us denote the components of the vectors by x_i and y_i, respectively; these components are given by the number of ***b*** and ***a*** in the membranes labelled by $2i$ and $2i+1$, respectively. Then $x \cdot y$ is given by the number of ***d*** objects obtained in membrane 0 after the P system halts. This P system is described graphically as:


0
1
b^x1
b+v → e+v+d(0) > v → v(2)
e+u → b+u+d(0) > u → u(2)
2
a^y1 v
a+v → v(1)
a+u → u(1)
...
2k+1
b^xk+1
b+v → e+v+d(0) > v → v(2k+2)
e+u → b+u+d(0) > u → u(2k+2)
2k+2
a^yk+1 v
a+v → v(2k+1)
a+u → u(2k+1)
...

where $k = \overline{0, m-1}$. The PsQL query for retrieving the result is: `(count of (objects from 0))`.

4 On Parallelization of Sequential Simulators

There will always be a need for exploiting parallelism in computing, such that many difficult problems can be executed on parallel architectures. There is also a need to free the programmers from thinking about the parallelization of existing sequential programs. Automatic parallelization of code has been an active research topic in scientific computing for some decades. For a long period of time, the kind of attempts achieve little of the potential benefit of parallel computing. Recent contributions improve the performance, and efficient parallel codes have been created automatically to solve problems in various fields.

We refer to the sequential simulators of the P systems, particularly to those implemented in CLIPS [8]. These sequential simulators have both didactic and scientific values. However the P systems are inherently parallel and, in many variants, they also exhibit an intrinsic non-determinism, hard to be captured by sequential computers. By simulating parallelism and nondeterminism on a sequential machine, one can lose the real power of parallelism and attractiveness of P system computing. Therefore the simulations on multiple processors represent a particularly interesting subject. Currently the only known parallel and cluster implementation for P systems is presented in [2], using C++ and MPI.

4.1 Jess: Java Expert System Shell

Jess is the abbreviation for Java Expert System Shell; it is a rule-based programming environment for Java platforms; it is available at `http://herzberg.ca.sandia.gov/jess/`. More information about Jess can be found in [6]. Jess uses the RETE algorithm [5] for rules, an efficient mechanism for solving the difficult many-to-many matching problem. Our strategy is to build a parallel production systems with a high degree of flexibility, namely to construct a wrapper for the parallel system which allows cooperation between its instances. From this point of view, Jess has been considered mainly due to its flexibility (communication via sockets, ability to create Java objects, and call Java methods) and its compatibility with the C Language Interface Production System CLIPS; CLIPS is available at `http://www.ghg.net/clips/CLIPS.html`, where it is presented as a tool for building expert systems. Jess is also selected because of its active development and support, tight interaction with Java programs, and expressiveness.

The Jess rule language includes elements not present in many other production systems, such as arbitrary combinations of Boolean conjunctions and disjunctions. Its scripting language is powerful enough to generate full applications entirely within the Jess system. Jess is also a powerful Java scripting environment, from which one can create Java objects, and call Java methods without compiling any Java code. The core Jess language is fully compatible with CLIPS (Jess scripts are valid CLIPS scripts, and vice-versa). Jess adds many features to CLIPS, including backwards chaining, working memory queries, and the ability to manipulate and directly reason about Java objects.

We think to consider existing implementations, and improve them by using multiple computing units, leading to faster P system simulators than the current available ones. The proposed architecture for a such system is based on an accelerator, actually a wrapper allowing the cooperation between several instances of a program running on different computers, in order to speedup the current P system simulators.

We intend to apply our P accelerators to WebPS. As a first step, we consider a simpler CLIPS simulator [8]. The set of rules and the configurations in each step of the evolution are expressed as facts in a knowledge base. We first show that a splitting technique of the membranes in several Java threads running embedded Jess can lead to a faster simulation. Then we use the P accelerator to further speedup the simulation.

4.2 P Accelerators

Having in mind the parallel evolutions of its membranes based on different rules, we build a parallel distributed memory version of a production system based on task parallelism. The target architecture is a homogeneous cluster of workstations. Building a Jess application based on socket communication facilities is considered to be difficult and can distract the user from its main aim. Therefore a middleware is needed. We adopt a modular scheme composed by Jess instances, Connectors, and Messengers.

A first component of the P accelerator is called Connector, and consists of a Java code. Each Jess instance has one corresponding Connector. A Jess instance (acting as a client) contacts its Connector (acting as a server) via a socket. As soon as such a connection is established, the Connector interprets the Jess special incoming requests for communications, and controls other Jess instances (namely send or receive information, launch or kill other instances). An information in transit is a string containing a command written in Jess language.

Another component of the P accelerator is represented by the Messengers. Each Messenger is associated with one Connector and its purpose is to execute the commands received by Connector, and to communicate with various Messengers associated with the other Jess instances. We add new commands to Jess, assuring a valid message passing interface. More details about the set of commands, and some other aspects related to P accelerators can be found in [4].

4.3 Implementation Details

A Connector uses the standard Java ServerSockets methods. The current Messenger is written in Java and JPVM, a Java implementation of Parallel Virtual Machine selected due to its ability to dynamically create and destroy tasks, a useful ability when simulating the division and dissolution of membranes. JPVM is available at `http://www.cs.virginia.edu/~ajf2j/jpvm.html`. Adopting a PVM variant, the user is absolved of the duties to nominate the hosts were the Jess instances are running, as well as to treat sequentially the incoming messages (as in the case of a socket connection), or to check the status of the machines on which the Jess instances are running.

Regarding CLIPS, we should mention that a production system consists in a working memory, a set of rules and an inference engine. The working memory is a global database of data representing the system state. A rule is a condition-action pair. The inference engine is based on a three-phase cyclic execution model of condition evaluations (matching), conflict-resolution and action firing. Firing can add, delete or modify elements in the working memory. An instantiation is a rule with a set of working memory elements. In a sequential environment, conflict-resolution selects for firing a certain instantiation from the set of all instantiations. In a parallel environment, multiple instantiations can be selected for firing simultaneously. Considering a high degree of parallelism, the time performance can be improved.

4.4 Experiments

The computational power of a P system can be used not for solving small problems for which we have already faster algorithms, but for large and difficult problems. The main goal of our experiments is to measure the efficiency of P accelerator in a cluster environment when it is applied to a particular problem. Our experiments use the P system with active membrane described in [8] to solve the validity problem: given a Boolean formula in conjunctive normal form, to determine whether or not it is a tautology. If we consider the problem input in the form $\wedge_{i=1}^{m} \vee_{j=1}^{k_i} x_{ij}$ where $x_{ij} \in \{X_1, \ldots, X_n, \overline{X}_1, \ldots, \overline{X}_n\}$ the P system solves the NP-complete problem of validity in $5n + 2m + 4$ steps. The number of membranes increases by division from only three initial membranes to $2^n + 2$ membranes at the end of computation. This example is also of interest for parallel simulation using dynamic task creation. Different membranes can be distributed on different machines of a cluster. They can evolve independently accordingly to the evolution rules. Send-in and Send-out rules request message exchanges.

Based on the fact that CLIPS code is fully compatible with Jess, we have repeated the experiments reported in [8]. We concentrate our attention to the most consuming part of the simulation, namely after the final division when we have already $2^n + 2$ membranes. The number of rules to be fired is similar to those from the classical production systems benchmarks. The following table specifies the number of rules to be fired in the case of checking the validity of a Boolean expression with $m = n$.

$m \times n$	Membranes	Rules fired after the last division	Time for firing those rules	Reaching the final configuration	Total time of simulation
2×2	6	335	1 s	1 s	2 s
3×3	10	779	13 s	3 s	16 s
4×4	18	1835	428 s	6 s	434 s
5×5	34	?	?	49 s	?

The evaluation of time is measured on one PC of a cluster used in our tests. The cluster is composed of four PCs at 1.5GHz and 256Mb RAM, connected via a Myrinet switch ensuring communication at 2Gb/s.

We describe now how the P accelerator is working in order to assure the distribution of the membranes and the communication between rules. The P accelerator can be activated by a user in a simple way. Each Jess instances reads the simulator rules, the facts (the membrane rules to be applied), the membrane structure, and their contents. Each membrane is owned by a Jess instance. An instance can own none, one or several membranes. Rules are fired by an instance only if they are associated to an owned membrane. Each instance has copies of the other instance membranes. A change in the content of the father membrane can lead to a change in the children membrane. The sources and destinations used in the send and receive commands are related to their membrane owners. The send and receive rules are activated only if the involved sources and destinations have different owners.

4.5 Running Time

It is easy to note a significant reduction of the running time, even when we use the P accelerator running on one machine of the cluster (i.e., Jess instances are running on the same machine). In the following table, the shorter time is underlined for each number of Jess instances working concurrently. The lowest time depends on the dimension of the problem. It seems that for a $m \times m$ problem, the lowest time is given by m Jess instances.

Membranes	1 instance	2 instances	3 instances	4 instances
6	$\underline{1}$ s	1 s	3 s	3 s
10	13 s	$\underline{5}$ s	5 s	10 s
18	428 s	45 s	$\underline{25}$ s	33 s
34	Mem.out	1054 s	325 s	$\underline{200}$ s

Further reduction of the simulation time is expected when our P accelerator is running on several machines. These expectations are confirmed by our tests. The speedup S_p and the efficiency E_p of the parallel implementation are registered in the following table, the values being close to the ideal ones. It is easy to remark a normal increasing of the speedup with the number of rules to be fired. It is expected to obtain even better results for larger dimension of the validity problem.

		Membranes			
Instances	Machines	6	10	18	34
1	1	1 s	13 s	428 s	Memory out
2	1	1 s	5 s	45 s	1054 s
	2	1 s	3 s	23 s	528 s
	S_2	1	1.7	1.9	2
	E_2	0.51%	0.65%	0.95%	0.99%
3	1	3 s	5 s	25 s	325 s
	3	3 s	2 s	10 s	126 s
	S_3	1	2.5	2.5	2.6
	E_3	0.33%	0.83%	0.83%	0.87%
4	1	3 s	10 s	33 s	200 s
	4	2 s	3 s	9 s	52 s
	S_4	1.5	3.3	3.7	3.8
	E_4	0.37%	0.82%	0.91%	0.96%

5 Conclusion and Further Work

We present a new simulator of the P systems. It is efficient, flexible, and does not require any previous knowledge or expertise in computers. Since the simulator has some novel and interesting features related to efficiency, ease of use and generality, it can become a useful tool for the community, both theoretically and practically. Being GPL licensed, we expect it is eligible to become a simulator benchmark reference. The simulator is available at `http://psystems.ieat.ro`.

We intend to make the simulator available as a web service. Moreover, continuing the commitment to standards compliance, we will strive for SBML compatibility for our specification language. Further improvements are related to better debugging and visualization capabilities (including a flexible, fine and coarse-grained tracer), developing a library of macros and methodologies using the principles of modularity, extensibility and structured design from software engineering, introducing rules for the development of macros for P systems.

In this paper we also present an automatic parallelization tool of the existing sequential simulators of the P systems. We use a splitting technique of membranes in several Java threads, and then we develop a P accelerator enabling the cooperation of many Jess instances running in a cluster environment to speedup a P simulator. The P accelerator will be further developed to include facilities for membrane dissolution and division. Further extensions are under investigation.

References

1. G. Ciobanu, "Distributed algorithms over communicating membrane systems", *BioSystems* vol.70, 123–133, Elsevier, 2003.
2. G. Ciobanu, W. Guo, "P Systems Running on a Cluster of Computers", in Gh.Păun et al. (Eds.): Proceedings 4th Workshop on Membrane Computing, *LNCS* 2933, 123–139, Springer, 2004.
3. G. Ciobanu, D. Paraschiv, "Membrane Software. A P System Simulator", *Fundamenta Informaticae*, vol.49, no.1-3, 61–66, 2002.
4. G. Ciobanu, D. Petcu, "P accelerators: Parallelization of sequential simulators", in M.A.Gutierrez-Naranjo, Gh.Păun, M.J.Perez-Jimenez (Eds.): *Cellular Computing; complexity aspects*, ESF PESC Exploratory Workshop, Fenix Editora, Sevilla, 177-186, 2005.
5. C.L. Forgy, "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence* vol.19, 17–37, 1982.
6. E. Friedman-Hill, *Jess in Action: Rule-Based Systems in Java*, Manning Publications, 2003.
7. Gh. Paun, *Membrane Computing. An Introduction*, Springer, 2002.
8. M.J. Perez-Jimenez, F.J. Romero-Campero, "A CLIPS Simulator for Recognizer P Systems with Active Membranes", *Proceedings 2nd Brainstorming Week on Membrane Computing*, University of Sevilla Tech. Rep 01/2004, 387-413, 2004.

Communication Complexity as a Principle of Quantum Mechanics

Adán Cabello

Departamento de Física Aplicada II, Universidad de Sevilla, 41012 Sevilla, Spain
adan@us.es

Abstract. We introduce a two-party communication complexity problem in which the probability of success by using a particular strategy allows the parties to detect with certainty whether or not some forbidden communication has taken place. We show that the probability of success is bounded by nature; any conceivable method which gives a probability of success outside these bounds is impossible. Moreover, any conceivable method to solve the problem which gives a probability success within these bounds is possible in nature. This example suggests that a suitably chosen set of communication complexity problems could be the basis of an information-theoretic axiomatization of quantum mechanics.

1 Introduction

Quantum mechanics (QM), in contrast to relativity theory, lacks a basic principle from which the theory develops in a compelling manner. We still do not know the "physical reasons" responsible for the Hilbert-space structure of QM. Therefore, deriving QM from a reduced set of primitives or axioms with a clear physical content remains an open problem for the foundations of physics.

In recent years, it has been suggested that a set of information-theoretic axioms would be sufficient to derive QM [1,2,3,4]. This information-based program has a clear precedent in Wheeler's project of deducing the nature of the quantum world from the idea that information plays a significant role at the foundations of physics [5,6,7].

One attempt to construct QM around information is Zeilinger's "foundational principle" for QM [8]. Knowledge in physics is acquired by means of experimental results. Any experiment can be decomposed into a set of yes-no tests. Therefore, it is reasonable to assume that any "elementary" physical system only gives a definite yes-no answer for *one* specific yes-no test. That is, the most elementary system would carry just one bit of information, and extracting this information would require knowing how the system was prepared.

Although Zeilinger's principle provides a framework for understanding some fundamental traits of QM, like essential randomness and entanglement, it is still insufficient to derive the details of QM, since the set of theories which are consistent with it is too large.

A different approach is due to Fuchs [9,10,11] and Brassard [12], who suggested that QM can be derived from two cryptographic principles: the possibility of secure key distribution [13,14] (i.e., the possibility that a common secret sequence of random bits can be distributed to two authorized parties, Alice and Bob, whilst any unauthorized party cannot acquire any information about the sequence) and the impossibility of unconditionally secure bit commitment [15,16,17] (i.e., the impossibility of a protocol in which Bob receives some evidence from Alice that she has a bit value b in mind, such that this evidence forces Alice to not change b, but does not allow Bob to obtain any information about b until Alice chooses to reveal it by supplying further information). These two principles capture two of the fundamental features of QM: quantum key distribution is built on the idea that information gathering causes a necessary disturbance to quantum systems [13], while the theorem on the impossibility of unconditionally secure bit commitment is based on an entanglement-based attack [16,17].

Fuchs and Brassard's program has been further developed by Clifton, Bub, and Halvorson (CBH), who proposed to deduce QM from three information-theoretic constraints: the impossibility of superluminal information transfer, the impossibility of perfectly broadcasting the information contained in an unknown physical state, and the impossibility of unconditionally secure bit commitment [18]. This proposal has stirred some controversy. By presenting a toy model which is a counterexample, Smolin has showed that the CBH axioms by themselves do not necessarily lead to QM [19]. A similar toy model satisfying the CBH axioms (as well as key distribution) has been proposed by Spekkens [20]. In response, Halvorson and Bub have argued that Smolin's model violates the independence condition for space-like separated systems [21].

Here we shall propose a different approach for finding information-theoretic axioms for QM. This proposal is based on the following observations:

(i) Bell's theorem [22] is not about what QM *is*, but about what it is *not* —QM is not reproducible by local hidden-variable theories (LHVTs)—. Bell's inequalities are necessary conditions for LHVTs, not for QM: there are many predictions of QM which are in agreement with all of Bell's inequalities, and there are also many conceivable violations of Bell's inequalities that cannot be achieved within QM. However, although Bell's theorem is not a solid ground on which to build QM, a set of suitably chosen Bell's inequalities turns out to provide —at least for the simplest Bell-like scenario (two particles and two alternative experiments per particle)— the necessary *and sufficient* conditions for the corresponding correlations to be achievable by LHVTs and, therefore, provides an starting point for an axiomatization of such theories.

(ii) On the other hand, every proof of Bell's theorem —with or without inequalities— can be translated into a communication complexity problem (CCP), in which the probability of success is bounded between certain limits if the parties are restricted to sharing strings of bits, bounds which can be surpassed by allowing the parties to share quantum states. This observation places Bell's theorem in an information-theoretic scenario more suitable to our purposes.

(iii) It is known that the necessary and sufficient conditions for four numbers to represent correlations achievable by QM in the simplest Bell-like scenario can be expressed as a set of suitably chosen nonlinear inequalities.

(iv) These necessary and sufficient conditions for quantum correlations can be translated into a communication complexity scenario. The result of this translation could provide a basis for an information-theoretic axiomatization of QM.

The structure of this paper is dictated by these four points. In sections 2, 3, 4, and 5 we develop in more detail points (i), (ii), (iii), and (iv), respectively. Sections 3 and 5 are particularly important. In section 3 we review a two-party CCP in which the parties can increase their probability of success if they share prior quantum entangled states rather than classically correlated data. In section 5, we modify the problem and show that the corresponding probability of success allows the parties to detect with certainty whether or not some forbidden communication has taken place. At the same time, we show that there is no mechanism in nature which allows the parties to succeed (or fail) beyond a certain probability. Moreover, we show that any conceivable method which gives a probability between these bounds is implementable in nature.

2 Bell's Theorem and the Necessary and Sufficient Conditions for Local Hidden-Variable Correlations

2.1 The EPR-Bell Scenario

The Einstein-Podolsky-Rosen-Bell scenario [23,24,22,25] consists of two alternative dichotomic experiments (i.e., having only two possible outcomes, which we can assume to be ± 1), A_0 or A_1, on a particle A, and other two alternative dichotomic experiments, B_0 or B_1, on a distant particle B; "distant" means that the experiments A_i and B_j are space-like separated.

If we assume no-signaling (i.e., that two distant observers cannot signal to one another via their choice of local experiment), the statistics of a Bell-type set of experiments are described by 8 numbers; for instance, 8 suitably chosen joint probabilities $P(A_i = a_k, B_j = b_l)$ for the various possible outcomes, or the four expectation values for the local observables $\langle A_i \rangle$ and $\langle B_j \rangle$, plus the four correlations $\langle A_i B_j \rangle$. The set of correlations is a 4-dimensional projection of the 8-dimensional set of joint probabilities. The connection between both sets is given by $\langle A_i B_j \rangle = \sum_{k,l \in \{-1,1\}} kl P(A_i = k, B_j = l)$.

The EPR-Bell scenario is the simplest scenario where the differences between classical and quantum correlations arise. Moreover, if we assume that the choice of local experiment plays some role, the EPR-Bell scenario is the most basic building block of all other symmetric information-theoretic scenarios, since it is contained in any scenario involving more particles (or, equivalently, parties), more experiments per particle, and more possible outcomes per experiment.

2.2 Necessary and Sufficient Conditions for Local Hidden-Variable Correlations

Froissart [26] and Fine [27,28] proved that, for the EPR-Bell scenario, the set of all joint probabilities attainable by any LHVT forms an 8-dimensional polytope with 16 vertices and 24 faces. The 4-dimensional projection corresponding to the set of all correlation functions that can be attained by a LHVT is defined by 8 Bell's inequalities. In a LHVT, four numbers $-1 \leq \langle A_i B_j \rangle \leq 1$ $(i,j=0,1)$ can represent the four correlations appearing in the EPR-Bell scenario if and only if they satisfy the following 8 inequalities:

$$-2 \leq \langle A_0 B_0 \rangle + \langle A_0 B_1 \rangle + \langle A_1 B_0 \rangle - \langle A_1 B_1 \rangle \leq 2, \tag{1}$$

$$-2 \leq \langle A_0 B_0 \rangle + \langle A_0 B_1 \rangle - \langle A_1 B_0 \rangle + \langle A_1 B_1 \rangle \leq 2, \tag{2}$$

$$-2 \leq \langle A_0 B_0 \rangle - \langle A_0 B_1 \rangle + \langle A_1 B_0 \rangle + \langle A_1 B_1 \rangle \leq 2, \tag{3}$$

$$-2 \leq -\langle A_0 B_0 \rangle + \langle A_0 B_1 \rangle + \langle A_1 B_0 \rangle + \langle A_1 B_1 \rangle \leq 2. \tag{4}$$

Moreover, assuming no-signaling, these 8 inequalities provide a necessary and sufficient condition not only for the 4-dimensional projection corresponding to the correlations, but for the whole 8-dimensional set of joint probabilities (i.e., the restrictions on the other four numbers are due only to statistical constraints).

There are many predictions of QM in agreement with all of Bell's inequalities, and there are also many conceivable violations of Bell's inequalities that cannot be achieved within QM [29]. Therefore, Bell's theorem is not a solid ground onto which to build QM. However, the set of Bell's inequalities (1)-(4) provides —at least in the simplest Bell-like scenario— the necessary *and sufficient* conditions for the corresponding correlations achievable by LHVTs, and therefore provides a starting point for an axiomatization of such theories.

3 Bell's Theorem and Communication Complexity Problems

One of the most impressive applications of quantum resources for information processing is the reduction of the communication complexity required for certain computations [30,31,32,33,34,35,36,37,38,39]. This branch of quantum information receives —at least— three different names: entanglement-enhanced communication [30], quantum communication complexity [32,33,37], and —for those CCPs in which no bits are transmitted between the parties— quantum "pseudo-telepathy" [36,38,39,40].

A typical example of how QM plays a significant role in a communication complexity scenario is the following. Suppose that two or more separated parties need to compute a function of a number of inputs distributed among them. Using the best classical strategy this would require a certain minimum amount of bits to be transmitted between them. However, if the parties initially share some entangled states, then the amount of bits required is smaller than if no entanglement were present. The quantum advantage can be shown in a slightly different, but equivalent, way: if the number of transmitted bits is fixed, then QM allows the parties to develop entanglement-based protocols such that the probability for both parties to achieve the correct value for the function —the probability of success— is higher than it would be when using any protocol without entanglement.

Most of the described CCPs in which QM provides an advantage derive from proofs of Bell's theorem with or without inequalities. For instance, the quantum advantage in the CCP involving three parties proposed in [30] derives directly from Mermin's three-party version [41,42,43] of Greenberger-Horne-Zeilinger's proof [44,45] of Bell's theorem without inequalities. Similarly, the two-party CCP presented in [35] derives from the Clauser-Horne-Shimony-Holt (CHSH) [25] Bell-like inequality. More recently, the two-party CCP with perfect quantum efficiency described in [38] is based on a two-party proof of Bell's theorem without inequalities [46,47,48,49]. As Vaidman noted, every proof of Bell's theorem without inequalities can be cast into a game in which the probability of success is higher if some specific entangled states are allowed [50]. Convert this game into the evaluation of a function whose inputs has been distributed among distant parties, and then you will have a CCP with a quantum advantage. Indeed, it has been shown that for a wide class of Bell's inequalities there is always a CCP for which the protocol assisted by quantum states which violate the corresponding inequality is more efficient than any classical protocol [51].

Let us review the two-party CCP proposed in [35], which is based on the EPR-Bell scenario and makes use of the CHSH inequalities:

Rules. Suppose two separated parties: Alice and Bob. They are assumed to be *isolated* from each other except for those communications explicitly mentioned below.

Alice receives two bits: $x_A \in \{0,1\}$ and $y_A \in \{-1,1\}$. Analogously, Bob receives two bits: $x_B \in \{0,1\}$ and $y_B \in \{-1,1\}$. Both know that these four bits are produced so that the 16 possible variations occur with the same frequency.

Then, Alice sends Bob a bit, and Bob sends Alice another. Possessing only this information, their common goal is to compute the value of the function

$$f(x_A, y_A, x_B, y_B) = y_A y_B (-1)^{x_A x_B}, \tag{5}$$

with the highest possible probability of success. They win if and only if the value announced by Alice and the value announced by Bob are *both* correct.

A Simple Optimal Classical Protocol. Either whether Alice receives $x_A = 0$ or $x_A = 1$, she sends Bob the value $s_A = y_A$. Analogously, either whether Bob receives $x_B = 0$ or $x_B = 1$, he sends Alice the value $s_B = y_B$. Both put $s_A s_B$ as the value of f.

It can be checked that:

- In the 1/4 of the cases in which $x_A = 0$ and $x_B = 0$, they give the correct value.
- In the 1/4 of the cases in which $x_A = 0$ and $x_B = 1$, they give the correct value.

– In the 1/4 of the cases in which $x_A = 1$ and $x_B = 0$, they give the correct value.
– In the 1/4 of the cases in which $x_A = 1$ and $x_B = 1$, they give the *incorrect* value.

Therefore, the probability of success by using this protocol is $P_f = 3/4 = 0.75$.

An Optimal Quantum Protocol. Initially, Alice and Bob share two qubits prepared in the singlet state

$$|\psi^-\rangle = \frac{1}{\sqrt{2}} \left(|01\rangle - |10\rangle\right). \tag{6}$$

If Alice receives $x_A = 0$, she measures $A_0 = \sigma_x$ on her qubit. If she receives $x_A = 1$, she measures $A_1 = \sigma_y$. The value obtained in this measurement is $r_A \in \{-1, 1\}$. If Bob receives $x_B = 0$, he measures $B_0 = -(\sigma_x + \sigma_y)/\sqrt{2}$ on his qubit. If he receives $x_B = 1$, he measures $B_1 = (\sigma_y - \sigma_x)/\sqrt{2}$. The value obtained in this measurement is r_B.

Alice sends Bob the value $s_A = y_A r_A$. Bob sends Alice the value $s_B = y_B r_B$. Both put $s_A s_B$ as the value of f.

It can be checked that:

– In the 1/4 of the cases in which $x_A = 0$ and $x_B = 0$, they give the correct value if $r_A = r_B$, and give the incorrect value if $r_A = -r_B$.
– In the 1/4 of the cases in which $x_A = 0$ and $x_B = 1$, they give the correct value if $r_A = r_B$, and give the incorrect value if $r_A = -r_B$.
– In the 1/4 of the cases in which $x_A = 1$ and $x_B = 0$, they give the correct value if $r_A = r_B$, and give the incorrect value if $r_A = -r_B$.
– In the 1/4 of the cases in which $x_A = 1$ and $x_B = 1$, they give the correct value if $r_A = -r_B$, and give the incorrect value if $r_A = r_B$.

Therefore, the probability of success by using this protocol is

$$\begin{aligned}
P_f &= \frac{1}{4}\left[P(A_0B_0 = 1) + P(A_0B_1 = 1) + P(A_1B_0 = 1) + P(A_1B_1 = -1)\right] && (7)\\
&= \frac{1}{4}\left[\frac{1}{2}\left(1 + \frac{1}{\sqrt{2}}\right) + \frac{1}{2}\left(1 + \frac{1}{\sqrt{2}}\right) + \frac{1}{2}\left(1 + \frac{1}{\sqrt{2}}\right) + \frac{1}{2}\left(1 + \frac{1}{\sqrt{2}}\right)\right] && (8)\\
&= \frac{1}{2}\left(1 + \frac{1}{\sqrt{2}}\right) && (9)\\
&\approx 0.853. && (10)
\end{aligned}$$

Connection with Bell's Inequalities. Taking into account that $\langle A_i B_j\rangle = P(A_i B_j = 1) - P(A_i B_j = -1)$ and $P(A_i B_j = 1) + P(A_i B_j = -1) = 1$, we can write Eq. (7) as

$$P_f = \frac{1}{2} + \frac{1}{8}\left(\langle A_0B_0\rangle + \langle A_0B_1\rangle + \langle A_1B_0\rangle - \langle A_1B_1\rangle\right), \tag{11}$$

which contains the term in the middle in (1), so P_f is classically bounded,

$$\frac{1}{4} \leq P_f \leq \frac{3}{4}. \tag{12}$$

The classical protocol introduced before is optimal because it gives the highest possible classical probability of success.

4 The Necessary and Sufficient Conditions for Quantum Correlations

There are two equivalent expressions of the necessary and sufficient conditions for four numbers to represent correlations attainable by QM in the EPR-Bell scenario. The first was provided by Tsirelson, and takes the form of an inequality containing a polynomial of degree 6 in the correlations [52]. The second characterization of the quantum correlations is due to Landau [53], and involves an inequality with square roots and products of two correlations. Both can be written in a way analogous to (1)-(4), by means of 8 inequalities [54]. In QM, four numbers, $-1 \leq \langle A_i B_j \rangle \leq 1$ $(i, j = 0, 1)$ can represent the four correlations of the EPR-Bell scenario if and only if they satisfy the following 8 inequalities:

$$\begin{aligned} -2 \leq \frac{2}{\pi} \arcsin \langle A_0 B_0 \rangle + \frac{2}{\pi} \arcsin \langle A_0 B_1 \rangle + \frac{2}{\pi} \arcsin \langle A_1 B_0 \rangle & \\ - \frac{2}{\pi} \arcsin \langle A_1 B_1 \rangle \leq 2, & \qquad (13) \\ -2 \leq \frac{2}{\pi} \arcsin \langle A_0 B_0 \rangle + \frac{2}{\pi} \arcsin \langle A_0 B_1 \rangle - \frac{2}{\pi} \arcsin \langle A_1 B_0 \rangle & \\ + \frac{2}{\pi} \arcsin \langle A_1 B_1 \rangle \leq 2, & \qquad (14) \\ -2 \leq \frac{2}{\pi} \arcsin \langle A_0 B_0 \rangle - \frac{2}{\pi} \arcsin \langle A_0 B_1 \rangle + \frac{2}{\pi} \arcsin \langle A_1 B_0 \rangle & \\ + \frac{2}{\pi} \arcsin \langle A_1 B_1 \rangle \leq 2, & \qquad (15) \\ -2 \leq -\frac{2}{\pi} \arcsin \langle A_0 B_0 \rangle + \frac{2}{\pi} \arcsin \langle A_0 B_1 \rangle + \frac{2}{\pi} \arcsin \langle A_1 B_0 \rangle & \\ + \frac{2}{\pi} \arcsin \langle A_1 B_1 \rangle \leq 2. & \qquad (16) \end{aligned}$$

Note that these inequalities provide a necessary and sufficient condition for quantum correlations, while the much more famous Tsirelson's inequalities [58,59,60,61] only give a necessary condition.

5 Quantum Correlations and Communication Complexity Problems

From a non-expert's perspective, the advantage of the quantum entangled states used to improve the probability of success in the CCP described in section 3 could have come out of two magic boxes, each with a switch with two possible settings (the two alternative local experiments) and a button such that, when pressed, causes a light to flash red or green (the two possible outcomes) [56]. The interesting thing is that the light flashings for the four possible combinations of settings are mysteriously correlated. A naive, but reasonable, question is then why "better" pairs of magic boxes cannot be designed? The answer is simply that they are not allowed by nature. Then, the obvious question is which pairs of boxes are allowed by nature and which are forbidden. We know the answer (see section 4). Then why not choose this answer as an axiom of QM? This is an attractive possibility, since we can dress it up in more physical terms, and present it in an information-theoretic language by designing a CCP in which Alice and Bob are provided by a pair of boxes and their purpose is to find out with certainty, and just by looking at their final probability of success P, whether or not their boxes are allowed by nature.

In more practical terms, this CCP and its corresponding P would allow Alice and Bob to detect whether or not some forbidden communication between their boxes has taken place. Equivalently, it would also allow them to detect whether or not their boxes are "non-local machines" [57].

The following CCP is a modification of the one described in section 3, with the remarkable feature that the probability of success using a particular strategy allows Alice and Bob to detect whether or not some forbidden communication between their boxes has taken place.

The rules of the new CCP are the same as those of the CCP in section 3, except that the function Alice and Bob should compute is not (5), but a new function g calculated by a classical computer with a memory and a simple program, which implements, in the limit of infinite runs, a set of conditions described below. Hereafter, we will refer to such a classical computer as "the oracle".

The function g is such that, if Alice and Bob apply the protocol described in section 3, but replacing the alternative measurements on two qubits prepared in the singlet state by two boxes, one for Alice and the other for Bob, like those described above, then:

- In the 1/4 of the cases in which $x_A = 0$ and $x_B = 0$, they give the correct value with probability $\frac{1}{2} + \frac{1}{\pi} \arcsin(2P(r_A = r_B) - 1)$.
- In the 1/4 of the cases in which $x_A = 0$ and $x_B = 1$, they give the correct value with probability $\frac{1}{2} + \frac{1}{\pi} \arcsin(2P(r_A = r_B) - 1)$.
- In the 1/4 of the cases in which $x_A = 1$ and $x_B = 0$, they give the correct value with probability $\frac{1}{2} + \frac{1}{\pi} \arcsin(2P(r_A = r_B) - 1)$.
- In the 1/4 of the cases in which $x_A = 1$ and $x_B = 0$, they give the correct value with probability $\frac{1}{2} + \frac{1}{\pi} \arcsin(2P(r_A = -r_B) - 1)$.

It is easy to write a computer program implementing these conditions. The probability of success of getting the correct answer for g by using a protocol similar to the one described in section 3 is

$$\begin{aligned} P_{11} = \frac{1}{2} + \frac{1}{4\pi} & (\arcsin[2P(A_0B_0 = 1) - 1] + \arcsin[2P(A_0B_1 = 1) - 1] \\ & + \arcsin[2P(A_1B_0 = 1) - 1] + \arcsin[2P(A_1B_1 = -1) - 1]) . \end{aligned} \quad (17)$$

Now suppose that Alice and Bob are provided with a pair of boxes and they want to know with certainty wether or not the light flashings for the four possible combinations of settings of these boxes are allowed by nature (or, equivalently, they want to know whether or not any forbidden communication has taken place between the boxes during their usage). The following protocol will give them the answer with certainty. If the answer is that no forbidden communication has taken place, this would mean that, if any forbidden communication has taken place, then such a communication would be trivial in the sense that it could be simulated without communication, just by using physical systems isolated inside the boxes.

The protocol is as follows. Alice and Bob randomly use one of the following strategies:

(a) In some of the runs, when Alice receives $x_A = 0$ ($x_A = 1$), she selects her box's A_0 (A_1) setting, and when Bob receives $x_B = 0$ ($x_B = 1$), he selects his box's B_0 (B_1) setting, as in the protocol described in section 3.

(b) In some of the runs, when Alice receives $x_A = 0$ ($x_A = 1$), she selects her box's A_0 (A_1) setting, and when Bob receives $x_B = 0$ ($x_B = 1$), he selects his box's B_1 (B_0) setting.

(c) In some of the runs, when Alice receives $x_A = 0$ ($x_A = 1$), she selects her box's A_1 (A_0) setting, and when Bob receives $x_B = 0$ ($x_B = 1$), he selects his box's B_0 (B_1) setting.

(d) In the remaining runs, when Alice receives $x_A = 0$ ($x_A = 1$), she randomly selects her box's A_1 (A_0) setting, and when Bob receives $x_B = 0$ ($x_B = 1$), he randomly selects his box's B_1 (B_0) setting.

The oracle —which knows which have been x_A and x_B, and has kept track of the settings chosen by Alice and Bob for each run— keeps record of Alice and Bob's frequency of success in computing g after many runs using strategies (a), (b), (c) and (d). The frequency of success using strategy (a) should be approximately P_{11}, given by (17). Similarly, we will call P_{10}, P_{01}, and P_{00}, those frequencies of success in computing g by using strategies (b), (c), and (d), respectively.

Check that P_{00}, P_{01}, and P_{10} are approximately

$$\begin{aligned} P_{00} = \frac{1}{2} + \frac{1}{4\pi} & (\arcsin[2P(A_0B_0 = -1) - 1] + \arcsin[2P(A_0B_1 = 1) - 1] \\ & + \arcsin[2P(A_1B_0 = 1) - 1] + \arcsin[2P(A_1B_1 = 1) - 1]), \end{aligned} \tag{18}$$

$$\begin{aligned} P_{01} = \frac{1}{2} + \frac{1}{4\pi} & (\arcsin[2P(A_0B_0 = 1) - 1] + \arcsin[2P(A_0B_1 = -1) - 1] \\ & + \arcsin[2P(A_1B_0 = 1) - 1] + \arcsin[2P(A_1B_1 = 1) - 1]), \end{aligned} \tag{19}$$

$$\begin{aligned} P_{10} = \frac{1}{2} + \frac{1}{4\pi} & (\arcsin[2P(A_0B_0 = 1) - 1] + \arcsin[2P(A_0B_1 = 1) - 1] \\ & + \arcsin[2P(A_1B_0 = -1) - 1] + \arcsin[2P(A_1B_1 = 1) - 1]). \end{aligned} \tag{20}$$

At the end, the oracle gives Alice and Bob the four P_{ij}. If all of them satisfy

$$\frac{1}{4} \leq P_{ij} \leq \frac{3}{4}, \quad \forall i,j \in \{0,1\}, \tag{21}$$

then, the light flashings of Alice and Bob's boxes are correlated in a way allowed by nature, and thus Alice and Bob conclude that there has been no communication between their boxes. Otherwise, Alice and Bob's conclusion would be that the light flashings of their boxes are correlated in a way not allowed by nature, and thus that some forbidden communication has taken place between them.

It is a simple exercise to check that condition (21) is equivalent to (14)-(16), and therefore contains the necessary and sufficient condition for four correlations to be obtainable within QM. Then, an information-theoretic axiomatization of QM could contain the following principle: *"Nature can only solve this CCP with a probability of success $\frac{1}{4} \leq P_{ij} \leq \frac{3}{4}$: any conceivable solution between these limits is possible (i.e., there exists at least one probability distribution giving the required four correlations), and any conceivable solution outside these limits is impossible."*

6 Conclusion

Our main aim in this paper has been to draw attention to the fact that, in QM, CCPs are interesting beyond the fact that they show how entangled states can be used to solve some of them more efficiently than classically correlated data. We have suggested that CCPs can be the basis of an information-theoretic axiomatization of QM. An open question is whether or not the specific principle proposed in this paper contains enough information about the structure of QM such as to permit to derive QM from it.

Acknowledgements

I would like to thank the Spanish Ministerio de Ciencia y Tecnología Grant No. BFM2002-02815 and the Junta de Andalucía Grant No. FQM-239 for support.

References

1. C.A. Fuchs, in *Decoherence and its Implications in Quantum Computation and Information Transfer*, edited by A. Gonis and P.E.A. Turchi (IOS Press, Amsterdam, 2001).
2. C.A. Fuchs, in *Quantum Theory: Reconsideration of Foundations*, edited by A. Khrennikov (Växjö University Press, Växjö, 2002).
3. C.A. Fuchs, *Notes on a Paulian Idea. Fundational, Historical, Anecdotal and Forward-looking Thoughts on the Quantum. Selected Correspondence, 1995-2001* (Växjö University Press, Växjö, 2003).

4. C.A. Fuchs, J. Mod. Opt. **50**, 987 (2003).
5. J.A. Wheeler, in *Complexity, Entropy, and the Physics of Information*, edited by W.H. Zurek (Addison-Wesley, Redwood City, California, 1990), p. 1.
6. J.A. Wheeler, in *Quantum Coherence and Reality*, edited by J. Anandan and J.L. Safko (World Scientific, Singapore, 1995).
7. J.A. Wheeler and K.W. Ford, *Geons, Black Holes, and Quantum Foam* (W.W. Norton, New York, 1998).
8. A. Zeilinger, Found. Phys. **29**, 631 (1999).
9. C.A. Fuchs, Fortschr. Phys. **46**, 535 (1998). Reprinted in *Quantum Computation: Where Do We Want To Go Tomorrow?*, edited by S.L. Braunstein (Wiley-VCH, Weinheim, 1999), p. 229.
10. C.A. Fuchs, in *Quantum Communication, Computing, and Measurement*, edited by P. Kumar, G.M. D'Ariano, and O. Hirota (Kluwer, Dordrecht, 2000), p. 1.
11. C.A. Fuchs and K. Jacobs, Phys. Rev. A **63**, 062305 (2001).
12. G. Brassard, comments during discussion at meeting "Quantum Foundations in the Light of Quantum Information and Cryptography," Montreal, May 17–19, 2000.
13. C.H. Bennett and G. Brassard, *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing (Bangalore, India, 1984)* (IEEE, New York, 1984), p. 175.
14. H.-K. Lo and H.F. Chau, Science **283**, 2050 (1999).
15. G. Brassard and C. Crépeau, in *Advances in Cryptology: Proccedings of Crypto'90*, (Springer-Verlag, Berlin, 1991) p. 49.
16. D. Mayers, Phys. Rev. Lett. **78**, 3414 (1997).
17. H.-K. Lo and H.F. Chau, Phys. Rev. Lett. **78**, 3410 (1997).
18. R. Clifton, J. Bub, and H. Halvorson, Found. Phys. **33**, 1561 (2003).
19. J.A. Smolin, quant-ph/0310067.
20. R.W. Spekkens, unpublished, reported in [19].
21. H. Halvorson and J. Bub, quant-ph/0311065.
22. J.S. Bell, Physics **1**, 195 (1964).
23. A. Einstein, B. Podolsky, and N. Rosen, Phys. Rev. **47**, 777 (1935).
24. D. Bohm, *Quantum Theory* (Prentice-Hall, Englewood Cliffs, New Jersey, 1951).
25. J.F. Clauser, M.A. Horne, A. Shimony, and R.A. Holt, Phys. Rev. Lett. **23**, 880 (1969).
26. M. Froissart, Nuovo Cimento Soc. Ital. Fis. **64B**, 241 (1981).
27. A. Fine, Phys. Rev. Lett. **48**, 291 (1982).
28. A. Fine, J. Math. Phys. **23**, 1306 (1982).
29. S. Popescu and D. Rohrlich, Found. Phys. **24**, 379 (1994).
30. R. Cleve and H. Buhrman, Phys. Rev. A **56**, 1201 (1997).
31. H. Buhrman, R. Cleve, and A. Wigderson, in *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing* (ACM Press, New York, 1998), p. 63.
32. H. Buhrman, W. van Dam, P. Høyer, and A. Tapp, Phys. Rev. A **60**, 2737 (1999).
33. R. Raz, in *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing* (ACM Press, New York, 1999), p. 358.
34. A.M. Steane and W. van Dam, Phys. Today **53** (2), 35 (2000).
35. H. Buhrman, R. Cleve, and W. van Dam, SIAM J. Comput. **30**, 1829 (2001).
36. G. Brassard, A. Broadbent, and A. Tapp, *Proceedings of WADS 2003*; quant-ph/0306042.
37. G. Brassard, Found. Phys. **33**, 1593 (2003).
38. G. Brassard, A. Broadbent, and A. Tapp, quant-ph/0407221.
39. G. Brassard, A. Broadbent, and A. Tapp, quant-ph/0408052.

40. G. Brassard, A. A. Methot, and A. Tapp, quant-ph/0412136.
41. N.D. Mermin, Phys. Today **43** (6), 9 (1990).
42. N.D. Mermin, Am. J. Phys. **58**, 731 (1990).
43. N.D. Mermin, Phys. Rev. Lett. **65**, 3373 (1990).
44. D.M. Greenberger, M.A. Horne, and A. Zeilinger, in *Bell's Theorem, Quantum Theory, and Conceptions of the Universe*, edited by M. Kafatos (Kluwer Academic, Dordrecht, 1989), p. 69.
45. D.M. Greenberger, M.A. Horne, A. Shimony, and A. Zeilinger, Am. J. Phys. **58**, 1131 (1990).
46. A. Cabello, Phys. Rev. Lett. **86**, 1911 (2001).
47. A. Cabello, Phys. Rev. Lett. **87**, 010403 (2001).
48. P.K. Aravind, Found. Phys. Lett. **15**, 397 (2002).
49. P.K. Aravind, Am. J. Phys. **72**, 1303 (2004).
50. L. Vaidman, Phys. Lett. A **286**, 241 (2001).
51. Č. Brukner, M. Żukowski, J. W. Pan, and A. Zeilinger, Phys. Rev. Lett. **92**, 127901 (2004).
52. B.S. Tsirelson, Zap. Nauchn. Semin LOMI **142**, 174 (1985). English version: J. Soviet Math. **36**, 557 (1987).
53. L.J. Landau, Found. Phys. **18**, 449 (1988).
54. B.S. Tsirelson, Hadronic J. Supplement **8**, 329 (1993).
55. A. Cabello, quant-ph/0409192.
56. N.D. Mermin, Am. J. Phys. **49**, 940 (1981).
57. J. Barrett, N. Linden, S. Massar, S. Pironio, S. Popescu, and D. Roberts, quant-ph/0404097 (to be published in Phys. Rev. A).
58. B.S. Tsirelson, Lett. Math. Phys. **4**, 93 (1980).
59. L.A. Khalfin and B.S. Tsirelson, in *Symposium on the Foundations of Modern Physics: 50 Years of the Einstein-Podolsky-Rosen Experiment*, edited by P. Lahti and P. Mittelstaedt (World Scientific, Singapore, 1985), p. 441.
60. L.J. Landau, Phys. Lett. A **120**, 54 (1987).
61. L.A. Khalfin and B.S. Tsirelson, Found. Phys. **22**, 879 (1992).

On Model-Checking of P Systems[*]

Zhe Dang[1,**], Oscar H. Ibarra[2], Cheng Li[1], and Gaoyan Xie[1]

[1] School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164, USA
[2] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA

Abstract. Membrane computing is a branch of molecular computing that aims to develop models and paradigms that are biologically motivated. It identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. Because of the nature of maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way that would allow us to solve currently intractable problems. In this paper, we look at various models of P systems and investigate their model-checking problems. We identify what is decidable (or undecidable) about model-checking these systems under extended logic formalisms of CTL. We also report on some experiments on whether existing conservative (symbolic) model-checking techniques can be practically applied to handle P systems with a reasonable size.

1 Introduction

There has been a flurry of research activities in the area of membrane computing (a branch of molecular computing) initiated five years ago by Gheorghe Paun [9]. Membrane computing identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. It abstracts from the way living cells process chemical compounds in their compartmental structures. Thus, regions defined by a membrane structure contain objects that evolve according to given rules. The objects can be described by symbols or by strings of symbols, in such a way that multisets of objects are placed in regions of the membrane structure. The membranes themselves are organized as a Venn diagram or a tree structure where one membrane may contain other membranes. By using the rules in a nondeterministic, maximally parallel manner, transitions between the system configurations can be obtained. A sequence of transitions shows how the system is evolving. Various ways of controlling the transfer of objects from a region to another and applying the rules, as well as possibilities to dissolve, divide or create membranes have been studied [10]. Due to the maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way that would allow

[*] The work by Zhe Dang, Cheng Li and Gaoyan Xie was supported in part by NSF Grant CCF-0430531. The work by Oscar H. Ibarra was supported in part by NSF Grant CCF-0430945.

[**] Corresponding author(zdang@eecs.wsu.edu).

us to solve currently intractable problems (in much the same way as the promise of quantum and DNA computing) once future bio-technology gives way to a practical bio-realization.

Designing a P system to achieve a pre-defined computational goal is difficult and extremely error-prone. This is because, unlike traditional programming languages, the inherent maximal parallelism in the model makes the P system highly nondeterministic, concurrent, and, more importantly, lack of control-flow structure (e.g., without "control states"). The difficulties naturally call for algorithmic (i.e., decidable) solutions to the verification problem: whether a designed P system does have the desired behavioral property. The solutions will also be important in the future when people implement a P system in vivo. This is because an erroneous P system will be deemed a failure in an expensive lab realization. It is highly desirable to validate the P system in advance in vitro, e.g., through digital computers. Another important application of results concerning decidable properties of P systems is in biology, where such systems are now being proposed for the modeling and simulation of cells. While previous work on modeling and simulation use continuous mathematics such as differential equations, P systems will allow us to use discrete mathematics and algorithms. As a P system models the computation that occurs in a living cell, an important problem is to develop tools for determining reachability between configurations, i.e., how the system evolves over time. Specifically, given a P system and a configuration U (a configuration is the number and distribution of the different types of objects in the various membranes in the system) and some constraints f (e.g., a linear constraint over the numbers of different types of objects), is there a configuration V satisfying f that is reachable from U? This is essentially a model-checking [4] problem: whether a transition system meets a desired temporal property.

Unfortunately, to our best knowledge, model-checking theories for P systems have never been studied so far. In our opinion, this is, probably, due to the short history of membrane computing and also due to the theoretical difficulty of handling the maximal parallelism, which is quite different from the conventional infinite state transition systems currently being studied in model-checking.

In this paper, we try to identify what is decidable about model-checking of P systems. Clearly, since a P system is Turing complete in general, we have to focus on restricted P systems in order to make the model-checking decidable. The first restriction is to focus on P systems with only one membrane. Essentially, this is more like a technical convenience than a real restriction. Since the P system model studied in this paper does not have priority rules and membrane dissolving rules, multi-membranes can be equivalently collapsed into one membrane through properly renaming symbols in a membrane. The second restriction is to focus on bounded P systems (BPS) where rules are only in the form of $u \to v$, where u and v are multisets of objects with $|u| \geq |v|$ (the size $|u|$ denotes the number of objects in u). Notice that, since we do not require that a BPS starts with a multiset whose size is bounded by a fixed constant, the BPS is essentially an infinite state system (or more precisely, a system with an unbounded number of states). An execution of a BPS can be understood as a sequence of multisets (configurations). The formalism that we choose to specify the desired behavioral property is $\mathrm{CTL}^{\mathrm{REG}}$ and $\mathrm{CTL}^{\mathrm{LIN}}$, which allow us to reason upon the executions. In short,$\mathrm{CTL}^{\mathrm{REG}}$ and $\mathrm{CTL}^{\mathrm{LIN}}$ are simply CTL [3] augmented with atomic predicates in REG and in LIN, respectively. More precisely, in REG, one can compare the multiplicity of a symbol against an integer constant, while in LIN, one can compare a linear combination of the multiplicities of all the symbols against an integer constant. Notice that basic properties like halting are expressible in $\mathrm{CTL}^{\mathrm{REG}}$. The corresponding $\mathrm{CTL}^{\mathrm{REG}}$ (as well as $\mathrm{CTL}^{\mathrm{LIN}}$) model-checking problem is to argue whether a given temporal formula is interpreted as an empty multiset of configurations.

We first look at a non-cooperative BPS M where each rule is in the form of $a \to b$, where a and b are symbols, in Section 3. Surprisingly, for such systems, the $\mathrm{CTL}^{\mathrm{REG}}$ model-checking problem is undecidable, even for a simple form of $\exists\mathcal{U}$ (exist-until) properties. When we further require that, in M, a symbol can evolve into at most one kind of symbol, we show that the $\mathrm{CTL}^{\mathrm{REG}}$ becomes decidable. On the other hand, when $\mathrm{CTL}^{\mathrm{LIN}}_{-}$ (roughly, dropping $\exists\mathcal{U}$ from $\mathrm{CTL}^{\mathrm{LIN}}$) is considered, its model-checking problem becomes decidable for non-cooperative BPS. Lastly, when some form of determinism is used to restrict a BPS, the $\mathrm{CTL}^{\mathrm{LIN}}$ is decidable. We then turn to study the model-checking problems for BPS (which is not necessarily non-cooperative, i.e., $|u|$ can be greater than 1), in Section 4. We first give an exact automata-theoretic characterization of (non)deterministic BPSs reachable and halting configurations. That is, BPS is equivalent to each of the following three classes of automata: linear-bounded multicounter machines, $log\ n$ space-bounded Turing machines, two-way multihead finite automata. From this result, one can easily conclude that even $\mathrm{CTL}^{\mathrm{REG}}_{-}$ is undecidable for (non)deterministic BPS. In the section, we also study some notions of determinism that make BPS decidable for various model-checking problems.

Given the undecidability results in model-checking P systems, finally, in Section 5, we conduct some experiments to see whether existing conservative (symbolic) model-checking techniques such as Omega (which handles infinite state space) and SPIN (which handles finite state space) for concurrent linear arithmetic programs can be practically applied to handle P systems (which are not necessarily BPS, and also with multi-membranes, and even with priority among rules) with a reasonable size. Previous experiments [1] used a model-checker of rewriting systems where, like our SPIN experiments, object multiplicities have to be restricted to a finite domain. One of the purposes of our experiments is to let us know if the maximally parallelism and "lack of control-flow structure" in P systems would cause existing symbolic encodings for concurrent systems to fail terribly. Our preliminary experiments show that additional effort is needed in studying more efficient encodings and, in particular, new techniques to extract the implicit control-flow from P system rules.

2 P Systems and Their CTL Model-Checking Problems

Let $\mathbf{N}$ be the set of nonnegative integers and $\Sigma = \{a_1, \ldots, a_k\}$ be an alphabet, for some k, and u be a (finite) multiset over the alphabet. In this paper, we do not distinguish between several representations of u. That is, u can be treated as a vector in $\mathbf{N}^k$ (the components are the multiplicities of the symbols in Σ); u can be treated as a word where we only care about the counts of symbols (i.e., its Parikh map). We now introduce formulas to define some sets of multisets. An atomic *regular* predicate is in the form of

$\#(a) \sim n$, where $a \in \Sigma$, $n \in \mathbf{N}$ and $\sim \in \{>, <, =, \geq, \leq\}$. The predicate is interpreted as the subset of multisets u over Σ such that the multiplicity $\#(a)$ of symbol a satisfies the predicate. A regular formula is a Boolean combination of atomic regular predicates. We use REG to denote the set of regular formulas. An atomic *linear* predicate is in the form of $\sum_{1\leq i\leq k} n_i \cdot \#(a_i) \sim n$, where the n_i's and n are integers (positive, 0, negative), and $\sim \in \{>, <, =, \geq, \leq, \equiv_m\}$ with $0 \neq m \in \mathbf{N}$. The predicate is interpreted as a subset of multisets over Σ accordingly. A linear formula is a Boolean combination of atomic linear predicates. We use LIN to denote the set of linear formulas. A set $S \subseteq \mathbf{N}^k$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in $\mathbf{N}^k$ such that $S = \{v \mid v = v_0 + a_1 v_1 + \ldots + a_t v_t,\ a_i \in \mathbf{N}\}$. A set $S \subseteq \mathbf{N}^k$ is *semilinear* if it is a finite union of linear sets. A *Presburger formula* is constructed from atomic linear predicates using quantification and Boolean operators. It is known that the following items are equivalent: (1) a set of multisets (treated as vectors) is semilinear, (2) the set is definable by a linear formula, (3) the set is definable by a Presburger formula.

In this paper, we only focus on P systems without priority rules and membrane dissolving rules. In this case, as we mentioned earlier, it suffices for us to consider P systems with one membrane since multiple membranes can be equivalently collapsed into one by properly renaming symbols within a membrane.

A (1-membrane) P system M is specified by a finite set of rules. Each rule is in the form of $u \to v$ where u and v are multisets over alphabet Σ. A configuration in M is a multiset. As with the standard semantics of P systems [9,10,11], each evolution step, called a *maximally parallel move*, is a result of applying all the rules in G in a maximally parallel manner. More precisely, let $u_i \to v_i$, $1 \leq i \leq m$, be all the rules in M. We use $R = (r_1, \ldots, r_m) \in \mathbf{N}^m$ to denote a multiset of rules, where there are r_i instances of rule $u_i \to v_i$, for each $1 \leq i \leq m$. Let U and V be two configurations (multisets) over Σ. The rule multiset R is *enabled* under configuration U if U contains $\sum_{1\leq i\leq m} r_i \cdot u_i$ (i.e., U contains the multiset union of r_i copies of multiset u_i, for all $1 \leq i \leq m$). The result of applying R over U is to replace, in parallel, each of the r_i copies of u_i in U with v_i. The rule multiset R is *maximally enabled* under configuration U if it is enabled under U and, for any other rule multiset R' that strictly contains R, R' is not enabled under configuration U. Notice that, for the same U, a maximally enabled rule multiset may not be unique (i.e., M is in general nondeterministic). U can reach V through a maximally parallel move, written $U \to_M V$, if there is a maximally enabled rule multiset R such that V is the result of applying R over U. Formally, $U \to_M V$ iff $\exists r_1, \ldots, r_m \in \mathbf{N}.\ \mathrm{MaxEnable}(r_1, \ldots, r_m, U) \wedge\ \mathrm{Apply}(r_1, \ldots, r_m, U, V)$, where $\mathrm{MaxEnable}(r_1, \ldots, r_m, U)$, indicating that $(r_1, \ldots, r_m)$ is maximally enabled under configuration U, is the following formula:

$$U \geq \sum_{1\leq i\leq m} r_i \cdot u_i \wedge \forall r_1' \geq r_1, \ldots, r_m' \geq r_m. (U \geq \sum_{1\leq i\leq m} r_i' \cdot u_i \Rightarrow r_1' = r_1 \wedge \ldots \wedge r_m' = r_m),$$

and $\mathrm{Apply}(r_1, \ldots, r_m, U, V)$, indicating that V is the result of applying $(r_1, \ldots, r_m)$ over U, is the following formula: $V = U - \sum_{1\leq i\leq m} r_i \cdot u_i + \sum_{1\leq i\leq m} r_i \cdot v_i$. Notice that, in above, we treat the multisets (i.e., U, V, the u's, and the v's) as vectors in $\mathbf{N}^k$. Clearly, a maximally parallel move in M is always definable by a Presburger formula. Starting from some initial configuration, an execution of M goes through a sequence of configurations, where each configuration is derived from the directly preceding configuration in one maximally parallel move. Formally, we use $U \rightsquigarrow_M V$ to denote the fact that V is reachable from V; i.e., for some n and $U_0, \ldots, U_n$, we have $U = U_0 \rightarrow_M \ldots \rightarrow_M U_n = V$.

From above, a P system M can be treated as a transition system between multisets or vectors in $\mathbf{N}^k$. There has been an established theory, called model-checking, in algorithmically answering verification queries over a transition system's behavior. For a finite state transition system, the queries can be specified in a temporal logic like the computation tree logic (CTL) [3] and various model-checking algorithms are known [4]. For infinite state transition systems, the logic can also be interpreted in many cases (e.g., [2]). In below, we formulate the CTL formalism that we will use to specify our verification queries for P systems.

Let $\mathcal{A}$ be a given class of *atomic* predicates. The CTL$^{\mathcal{A}}$ formulas f are exactly defined with the following grammar: $f ::= A \mid f \wedge f \mid f \vee f \mid \neg f \mid \exists \circ f \mid \forall \circ f \mid f \ \exists\mathcal{U}\ f \mid f \ \forall\mathcal{U}\ f$, where $A \in \mathcal{A}$ is an atomic formula (predicate), and $\circ$ stands for "next" and $\mathcal{U}$ stands for "until". As usual, the eventuality operator $\exists \diamond f$ is the shorthand of $true \ \exists\mathcal{U}\ f$, and, its dual $\forall\Box f$ is simply $\neg \exists \diamond \neg f$. We use CTL$^{\mathcal{A}}_{-}$ to denote the fragment of CTL$^{\mathcal{A}}$ where the formulas f are exactly defined with the following grammar: $f ::= A \mid f \wedge f \mid f_1 \vee f_2 \mid \neg f \mid \exists \circ f \mid \forall \circ f \mid \exists \diamond f \mid \forall \Box f$, where $A \in \mathcal{A}$ is an atomic formula (predicate).

Let M be a P system. We interpret each CTL$^{\mathcal{A}}$ formula as a subset of configurations of M. That is, the interpretation, written $[f]^M$, is a subset of multisets of objects in M. Formally, the interpretation is recursively defined as follows [2]:

- $[A]^M$ is a *given* subset of multisets of objects in M, where $A \in \mathcal{A}$;
- $[f_1 \wedge f_2]^M$ is $[f_1]^M \cap [f_2]^M$; $[f_1 \vee f_2]^M$ is $[f_1]^M \cup [f_2]^M$;
- $[\neg f_1]^M$ is the complement of $[f_1]^M$; (the universe is the set of all multisets of objects in M)
- $[\exists \circ f_1]^M$ (resp. $[\forall \circ f_1]^M$) is the set of configurations U_1 such that, for some (resp. any) execution $U_1 \rightarrow_M U_2 \rightarrow_M \ldots$, we have $U_2 \in [f_1]^M$;
- $[f_1 \ \exists\mathcal{U}\ f_2]^M$ (resp. $[f_1 \ \forall\mathcal{U}\ f_2]^M$) is the set of configurations U_1 such that, for some (resp. any) execution $U_1 \rightarrow_M U_2 \rightarrow_M \ldots$, we have $U_1, \ldots, U_n \in [f_1]^M$ and $U_{n+1} \in [f_2]^M$, for some n.

The CTL$^{\mathcal{A}}$ *model-checking problem* is to decide whether, given a P system M and a CTL$^{\mathcal{A}}$ formula f, the set $[f]^M$ is empty. Notice that, in our definition of the CTL$^{\mathcal{A}}$ model-checking problem shown above, we did not mention the initial configurations of M. In fact, a verification question like whether a given initial configuration U_{init} satisfies f can also be formulated in our definition as follows: is $[A_{\text{init}} \wedge f]^M$ empty? where A_{init} is an atomic regular predicate where U_{init} is the only satisfying configuration.

In this paper, we focus on model-checking problems of CTL$^{\text{REG}}$ and CTL$^{\text{LIN}}$. Unfortunately, the maximal parallelism in P systems is too powerful to make P systems model-checkable; even in simple cases, P systems are able to be Turing complete. This leads us to study restricted forms of P systems where model-checking problems could be decidable. To this end, we focus on *bounded P systems* (BPS), in which each rule is in the form of $u \rightarrow v$ with $|u| \geq |v|$ ($|u|$ denotes the number of objects in u).

3 CTL Model-Checking of Non-cooperative Bounded P Systems

Let M be a non-cooperative BPS. That is, M is a 1-membrane P system whose rules are in the form of $a \to b$ or in the form of $a \to \Lambda$ (i.e., one object evolves into at most one object), where $a, b \in \Sigma$. We first show that the $\mathrm{CTL}^{\mathrm{REG}}$ model-checking problem is undecidable for M. Clearly, as we have mentioned earlier, when M has multi-membranes, it can be collapsed into one with 1-membrane. Hence, all the results in this section can be easily generalized to non-cooperative BPSs with multiple membranes.

Theorem 1. *The* $\mathrm{CTL}^{\mathrm{REG}}$ *model-checking problem for non-cooperative BPSs is undecidable. In fact, the undecidability remains even for* $\mathrm{CTL}^{\mathrm{REG}}$ *formulas in the form of* $\mathrm{INIT} \wedge (A \exists \mathcal{U} H)$*, where* INIT, A *and* H *are regular formulas in* REG.

We should point out that in the proof of Theorem 1 we did not use rules in the form of $a \to \Lambda$. Hence, Theorem 1 still holds when only rules in the form $a \to b$ are used. Because of the theorem, we will study a restricted form of M that makes $\mathrm{CTL}^{\mathrm{REG}}$ model-checking decidable. A non-cooperative BPS M is *special* when, for any a, if $a \to b$ and $a \to c$ with $b, c \neq \Lambda$ are rules in M, then $b = c$ (i.e., a could be disappear with $a \to \Lambda$ but it can not evolve into two kinds of symbols).

Theorem 2. *The* $\mathrm{CTL}^{\mathrm{REG}}$ *model-checking problem for special and non-cooperative BPSs is decidable.*

Because of the undecidability result in Theorem 1, we would like to investigate a fragment of a CTL logic that makes the model-checking problem for non-cooperative BPSs decidable. Before we proceed further, we need an intermediate result. Let M be a non-cooperative BPS, whose alphabet is $\Sigma = \{a_1, \ldots, a_k\}$. Recall that we use $u \rightsquigarrow_M v$ to denote the fact that multiset u can reach multiset v in M through some number of maximally parallel moves. We first show a characterization on the reachability relation $\rightsquigarrow_M \subseteq \mathbf{N}^k \times \mathbf{N}^k$, which leads to Theorem 4 later.

Theorem 3. *The reachability relation* $\rightsquigarrow_M \subseteq \mathbf{N}^k \times \mathbf{N}^k$ *for a non-cooperative BPS M is definable by a linear formula in* LIN.

Theorem 4. *The* $\mathrm{CTL}_{-}^{\mathrm{LIN}}$ *model-checking problem for non-cooperative BPSs is decidable.*

4 Reachability in Bounded P Systems

We now consider a bounded P system (BPS) M that is not necessarily noncooperative. That is, rules in M are in the form of $u \to v$ with $|v| \leq |u|$. Clearly, from Theorem 1, the $\mathrm{CTL}^{\mathrm{REG}}$ model-checking problem remains undecidable for M. However, encouraged by the decidability results in Theorem 4 for non-cooperative bounded P systems, we would like to know whether the $\mathrm{CTL}_{-}^{\mathrm{REG}}$ model-checking problem for (not necessarily non-cooperative) BPSs would still be decidable. In this section, we will prove that this is not true, even in very simple cases. We say that, when started with some given configuration, a BPS M has a halting computation if M has an execution that leads to a halting configuration (i.e., none of the rules is enabled).

We first consider the following problem: Given a bounded P system M with rules of the form $u \rightarrow v$, where $|u| = |v| = 1$ or 2 and a fixed multiset w and a distinct symbol o not in w, is there an n such that when M is started with multiset wo^n (the multiset union of w and n copies of o), it eventually halts? We shall refer to this as the emptiness problem for bounded P systems. We will show that this problem is undecidable. In fact, this result holds even when the system is *deterministic* in the sense that the maximally parallel multiset of rules applicable at each step in the computation is unique. We only sketch the proof in this paper. The idea is to relate the computation of M to a restricted type of multicounter machine, called linear-bounded multicounter machine, whose emptiness is known undecidable.

Consider a deterministic (nondeterministic) multicounter machine Z that is linear-bounded in the sense that when given an input n in one of the counters (called the input counter) and zeros in the other counters, computes in such a way that the sum of the values of the counters at any time during the computation is at most n. One can normalize the computation so that every increment is preceded by a decrement (i.e., if Z wants to increment a counter C, it first decrements some counter D and then increments C) and every decrement is followed by an increment. We do not require that the contents of the counters are zero when the machine halts.

We will show that we can construct a deterministic (nondeterministic) bounded P system M which uses a fixed multiset w such that, when M is started with multiset wo^n, it simulates Z and has a halting computation if and only if Z halts on input n. (Again, we do not assume that the halting configuration of M to be in any special form.) Moreover, the rules of M are of the form $u \rightarrow v$, where $|u| = |v| = 1$ or 2. Clearly, it follows that the computation of M is linear-bounded in the sense that any reachable configuration has length exactly $|w| + n$ (i.e., the size of the computation space is always the same).

It is convenient to use an intermediate P system, which we shall call RCPS, a restricted version of the CPS (communicating P system) introduced in [13]. A CPS has multiple membranes labeled $1, 2, ...,$ where 1 is the skin membrane. The rules in any membrane are of the forms: (1). $a \rightarrow a_x$, (2). $ab \rightarrow a_x b_y$, (3). $ab \rightarrow a_x b_y c_{come}$, where a, b, c are objects, x, y (which indicate the directions of movements of a and b) can be $here$, out, or in_j. The designation $here$ means that the object remains in the membrane containing it, out means that the object is transported to the membrane directly enclosing the membrane that contains the object (or to the environment if the object is in the skin membrane). The designation in_j means that the object is moved into the membrane, labeled j, that is directly enclosed by the membrane that contains the object. A rule of the form (3) can only appear in the skin membrane. When such a rule is applied, c is imported through the skin membrane from the environment (i.e., outer space) and will become an element in the skin membrane. In one step, all rules are applied in a maximally parallel manner. For notational convenience, when the target designation is not specified, we assume that the symbol remains in the membrane containing the rule.

Let V be the set of all objects (i.e., symbols) that can appear in the system, and o be a distinguished object (called the *input symbol*). A CPS M has m membranes, with a distinguished *input membrane*. We assume that only the symbol o can enter and exit the skin membrane (thus, all other symbols remain in the system during the computation). We say that M accepts o^n if M, when started with o^n in the input membrane initially (with no o's in the other membranes), eventually halts. Note that objects in $V - \{o\}$ have fixed numbers and their distributions in the different membranes are fixed initially. Moreover, their multiplicities remain the same during the computation, although their distributions among the membranes may change at each step. The language accepted by M is $L(M) = \{o^n \mid o^n$ is accepted by $M\}$.

It is known that a language $L \subseteq o^*$ is accepted by a deterministic (nondeterministic) CPS if and only if it is accepted by a deterministic (nondeterministic) multicounter machine. (Again, define the language accepted by a multicounter machine Z to be $L = \{o^n \mid Z$ when given n has a halting computation $\}$). The "if" part was shown in [13]. The 'only if" part is easily verified. Hence, every unary recursively enumerable language can be accepted by a deterministic CPS (hence, also by a nondeterministic CPS).

In a recent paper [8], it was shown that $L \subseteq o^*$ is accepted by a deterministic (nondeterministic) linear-bounded multicounter machine if and only if it is accepted by a deterministic (nondeterministic) CPS which is restricted in that the environment does not contain any object initially. The system can expel objects into the environment but only expelled objects can be retrieved from the environment. The restricted system is called deterministic (nondeterministic) RCPS.

We can now modify the construction in [8] by introducing a new membrane in the skin membrane which would simulate the environment. This is possible since, in an RCPS, the environment does not contain any object initially and only o can be expelled into the environment and can be retrieved from the environment. It follows that the modified RCPS need only use rules of the form (1) and (2). But the modified RCPS, call it M, has multiple membranes. We will convert this to a 1-membrane system M'. Suppose that M has membranes $1, ..., m$. For each object a in V, M' will have symbols $a_1, ..., a_m$. In particular, for the distinguished input symbol o in V, M' will have $o_1, ..., o_m$. Hence the distinguished input symbol in M' is o_{i_0}, where i_0 is the index of the input membrane in M. We can convert M to the system M' as follows:

1. If $a \rightarrow a_x$ is a rule in membrane i of M, then $a_i \rightarrow a_j$ is a rule in M, where j is the index of the membrane into which a is transported to, as specified by x.
2. If $ab \rightarrow a_x a_y$ is a rule in membrane i of M, then $a_i b_i \rightarrow a_j b_k$ is a rule in M, where i and j are the index numbers of the membranes into which a and b are transported to, as specified by x and y.

Thus, corresponding to the initial configuration wo^n of M, where o^n is in the input membrane i_0 and w represents the configuration denoting all the other symbols (different from w) in the other membranes, M' will have initial configuration $w'o_{i_0}^n$, where w' are symbols in w renamed to identify their locations in M.

Clearly, M' accepts $o_{i_0}^n$ if and only if M accepts o^n, and M' is a deterministic (nondeterministic) bounded P system. Now it is easy to show that the emptiness problem for deterministic linear-bounded multicounter machines (i.e., given Z, is there an input n such that Z halts?) is undecidable. Hence, we have:

Theorem 5. *It is undecidable to determine, given a deterministic (nondeterministic) BPS M and a fixed multiset w, whether there is an n such that M starting with multiset wo^n has a halting computation.*

For the next result, we need the fact that linear-bounded multicounter machines, $log\ n$ space-bounded TMs, and two-way multihead FAs are all equivalent (for both the deterministic and nondeterministic versions). As a corollary to Theorem 5, we can show that Theorem 4 does not hold for deterministic (nondeterministic) bounded P systems, even in very simple cases. Recall that $Halt$ is a regular formula in REG that defines all the halting configurations. For a fixed multiset w, the set of all wo^n is clearly definable by a regular formula I_w in REG. Theorem 5 essentially says that the emptiness of $[I_w \wedge \exists \diamond Halt]^M$ is undecidable. Hence, in contrast to Theorem 4, we have,

Corollary 1. *The* $\mathrm{CTL}_{-}^{\mathrm{REG}}$ *model-checking problem for (nondeterministic) bounded P systems is undecidable. The undecidability remains even for* $\mathrm{CTL}_{-}^{\mathrm{REG}}$ *formulas in the form of* INIT $\wedge \exists \diamond H$ *where* INIT *and* H *are regular formulas in* REG.

We have seen that the emptiness problem for deterministic bounded P systems is undecidable. We now look at a special case when the cardinality of the maximally parallel multiset of rules applicable at each step is at most 1. Thus the computation of the system would be sequential. More generally, consider a (nondeterministic) bounded P system whose computation is restricted in that at every step, only one nondeterministically selected rule is applied. Call such a system a sequential bounded P system. In contrast to Theorem 5, We show that the emptiness problem for sequential bounded P system is decidable. In fact, this result is true even if the system is not bounded, i.e., in the rules of the form $u \rightarrow v$, we no longer require that $|v| \leq |u|$. We can show that such a sequential P system is equivalent to a partially blind multicounter machine (PBCM). Note that a PBCM [6] can increment/decrement any counter by 1 or leave it unchanged; however, it can not test a counter for zero. When there is an attempt to decrement a zero counter, the machine gets stuck and the computation is aborted. The machine starts with the input counter set to a value n with all other counters set to zero. We say that the machine accepts if it eventually halts in an accepting state with *all* the counters zero.

It can be shown that a language $L \subseteq o^*$ is accepted by a sequential P system if and only if it is accepted by a PBCM. Since the emptiness problem for PBCMs is decidable (as this problem is reducible to the reachability problem for vector addition systems (i.e., Petri nets)) [6], we have:

Theorem 6. *The emptiness problem for sequential P systems (and, hence, also for sequential bounded P systems) is decidable.*

A BPS M is *separated* if for any two distinct rules $u_i \rightarrow v_i$ and $u_j \rightarrow v_j$ in M, the multiset union of u_i and v_i is disjoint with the multiset union of u_j and v_j. For instance, the system with rules $ab \rightarrow ae$ and $cd \rightarrow d$ is separated. But the system with rules $ab \rightarrow ae$ and $cd \rightarrow e$ is not. In contrast to Corollary 1, we have the following result. Currently, we do not know whether the result still holds when we modify the above "separated" definition into the following: for any two distinct rules $u_i \rightarrow v_i$ and $u_j \rightarrow v_j$ in M, multisets u_i and u_j are disjoint.

Theorem 7. *For separated bounded P systems, the model-checking problem for formulas in the form of* $\mathrm{INIT} \wedge \exists \diamond H$*, where* INIT *and* H *are regular formulas in* REG*, is decidable.*

Notice that separated systems can demonstrate nonlinear reachability relations. For instance, consider such a system M with rules $ea \rightarrow a$ and $ccb \rightarrow cbd$. Define INIT to be $\#(b) = 1 \wedge \#(a) = 1 \wedge \#(d) = 0$ and H to be $\#(e) > 0 \wedge \#(c) \geq 2$. Then, the set of all $V \in [H]^M$ that is reachable from some $U \in [\mathrm{INIT}]^M$ (i.e., $U \rightsquigarrow_M V$) is exactly the set of V satisfying the following nonlinear relation: $\#(e) > 0 \wedge \#(c) \geq 2 \wedge \#(a) = 2^{\#(d)}$. We believe that Theorem 7 can be generalized to the entire $\mathrm{CTL}^{\mathrm{REG}}$, further investigation of which will be left for the full version of the paper.

We now investigate the case when a BPS M is *bounded maximally parallel*; i.e., there is a constant K such that on every execution of M, every maximally parallel move only fires at most K instances of rules. Examples of such M include purely catalytic systems [13,14,5], and following the same ideas of the proof of Theorem 5 but using constructions in [13,14,5], one can show that simple reachability queries like formulas $\mathrm{INIT} \wedge \exists \diamond H$ in $\mathrm{CTL}^{\mathrm{REG}}$ are undecidable for these M's. To make the query decidable, we add one more restriction. A maximally parallel move from $u = (t_1, \ldots, t_k)$ (the vector representation of the multiset u) to $v = (s_1, \ldots, s_k)$ is 1-non-monotonic if $t_2 \leq s_2, \ldots, t_k \leq s_k$. M is 1-non-monotonic if its executions consist of 1-non-monotonic maximally parallel moves only. With this restriction, we can show that linear reachability queries are decidable:

Theorem 8. *For bounded maximally parallel and 1-non-monotonic BPSs, the model-checking problem for formulas in the form of* $\mathrm{INIT} \wedge \exists \diamond H$*, where* INIT *and* H *are linear formulas in* LIN*, is decidable.*

Let N be a constant. A configuration $u = (t_1, \ldots, t_k)$ is 1-unbounded if each of $t_2, \ldots, t_k$ is bounded by N (i.e., only the first t_1 is possibly larger than N). M is 1-unbounded if its executions consist of 1-unbounded configurations only. In this case, we can generalize Theorem 8 to the full $\mathrm{CTL}^{\mathrm{LIN}}$.

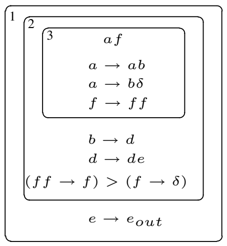
Theorem 9. *The* $\mathrm{CTL}^{\mathrm{LIN}}$ *model-checking problem for bounded maximally parallel and 1-unbounded BPSs is decidable.*

5 Experiments

From the results presented so far, even simple reachability queries like formulas $\mathrm{INIT} \wedge \exists \diamond H$ in $\mathrm{CTL}^{\mathrm{REG}}$ are undecidable for a bounded P system M in general. In this section, we investigate *conservative* behavior approximations that can be applied over M such that every execution of the approximated system is also an execution of the original M. Hence, such a conservative behavior approximation at least provides a way to help us analyze the original system, partially. This resembles similar approximation techniques in traditional model-checking of (in)finite state transition systems.

One such approximation is to let M to execute for at most B maximally parallel steps for a given constant B. Clearly, since B is a fixed, the reachability relation of M now is expressible as a Presburger formula, which can be calculated with a Presburger manipulator like Omega [12]. Another approximation is to force M to crash whenever M reaches a multiset with more than S objects for a given constant B. Under this approximation, M can be simulated by a finite-state transition system and, accordingly, tools like the LTL model-checker SPIN [7] can be used to analyze it. In fact, these two approximations are applicable to a general P system (which is not necessarily a BPS) with multi-membranes, priority rules and dissolving membranes. Below, we briefly report our experiences in using Omega and SPIN to conservatively analyze a general P system, which is taken from literature [11]. Since Omega (resp. SPIN) has been proved effective in handling even fairly large infinite (resp. finite) real-world applications [2,7], the primary purpose of our experiments is to identify whether these tools are also effective for a general P system with a reasonable size, where the inherent maximal parallelism makes the model highly nondeterministic, concurrent, and, more importantly, lack of control-flow structure.

The example P system M is shown in the figure below. It has three membranes where, in particular, membrane 2 (resp. membrane 3) are dissolved (i.e., objects in the membrane are immediately become objects in the outside membrane and the membrane along with the membrane's rules is all gone) whenever the rule in membrane 2 (resp. membrane 3) that contains δ fires. In membrane 2, the relation $ff \to f > f \to \delta$ says that, roughly, in a maximally parallel move, the former rule is given higher priority to fire than the latter rule. The P system is to compute a quadratic relation between certain objects; see [11] for details. Using Omega, we encode a maximally parallel move $\to_M$ in a Presburger relation which contains 34 variables (i.e., $\mathbf{N}^{17} \times \mathbf{N}^{17}$). Notice that a symbol may need up to three variables to represent, in order to specify its multiplicity in one of the three membranes. Additionally, a number of quantified variables are needed to encode the maximal parallelism, the priority rules and the dissolving membranes. Due to space limitation, we omit the detail of the Omega encoding. We used Omega to compute the reachability relation of M within B maximally parallel moves. Unfortunately, the tool crashed when computing with $B = 6$ (memory usage was 1.6GB including virtual memory), though it was successfully completed with $B = 5$ (in 489 CPU seconds).



To use SPIN, we encode M in Promela, the front-end specification language in SPIN. A Promela process is defined for each membrane, where the process exits when its corresponding membrane dissolves. Object-transfers across a membrane are simulated through rendezvous communications among processes, and the priority relation between evolution rules is implemented by carefully designed guards of the related selections. Again, we omit the detail of the Promela encoding. Using SPIN's default option, we checked the system for deadlock states. Unfortunately, SPIN could not finish any run within one hour as we varied the variable types from byte to short and long, respectively. Then, we checked a liveness property: eventually, the evolution of this P system will come to an end, i.e., only the skin membrane is left and no evolution rules in the skin can be applied; this is equivalent to checking that eventually all the three processes shall reach the ends of their bodies. Surprisingly, SPIN handled this property easily —

the total time consumed, as we varied the variable types from byte to short and long, increased merely from less than 0.1 second to several seconds and several minutes. The results of these checkings are all “false” since the inner membranes may not necessarily dissolve. Another property we checked about this P system is that: whenever the evolution of this P system comes to an end, the number of e objects outside the skin membrane is the square of the number of d objects inside the skin membrane. Again, SPIN gave the correct answer (“true”) fairly fast (in less than 1 second) for each of the three cases (byte, short, long).

Through these preliminary experiments, we prefer SPIN over Omega to serve as the back-end solver in a future P system model-checker. On the other hand, Omega has its own strength in handling infinite state systems. Still, more research is needed for both approximation methods to create a more efficient encoding. All our experiments were run on a PC server with two 1GHz PIII processors running Linux with 1GB physical memory. The encodings can be found in the long version of the paper, which is available at `www.eecs.wsu.edu/∼zdang`.

References

1. O. Andrei, G. Ciobanu, and D. Lucanu. Executable specifications of p systems. In *Proc. 5th Workshop on Membrane Computing*, pages 126–145, 2005.
2. T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. *ACM Trans. Program. Lang. Syst.*, 21(4):747–789, 1999.
3. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
4. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.
5. R. Freund, L. Kari, M. Oswald, and P. Sosik. Computationally universal P systems without priorities: two catalysts are sufficient. Available at *http://psystems.disco.unimib.it*, 2003.
6. S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7:311–324, 1978.
7. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
8. O. H. Ibarra. The number of membranes matters. In *Proc. 4th Workshop on Membrane Computing*, pages 218–231, 2003.
9. Gh. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
10. Gh. Paun. *Membrane Computing: An Introduction*. Springer-Verlag, 2002.
11. Gh. Paun and G. Rozenberg. A guide to membrane computing. *TCS*, 287(1):73–100, 2002.
12. W. Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 35(8):102–114, 1992.
13. P. Sosik. P systems versus register machines: two universality proofs. In *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Arges, Romania*, pages 371–382, 2002.
14. P. Sosik and R. Freund. P systems without priorities are computationally universal. In *WMC-CdeA2002*, volume 2597 of *LNCS*, pages 400–409. Springer, 2003.

Looking for Simple Common Schemes to Design Recognizer P Systems with Active Membranes That Solve Numerical Decision Problems

Carmen Graciani-Díaz and Agustín Riscos-Núñez

Dpto. Ciencias de la Computación e Inteligencia Artificial
{cgdiaz, ariscosn}@us.es

Abstract. Earlier solutions to decision problems by means of P systems used many counter objects to control the synchronization of different stages in a computation (usually as many counters as the stage must last in the worst case). In this paper we propose a way to replace those counters with some spacial objects for each stage. Furthermore, following the ideas presented in [1], in order to have a common scheme to attack numerical problems, all instances of a problem with the same size are solved by the same P system (which depends on the size) given an input which describes the corresponding instance of the problem. We illustrate these ideas with a cellular solution to the Subset-Sum problem.

1 Preliminaries

Since the introduction of P systems [3] a great amount of contributions in that field has been reported. In particular, many papers are devoted to solving decision or numerical NP–complete problems in polynomial time. In order to deal with such kind of problems, an exponential size workspace is generated (in the number of objects and the number of membranes). In this paper we deal with decision problems in the framework of P systems.

We recall that a *decision problem*, X, is a pair (I_X, θ_X) such that I_X is a language over a finite alphabet whose elements are called *instances* and θ_X is a boolean function over I_X. For an instance u of the problem X, if $\theta_X(u) = 1$ (resp. $\theta_X(u) = 0$) the answer of the problem for that instance is Yes (resp. No).

In the general definition, P systems are *non-deterministic*. Therefore they do not seem to be a suitable tool to solve a decision problem. For that reason a condition that restricts, in a certain way, the non-determinism is demanded. More specifically, we will work with *confluent* systems (all computations with the same initial configuration produce the same answer).

When working with P systems with *external output*, the user can ignore the inner processes and take only into account the objects that the system expels to the environment. To know when a computation halts, it is demanded that some *halting indicator* is sent to the environment exactly in the last step.

These restrictions make more difficult the design of such systems. Earlier approaches in this area used counter objects to control the synchronization of different stages in a computation. This kind of solutions need, therefore, extra objects and steps that are not necessary to obtain an answer but to control the procedure of obtaining it.

In this paper we want to show how this control can be obtained with only a few objects.

Furthermore, earlier solutions to NP-complete problems in polynomial time used to design one P system that solves one instance of the problem; therefore the system could not be used to solve any other instance of the problem, even if it was of the same size (see [8,2]). The introduction of P systems with *input* [6] gave rise to the design of families of systems, each of them able to solve all the instances of the problem of a given size.

Another goal of this paper is to present a solution of Subset-Sum problem with schemes of rules more uniform that depend only in the cardinality of the set.

The present work is a continuation of [4] and [1]. For this reason we have chosen the same problem and P system model: Subset-Sum problem and recognizer P systems with active membranes, respectively. The solution to Subset-Sum problem will illustrate also how the given schemes can be adapted for the new approach.

1.1 P Systems with Active Membranes

For a detailed description of a P system, $\Pi = (\Gamma, \mathsf{H}, \mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_p, \mathsf{R})$, with active membranes, we refer the reader to [2] and [7]. In what follows we briefly describe the rules of the model that will be used in next sections.

(a) $[_\mathsf{l}\, \mathsf{a} \rightarrow \mathsf{v}\,]_\mathsf{l}^\alpha$ (*evolution rules*), where $\mathsf{a} \in \Gamma$, $\mathsf{v} \in \Gamma^*$, $\alpha \in \{+,-,0\}$, $\mathsf{l} \in \mathsf{H}$. Substitutes an object a by a multiset of objects v in a membrane with label l and charge α.

(b) $[_\mathsf{l}\, \mathsf{a}\,]_\mathsf{l}^\alpha \rightarrow \mathsf{b}\, [_\mathsf{l}\ ]_\mathsf{l}^\beta$ (*communication rules*), where $\mathsf{a}, \mathsf{b} \in \Gamma$, $\alpha, \beta \in \{+,-,0\}$ and $\mathsf{l} \in \mathsf{H}$. Sends out (to its father) an object a from a membrane with label l and charge α transformed into the object b. In addition, the charge of the membrane changes to β.

(c) $\mathsf{a}\, [_\mathsf{l}\ ]_\mathsf{l}^\alpha \rightarrow [_\mathsf{l}\, \mathsf{b}\,]_\mathsf{l}^\beta$ (*communication rules*), where $\mathsf{a}, \mathsf{b} \in \Gamma$, $\alpha, \beta \in \{+,-,0\}$ and $\mathsf{l} \in \mathsf{H}$. An object a enters in a membrane with label l and charge α (from its father) transformed into the object b. In addition, the charge of the membrane changes to β.

(d) $[_\mathsf{l}\, \mathsf{a}\,]_\mathsf{l}^\alpha \rightarrow [_\mathsf{l}\, \mathsf{b}\,]_\mathsf{l}^\beta\, [_\mathsf{l}\, \mathsf{c}\,]_\mathsf{l}^\gamma$ (*division rules*), where $\mathsf{a}, \mathsf{b}, \mathsf{c} \in \Gamma$, $\alpha, \beta, \gamma \in \{+,-,0\}$ and $\mathsf{l} \in \mathsf{H}$. An object a divides a membrane with label l and charge α into two membranes with the same label and charges β and γ. In each of the new membranes the object a changes into objects b and c, respectively. This rule can only be applied to *elementary* membranes and never to the skin.

Rules of type (a) are applied as usual in the framework of P systems, that is, in a maximally parallel way. However, only one rule among the remaining types (b)–(d) can be applied to a membrane. The application of the rules is supposed to occur simultaneously (if division must take place in a membrane consider that the objects present in that membrane evolve previously). For a precise definition we refer the reader to [2] and [7].

1.2 P Systems with Input

A variant of P systems arises when considering the possibility of admitting external information before a computation starts.

A P system of degree p *with input* is a tuple $(\Pi, \Sigma, \mathsf{i}_\Pi)$ where:

- Π is a P system of degree p.
- Σ is an *input* alphabet strictly contained in the *work* alphabet, Γ.
- All the initial multisets are over the alphabet $\Gamma - \Sigma$.
- i_Π is a label that distinguishes the *input* membrane.

In a P system of degree p, with initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_p$, given a multiset of objects m over the input alphabet, the *initial configuration with input* m is the tuple $(\mu_\Pi, \mathcal{M}_1, \ldots, \mathcal{M}_{\mathsf{i}_\Pi} \cup \mathsf{m}, \ldots, \mathcal{M}_p)$. Let us denote by I_Π the set of all the possible input multisets.

1.3 P Systems with External Output

In this variant the environment collects the output of the computations, instead of an inner membrane.

There will be some special objects called *halting indicators.* A P system with external output is *valid* if no computation sends any halting indicator to the environment except in the last step. And that must only occurs if the computation is a halting one.

1.4 Language Recognizer P Systems

A *language recognizer* P system is a P system with input and external output such that the working alphabet contains two halting indicators yes and no. A language recognizer P system is *valid* and all its computations halt. If the object is yes (resp. no) we say that the computation is an *accepting* (resp. *rejecting*) one.

We say that $\{\Pi_n\}_{n\in\mathbb{N}}$ is a family of language recognizer P systems that solves, in polynomial time, a decision problem (I_X, θ_X) if it verifies the following properties:

- All the P systems in the family are language recognizers.
- There exists a deterministic Turing machine that constructs each member of the family, Π_n, from n in polynomial time.
- There exists a polynomial encoding for the set of instances, I_X, into the family of P systems, Π (that is, a pair of polynomial time computable functions $(\mathsf{cod}, \mathsf{s})$ where $\mathsf{cod}\colon \mathsf{I}_X \to \bigcup_{n\in\mathbb{N}} \mathsf{I}_{\Pi_n}$ and $\mathsf{s}\colon \mathsf{I}_X \to \mathbb{N}$ verifying $\mathsf{cod}(u) \in \mathsf{I}_{\Pi_{\mathsf{s}(u)}}$ for all $u \in \mathsf{I}_X$) such that:
  - A polynomial function, p, exists so that for each $u \in \mathsf{I}_X$ all the computations of the system $\Pi_{\mathsf{s(u)}}$ with input $\mathsf{cod}(u)$ halt at most in $\mathsf{p}(|u|)$ steps.

- For each $u \in \mathsf{I}_X$, if there exists an accepting computation in the system $\Pi_{\mathsf{s(u)}}$ with input $\mathsf{cod}(u)$, then $\theta_X(u) = 1$. It is said then that the family is sound.
- For each $u \in \mathsf{I}_X$, if $\theta_X(u) = 1$, then every computation in the system $\Pi_{\mathsf{s(u)}}$ with input $\mathsf{cod}(u)$ is an accepting one. It is said then that the family is complete.

The resolution of an instance $u \in \mathsf{I}_X$ by a family of P systems Π consist of two stages: during the first one (usually called pre-computation stage) we calculate $\mathsf{s}(u)$, $\mathsf{cod}(u)$ and $\Pi_{\mathsf{s}(u)}$; during the second stage the P system $\Pi_{\mathsf{s}(u)}$ with input $\mathsf{cod}(u)$ carries out its computation.

2 A Solution to the Subset-Sum Problem

We illustrate the previous discussion with a solution to Subset-Sum problem that can be stated as follows:

Given a finite set $\mathsf{A} = \{\mathsf{a}_1, \ldots, \mathsf{a}_n\}$, *a weight function* $\omega\colon \mathsf{A} \to \mathbb{N}$ *such that* $\omega(\mathsf{a}_i) = \omega_i$ *for* $i = 1, \ldots, n$, *and a constant* $k \in \mathbb{N}$, *determine whether or not there exists a subset* $\mathsf{D} \subseteq \mathsf{A}$ *such that* $\omega(\mathsf{D}) = k$.

The proposed solution is based on the one given at [4], and is divided into several stages:

– *Generation stage:* Elementary membrane divisions are carried out until obtaining a membrane associated with each subset of A.
– *Calculating stage:* In each membrane the weight of the associated subset is calculated. This stage will take place in parallel with the previous one.
– *Checking stage:* In each membrane it is verified if the weight of the associated subset is equal to the constant k. This stage begins in each membrane after the previous ones are over.
– *Output stage:* When the previous stage has been completed in all membranes, the system sends the corresponding answer to the environment and the computation halts.

For each $n \in \mathbb{N}$ (the cardinality of set $\mathsf{A} = \{\mathsf{a}_1, \ldots, \mathsf{a}_n\}$) a P system with active membranes, input and external output is defined as follows: $(\Pi_n, \Sigma_n, \mathsf{i}_{\Pi_n})$ where $\Pi_n = (\Gamma_n, \mathsf{H}, \mu_\Pi, \mathcal{M}_\mathsf{s}, \mathcal{M}_{\mathsf{e},n}, \mathcal{M}_\mathsf{r}, \mathsf{R}_n)$, P system of degree.

– Working alphabet: $\Gamma_n = \{\mathsf{x}_i \mid 0 \leq i \leq n\} \cup \{\#, \mathsf{yes}, \mathsf{no}, \overline{\mathsf{no}}, \mathsf{q}, \mathsf{q}_0, \mathsf{q}_1, \mathsf{q}_2, \mathsf{q}_3, \mathsf{c}, \mathsf{g}, \overline{\mathsf{g}}, \mathsf{d}\ , \mathsf{f}_0, \mathsf{f}, \mathsf{f}_+, \overline{\mathsf{b}}, \mathsf{b}, \overline{\mathsf{x}}_0, \overline{\mathsf{b}}_0, \mathsf{b}_0, \mathsf{z}, \mathsf{z}_0, \mathsf{z}_+, \overline{\mathsf{z}}, \mathsf{h}_0, \overline{\mathsf{h}}_1, \mathsf{h}_1, \mathsf{p}, \mathsf{t}\}$.
– Set of labels: $\mathsf{H} = \{\mathsf{s}, \mathsf{e}, \mathsf{r}\}$.
– Membrane structure: $\mu_\Pi = [_\mathsf{s} [_\mathsf{e}\ ]_\mathsf{e} [_\mathsf{r}\ ]_\mathsf{r}]_\mathsf{s}$.
– Initial multisets: $\mathcal{M}_\mathsf{s} = \overline{\mathsf{no}}$, $\mathcal{M}_{\mathsf{e},n} = \overline{\mathsf{g}}\ \mathsf{f}_0\ \mathsf{d}^n\ \mathsf{z}_0$ and $\mathcal{M}_\mathsf{r} = \overline{\mathsf{h}}_1$.
– Set of rules: R_n that consists of the following rules:

(a)
$$\begin{array}{lll}
[_\mathsf{e}\, \mathsf{f}_0\,]_\mathsf{e}^0 \to [_\mathsf{e}\, \mathsf{q}\,]_\mathsf{e}^- \, [_\mathsf{e}\, \mathsf{f}\,]_\mathsf{e}^+ & [_\mathsf{e}\, \mathsf{z}_0 \to \mathsf{z}\,]_\mathsf{e}^0 & [_\mathsf{e}\, \mathsf{z}_+ \to \mathsf{z}_0\,]_\mathsf{e}^0 \\
[_\mathsf{e}\, \mathsf{f}_+\,]_\mathsf{e}^0 \to [_\mathsf{e}\, \mathsf{f}_0\,]_\mathsf{e}^0 \, [_\mathsf{e}\, \mathsf{f}\,]_\mathsf{e}^+ & [_\mathsf{e}\, \mathsf{z}_0 \to \ ]_\mathsf{e}^+ & [_\mathsf{e}\, \mathsf{z}_+\,]_\mathsf{e}^+ \to \mathsf{z}\, [_\mathsf{e}\ ]_\mathsf{e}^+ \\
[_\mathsf{e}\, \mathsf{f} \to \mathsf{f}_+\,]_\mathsf{e}^+ & [_\mathsf{e}\, \mathsf{d}\,]_\mathsf{e}^+ \to \#\, [_\mathsf{e}\ ]_\mathsf{e}^0 & [_\mathsf{e}\, \mathsf{z} \to \mathsf{z}_+\,]_\mathsf{e}^+
\end{array}$$

The goal of these rules is the generation of one membrane for each subset of A. When an object f_0 is present in a neutrally charged membrane we pick a new element from A for its associated subset (summing its weight to the previous ones) and then divide the membrane. In the membranes where q appears no further objects will be added, and the charge of the membrane changes in order to activate the checking stage. The multiplicity of object d controls the number of divisions that must take place. The object z evolves in order to remain only in the last generated membrane, collaborating to control the beginning of the output stage.

(b) $[_{\mathsf{e}}\, \mathsf{x}_i \to \mathsf{x}_{i-1}\,]_{\mathsf{e}}^{+} \quad 1 \le i \le n$

$[_{\mathsf{e}}\, \mathsf{x}_0 \to \overline{\mathsf{x}}_0\,]_{\mathsf{e}}^{0}$ $\quad [_{\mathsf{e}}\, \overline{\mathsf{x}}_0 \to \overline{\mathsf{b}}_0\,]_{\mathsf{e}}^{0}$ $\quad [_{\mathsf{e}}\, \overline{\mathsf{x}}_0 \to\ ]_{\mathsf{e}}^{+}$

In the beginning, objects x_i, $1 \le i \le n$, are introduced encoding the weights of the corresponding elements of A. When the generation stage ends, the multiplicity of object $\overline{\mathsf{b}}_0$ will encode the weight of the subset associated with the membrane.

(c) $[_{\mathsf{e}}\, \mathsf{q} \to \mathsf{q}_0\,]_{\mathsf{e}}^{-}$ $\quad [_{\mathsf{e}}\, \overline{\mathsf{b}}_0 \to \mathsf{b}_0\,]_{\mathsf{e}}^{-}$ $\quad [_{\mathsf{e}}\, \overline{\mathsf{b}} \to \mathsf{b}\,]_{\mathsf{e}}^{-}$

These rules mark the beginning of the checking stage in a membrane. Now, the multiplicity of object b_0 encode the weight of the corresponding subset of A and the multiplicity of object b encode the value of the constant k.

$[_{\mathsf{e}}\, \overline{\mathsf{g}}\,]_{\mathsf{e}}^{-} \to \overline{\mathsf{g}}\, [_{\mathsf{e}}\ ]_{\mathsf{e}}^{-}$

Object $\overline{\mathsf{g}}$ will be used to mark the beginning of the output stage.

(d) $[_{\mathsf{e}}\, \mathsf{b}_0\,]_{\mathsf{e}}^{-} \to \#\, [_{\mathsf{e}}\ ]_{\mathsf{e}}^{+}$ $\quad [_{\mathsf{e}}\, \mathsf{b}\,]_{\mathsf{e}}^{+} \to \#\, [_{\mathsf{e}}\ ]_{\mathsf{e}}^{-}$

We compare the number of occurrences of objects b_0 and b sending them out alternatively.

$[_{\mathsf{e}}\, \mathsf{q}_0 \to \mathsf{q}_1\,]_{\mathsf{e}}^{-}$ $\quad [_{\mathsf{e}}\, \mathsf{q}_1 \to \mathsf{q}_0\,]_{\mathsf{e}}^{+}$ $\quad [_{\mathsf{e}}\, \mathsf{q}_1 \to \mathsf{q}_2\ \mathsf{c}\,]_{\mathsf{e}}^{-}$

$[_{\mathsf{e}}\, \mathsf{c}\,]_{\mathsf{e}}^{-} \to \#\, [_{\mathsf{e}}\ ]_{\mathsf{e}}^{+}$ $\quad [_{\mathsf{e}}\, \mathsf{q}_2 \to \mathsf{q}_3\,]_{\mathsf{e}}^{+}$

Objects q_i and c control if both objects have been actually sent out or not (if there is an excess or lack of any of them).

$[_{\mathsf{e}}\, \mathsf{q}_3\,]_{\mathsf{e}}^{+} \to \mathsf{yes}\, [_{\mathsf{e}}\ ]_{\mathsf{e}}^{0}$ $\quad [_{\mathsf{e}}\, \mathsf{q}_3\,]_{\mathsf{e}}^{-} \to \#\, [_{\mathsf{e}}\ ]_{\mathsf{e}}^{0}$ $\quad [_{\mathsf{e}}\, \mathsf{q}_0\,]_{\mathsf{e}}^{+} \to \#\, [_{\mathsf{e}}\ ]_{\mathsf{e}}^{0}$

These rules deal with the different checking results.

(e) $[_{\mathsf{s}}\, \mathsf{z} \to \overline{\mathsf{z}}\ \overline{\mathsf{z}}\,]_{\mathsf{s}}^{0}$ $\quad \overline{\mathsf{z}}\, [_{\mathsf{r}}\ ]_{\mathsf{r}}^{0} \to [_{\mathsf{r}}\, \overline{\mathsf{z}}\,]_{\mathsf{r}}^{0}$ $\quad [_{\mathsf{r}}\, \overline{\mathsf{z}} \to \mathsf{p}\,]_{\mathsf{r}}^{0}$

Object z controls the beginning of a process in membrane r that will trigger the output stage. When z appears in membrane s 2^n objects $\overline{\mathsf{g}}$ are present in it.

$[_{\mathsf{s}}\, \overline{\mathsf{z}}\,]_{\mathsf{s}}^{0} \to \#\, [_{\mathsf{s}}\ ]_{\mathsf{s}}^{+}$ $\quad [_{\mathsf{s}}\, \overline{\mathsf{g}} \to \mathsf{g}\,]_{\mathsf{s}}^{+}$ $\quad \mathsf{g}\, [_{\mathsf{e}}\ ]_{\mathsf{e}}^{0} \to [_{\mathsf{e}}\, \mathsf{g}\,]_{\mathsf{e}}^{+}$

When a membrane ends its checking stage it admits one object g.

(f) $\mathsf{g}\, [_{\mathsf{r}}\ ]_{\mathsf{r}}^{+} \to [_{\mathsf{r}}\, \mathsf{g}\,]_{\mathsf{r}}^{-}$ $\quad [_{\mathsf{r}}\, \mathsf{h}_1 \to \mathsf{h}_0\,]_{\mathsf{r}}^{+}$ $\quad [_{\mathsf{r}}\, \mathsf{h}_0 \to \mathsf{h}_1\,]_{\mathsf{r}}^{-}$

$[_{\mathsf{r}}\, \mathsf{p}\,]_{\mathsf{r}}^{-} \to \mathsf{p}\, [_{\mathsf{r}}\ ]_{\mathsf{r}}^{0}$ $\quad [_{\mathsf{r}}\, \mathsf{g}\,]_{\mathsf{r}}^{0} \to \mathsf{g}\, [_{\mathsf{r}}\ ]_{\mathsf{r}}^{-}$ $\quad \mathsf{p}\, [_{\mathsf{r}}\ ]_{\mathsf{r}}^{-} \to [_{\mathsf{r}}\, \mathsf{p}\,]_{\mathsf{r}}^{+}$

$[_{\mathsf{r}}\, \mathsf{h}_0\,]_{\mathsf{r}}^{+} \to \mathsf{t}\, [_{\mathsf{r}}\ ]_{\mathsf{r}}^{+}$ $\quad [_{\mathsf{r}}\, \overline{\mathsf{h}}_1 \to \mathsf{h}_1\,]_{\mathsf{r}}^{+}$

We will use membrane r to detect when all objects g have been admitted in a membrane e. That will mean that the checking stage has finished in all membranes and then, the output stage is triggered.

(g) $[_{\mathsf{s}}\, \mathsf{t}\,]_{\mathsf{s}}^{+} \to \#\, [_{\mathsf{s}}\ ]_{\mathsf{s}}^{-}$ $\quad [_{\mathsf{s}}\, \mathsf{yes}\,]_{\mathsf{s}}^{-} \to \mathsf{yes}\, [_{\mathsf{s}}\ ]_{\mathsf{s}}^{0}$

$[_{\mathsf{s}}\, \overline{\mathsf{no}} \to \mathsf{no}\,]_{\mathsf{s}}^{-}$ $\quad [_{\mathsf{s}}\, \mathsf{no}\,]_{\mathsf{s}}^{-} \to \mathsf{no}\, [_{\mathsf{s}}\ ]_{\mathsf{s}}^{0}$

The presence of object t in membrane s activates the answering process. If there is any object yes then it must be sent out. Otherwise, an object no goes out.

(h) Also, some cleaning can be done during the process.
$[_{\mathsf{e}}\, \mathsf{x}_i \to\ ]_{\mathsf{e}}^{-}$ $\quad 1 \le i \le n \quad$ $[_{\mathsf{e}}\, \mathsf{z} \to\ ]_{\mathsf{e}}^{-}$ $\quad$ $[_{\mathsf{e}}\, \mathsf{d} \to\ ]_{\mathsf{e}}^{-}$
$[_{\mathsf{e}}\, \mathsf{b} \to\ ]_{\mathsf{e}}^{0}$ $\quad$ $[_{\mathsf{e}}\, \mathsf{b}_0 \to\ ]_{\mathsf{e}}^{0}$

– Input alphabet: $\Sigma_n = \{\overline{\mathsf{b}}\} \cup \{\mathsf{x}_i \mid 1 \le i \le n\}$.
– Input membrane: $\mathsf{i}_\Pi = \mathsf{e}$.

So we have defined a family of P systems $\{\Pi_n\}_{n\in\mathbb{N}}$. Each of the members of the family, Π_n, solves all the instances of the Subset-Sum problem for a finite set A with cardinality n. Each instance will be determined by the values of the weight function, ω_i for $i = 1, \ldots, n$, and the value of the constant k. The set of possible input multisets is $\mathsf{I}_{\Pi_n} = \{\overline{\mathsf{b}}^k\ \mathsf{x}_1^{\omega_1} \ldots \mathsf{x}_n^{\omega_n} \mid k, \omega_1, \ldots, \omega_n \in \mathbb{N}\}$. As we can see, all the members of the family can be constructed by a Turing machine in polynomial time from n.

Let us consider $\mathsf{I}_X = \{(n, (\omega_1, \ldots, \omega_n), k) \mid n, \omega_1, \ldots, \omega_n, k \in \mathbb{N}\}$ (all the instances of the Subset-Sum problem). The pair of functions (cod, s) defined by $\mathsf{cod}(n, (\omega_1, \ldots, \omega_n), k) = \overline{\mathsf{b}}^k\ \mathsf{x}_1^{\omega_1} \ldots \mathsf{x}_n^{\omega_n}$ and $\mathsf{s}(n, (\omega_1, \ldots, \omega_n), k) = n$ is a polynomial encoding of I_X into $\{\Pi_n\}_{n\in\mathbb{N}}$.

The following data gives us an idea of Π_n complexity:

– Size of the working alphabet: $n + 31 \in O(n)$.
– Number of membranes: $3 \in O(1)$.
– $|\mathcal{M}_{\mathsf{s}}| + |\mathcal{M}_{\mathsf{e,n}}| + |\mathcal{M}_{\mathsf{r}}| = n + 5 \in O(n)$.
– Input size: $k + \omega(\mathsf{A})$
– Number of rules: $2n + 48$
– Number of computation steps needed in the worst case: $3n + 2\min(k, \omega(\mathsf{A})) + 19$

In what follows we will prove that the systems of the family are recognizer P systems that solve the Subset-Sum problem in linear time; that is, that the family is sound, complete, and polynomially bounded.

3 Formal Verification

Proposition 1. *Consider $k, n \in \mathbb{N}$ and a weight function $\omega\colon A \to \mathbb{N}$ such that $\omega(a_i) = \omega_i$ for $i = 1, \ldots, n$. For any $l \in \mathbb{N}$ and i, $1 \le i \le n$, if l is the weight of a subset $D \subseteq \{a_1, \ldots, a_{i-1}\}$, then from a membrane of the following form $[_{\mathsf{e}}\, \overline{\mathsf{b}}^k\ \overline{\mathsf{g}}\ \mathsf{f}_+\ \mathsf{d}^{n-i}\ \overline{\mathsf{b}}_0^l\ \mathsf{x}_0^{\omega_i} \cdots \mathsf{x}_{n-i}^{\omega_n}\ ]_{\mathsf{e}}^{0}$ we obtain the set of membranes*

$\{[_{\mathsf{e}}\, \mathsf{b}^k\ \mathsf{q}_0\ \mathsf{b}_0^{l'}\ ]_{\mathsf{e}}^{-} \mid$ where l' is the weight of $D \cup D'$ for $\emptyset \ne D' \subseteq \{a_i, \ldots, a_n\}\}$
They will be called relevant membranes.

The last membrane of this set will be generated after $3(n-i+1)$ steps. During the process $2^{n-i+1} - 1$ objects $\overline{\mathsf{g}}$ will appear in membrane s.

Moreover, we also obtain the following set of membranes:

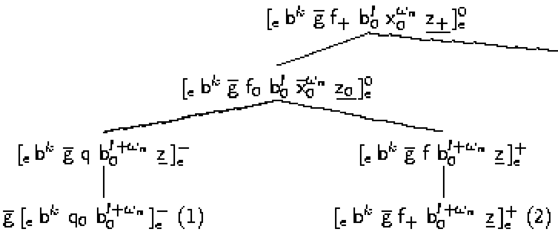
$$\{[_{\mathsf{e}}\, \overline{\mathsf{b}}^k\ \overline{\mathsf{g}}\ \mathsf{f}_+\ \overline{\mathsf{b}}_0^{l'}\ ]_{\mathsf{e}}^{+} \mid l' = \omega(D \cup D') \text{ for } D' \subseteq \{a_i, \ldots, a_n\}\}$$

These membranes will be called irrelevant and the last one will be generated after $3(n-i+1)$ steps.

If an object z_+ is present in the considered membrane, then it will only remain, as an object z, in the last generated irrelevant membrane.

Proof: By decreasing induction on i, starting from $i = n$.

Figure 1 shows the evolution of a membrane $[_{e}\,\overline{b}^k\ \overline{g}\ f_+\ \overline{b}_0^l\ x_0^{\omega_n}\ ]_e^0$ where l is the weight of $D \subseteq \{a_1, \dots, a_{n-1}\}$. The branching represents new generated membranes obtained by division.



$[_e\, b^k\ \overline{g}\ f_+\ b_0^l\ x_0^{\omega_n}\ \underline{z_+}]_e^0$
$[_e\, b^k\ \overline{g}\ f_0\ b_0^l\ \overline{x}_0^{\omega_n}\ \underline{z_0}]_e^0$
$[_e\, b^k\ \overline{g}\ f\ b_0^l\ \overline{x}_0^{\omega_n}\ \underline{z_0}]_e^+$
$[_e\, b^k\ \overline{g}\ q\ b_0^{l+\omega_n}\ \underline{z}]_e^-$
$[_e\, b^k\ \overline{g}\ f\ b_0^{l+\omega_n}\ \underline{z}]_e^+$
$[_e\, b^k\ \overline{g}\ f_+\ b_0^l]_e^+$ (3)
$\overline{g}\,[_e\, b^k\ q_0\ b_0^{l+\omega_n}]_e^-$ (1)
$[_e\, b^k\ \overline{g}\ f_+\ b_0^{l+\omega_n}\ \underline{z}]_e^+$ (2)

Fig. 1. Case $i = n$

In node (1) we can see that the relevant membrane $[_{e}\,b^k\ q_0\ b_0^{l+\omega_n}\ ]_e^-$ (where $l + \omega_n = \omega(D \cup \{a_n\})$) is obtained after $3 = 3(n - n + 1)$ steps. Besides, in the last step $1 = 2^{n-n+1} - 1$ object $\overline{g}$ has been sent to membrane s.

In (2) the irrelevant membrane $[_{e}\,\overline{b}^k\ \overline{g}\ f_+\ \overline{b}_0^{l+\omega_n}\ ]_e^+$ (where $l+\omega_n$ is the weight of $D \cup \{a_n\}$) is obtained after $3 = 3(n - n + 1)$ steps.

In (3) we have obtained the irrelevant membrane $[_{e}\,\overline{b}^k\ \overline{g}\ f_+\ \overline{b}_0^l\ ]_e^+$, after 2 steps ($l$ is the weight of $D = D \cup \emptyset$).

Moreover, Figure 1 shows (underlined) the evolution of an object z_+ when it is present in the initial membrane and we can see that it only remains, as an object z, in the last obtained irrelevant membrane.

Thus, the proposition holds for $i = n$.

Induction step: $i + 1 \rightarrow i$

The evolution of $[_{e}\,\overline{b}^k\ \overline{g}\ f_+\ d^{n-i}\ \overline{b}_0^l\ x_0^{\omega_i}\ \cdots\ x_{n-i}^{\omega_n}\ ]_e^0$ where l is the weight of a subset $D \subseteq \{a_1, \dots, a_{i-1}\}$ is shown in Figure 2.

In (1) the relevant membrane $[_{e}\,b^k\ q_0\ b_0^{l+\omega_i}\ ]_e^-$ is obtained. In it $l + \omega_i$ is the weight of $D \cup \{a_i\}$. In the last step one object $\overline{g}$ appears in membrane s.

In (2), a membrane $[_{e}\,\overline{b}^k\ \overline{g}\ f_+\ d^{n-(i+1)}\ \overline{b}_0^{l+\omega_i}\ x_0^{\omega_{i+1}}\ \cdots\ x_{n-(i+1)}^{\omega_n}\ ]_e^0$, in which l is the weight of $D \cup \{a_i\} \subseteq \{a_1, \dots, a_i\}$, is obtained.

By induction hypothesis, from this membrane we obtain the set of membranes $\{[_{e}\,b^k\ q_0\ b_0^{l'}\ ]_e^- \mid l' = \omega((D \cup \{a_i\}) \cup D')$ for $\emptyset \neq D' \subseteq \{a_{i+1}, \dots, a_n\}\}$. The last member of this set will be generated after $3(n - i) + 3 = 3(n - i + 1)$ steps. During this process $2^{n-i} - 1$ objects $\overline{g}$ will appear in membrane s.

In addition, the set of irrelevant membranes $\{[_{e}\,\overline{b}^k\ \overline{g}\ f_+\ \overline{b}_0^{l'}\ ]_e^+ \mid l'$ is the weight of $(D \cup \{a_i\}) \cup D'$ for $D' \subseteq \{a_{i+1}, \dots, a_n\}\}$ is obtained. The last member of this set will be generated after $3(n - i) + 3 = 3(n - i + 1)$ steps.

In (3), a membrane $[_{e}\,\overline{b}^k\ \overline{g}\ f_+\ d^{n-(i+1)}\ \overline{b}_0^l\ x_0^{\omega_{i+1}}\ \cdots\ x_{n-(i+1)}^{\omega_n}\ ]_e^0$ in which l is the weight of D is obtained.

By induction hypothesis, from this membrane the set of relevant membranes $\{[_{e}\,b^k\ q_0\ b_0^{l'}\ ]_e^- \mid l' = \omega(D \cup D')$ for $\emptyset \neq D' \subseteq \{a_{i+1}, \dots, a_n\}\}$ is generated. The

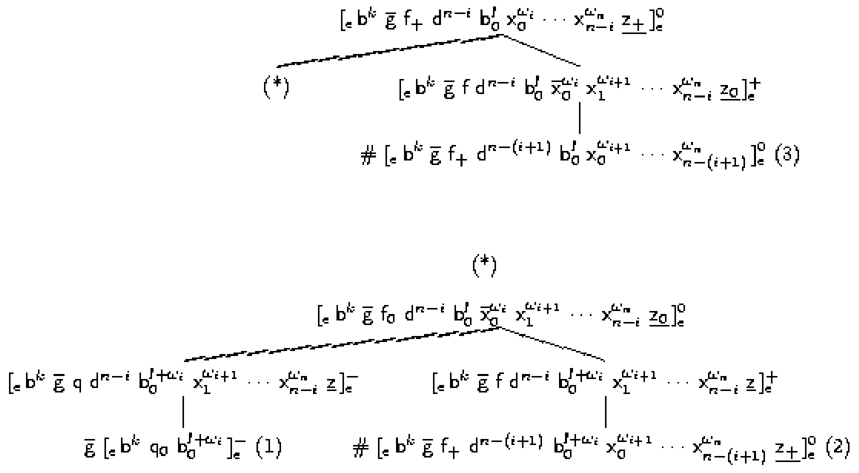

$[_e\, \overline{b}^k\, \overline{g}\, f_+\, d^{n-i}\, \overline{b}_0^l\, x_0^{\omega_i} \cdots x_{n-i}^{\omega_n}\, \underline{z_+}\,]_e^0$
(*)
$[_e\, \overline{b}^k\, \overline{g}\, f\, d^{n-i}\, \overline{b}_0^l\, \overline{x}_0^{\omega_i}\, x_1^{\omega_{i+1}} \cdots x_{n-i}^{\omega_n}\, \underline{z_0}\,]_e^+$
$\#\, [_e\, \overline{b}^k\, \overline{g}\, f_+\, d^{n-(i+1)}\, \overline{b}_0^l\, x_0^{\omega_{i+1}} \cdots x_{n-(i+1)}^{\omega_n}\,]_e^0$ (3)
(*)
$[_e\, \overline{b}^k\, \overline{g}\, f_0\, d^{n-i}\, \overline{b}_0^l\, \overline{x}_0^{\omega_i}\, x_1^{\omega_{i+1}} \cdots x_{n-i}^{\omega_n}\, \underline{z_0}\,]_e^0$
$[_e\, \overline{b}^k\, \overline{g}\, q\, d^{n-i}\, \overline{b}_0^{l+\omega_i}\, x_1^{\omega_{i+1}} \cdots x_{n-i}^{\omega_n}\, \underline{z}\,]_e^-$
$[_e\, \overline{b}^k\, \overline{g}\, f\, d^{n-i}\, \overline{b}_0^{l+\omega_i}\, x_1^{\omega_{i+1}} \cdots x_{n-i}^{\omega_n}\, \underline{z}\,]_e^+$
$\overline{g}\, [_e\, b^k\, q_0\, b_0^{l+\omega_i}\,]_e^-$ (1)
$\#\, [_e\, \overline{b}^k\, \overline{g}\, f_+\, d^{n-(i+1)}\, \overline{b}_0^{l+\omega_i}\, x_0^{\omega_{i+1}} \cdots x_{n-(i+1)}^{\omega_n}\, \underline{z_+}\,]_e^0$ (2)

Fig. 2. Induction step $i + 1 \to i$

last member of this set will be generated after $3(n - i) + 2 = 3(n - i + 1) - 1$ steps and during the process $2^{n-i} - 1$ objects $\overline{\mathsf{g}}$ will appear in membrane s. From (3), it is also generated the set of irrelevant membranes $\{[_{\mathsf{e}}\, \overline{\mathsf{b}}^k\, \overline{\mathsf{g}}\, \mathsf{f}_+\, \overline{\mathsf{b}}_0^{l'}\,]_{\mathsf{e}}^+ \mid l'$ is the weight of $\mathsf{D} \cup \mathsf{D'}$ for $\mathsf{D'} \subseteq \{\mathsf{a}_{i+1}, \ldots, \mathsf{a}_n\}\}$. The last member of this set will be generated after $3(n - i) + 3 = 3(n - i + 1)$ steps.

Thus, from a membrane $[_{\mathsf{e}}\, \overline{\mathsf{b}}^k\, \overline{\mathsf{g}}\, \mathsf{f}_+\, \mathsf{d}^{n-i}\, \overline{\mathsf{b}}_0^l\, \mathsf{x}_0^{\omega_i} \cdots \mathsf{x}_{n-i}^{\omega_n}\,]_{\mathsf{e}}^0$ we will obtain the set of relevant membranes $\{[_{\mathsf{e}}\, \mathsf{b}^k\, \mathsf{q}_0\, \mathsf{b}_0^{l'}\,]_{\mathsf{e}}^- \mid l'$ is the weight of $\mathsf{D} \cup \mathsf{D'}$ for a subset $\emptyset \neq \mathsf{D'} \subseteq \{\mathsf{a}_i, \ldots, \mathsf{a}_n\}\}$ (the last member after $3(n - i + 1)$ steps). During the process $2(2^{n-i} - 1) + 1 = 2^{n-i+1} - 1$ objects $\overline{\mathsf{g}}$ appear in membrane s.

Besides, we obtain the set of irrelevant membranes $\{[_{\mathsf{e}}\, \overline{\mathsf{b}}^k\, \overline{\mathsf{g}}\, \mathsf{f}_+\, \overline{\mathsf{b}}_0^{l'}\,]_{\mathsf{e}}^+ \mid l'$ is the weight of $\mathsf{D} \cup \mathsf{D'}$ for $\mathsf{D'} \subseteq \{\mathsf{a}_i, \ldots, \mathsf{a}_n\}\}$ (the last member after $3(n - i + 1)$ steps).

Finally, if an object z_+ is present in the initial membrane (underlined in Figure 2) it only appears in (2), then by induction hypothesis it will only remain, as an object z, in the last generated irrelevant membrane. □

Theorem 2. *Given k, $n \in \mathbb{N}$ and a weight function $\omega\colon A \to \mathbb{N}$, $\omega(\mathsf{a}_i) = \omega_i$ for $i = 1, \ldots, n$ from a membrane of the form $[_{\mathsf{e}}\, \overline{\mathsf{b}}^k\, \overline{\mathsf{g}}\, \mathsf{f}_0\, \mathsf{d}^n\, \mathsf{x}_1^{\omega_1} \cdots \mathsf{x}_n^{\omega_n}\, \mathsf{z}_0\,]_{\mathsf{e}}^0$ the set of relevant membranes $\{[_{\mathsf{e}}\, \mathsf{b}^k\, \mathsf{q}_0\, \mathsf{b}_0^l\,]_{\mathsf{e}}^- \mid l = \omega(D)$ for $D \subseteq A\}$ is obtained (last one after $3n + 2$ steps). During the process 2^n objects $\overline{\mathsf{g}}$ appear in membrane s.*

The set of irrelevant membranes $\{[_{\mathsf{e}}\, \overline{\mathsf{b}}^k\, \overline{\mathsf{g}}\, \mathsf{f}_+\, \overline{\mathsf{b}}_0^l\,]_{\mathsf{e}}^+ \mid l = \omega(D)$ for $D \subseteq A\}$ is also obtained (the last of them after $3n + 2$ steps). The object z_0 will evolve to an object z that will only remain in the last generated irrelevant membrane.

Proof:
Figure 3 shows the evolution of $[_{\mathsf{e}}\, \overline{\mathsf{b}}^k\, \overline{\mathsf{g}}\, \mathsf{f}_0\, \mathsf{d}^n\, \mathsf{x}_1^{\omega_1} \cdots \mathsf{x}_n^{\omega_n}\, \mathsf{z}_0\,]_{\mathsf{e}}^0$.

In (1) the relevant membrane $[_{\mathsf{e}}\, \mathsf{b}^k\, \mathsf{q}_0\,]_{\mathsf{e}}^-$ is obtained and during the process an object $\overline{\mathsf{g}}$ appears in membrane s.

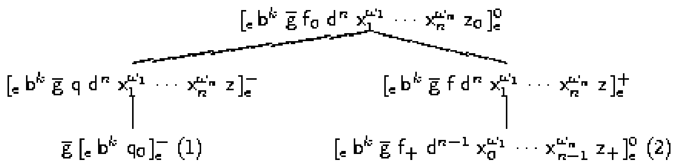

$[_e\, b^k\, \overline{g}\, f_0\, d^n\, x_1^{\omega_1} \cdots x_n^{\omega_n}\, z_0\,]_e^0$
$[_e\, b^k\, \overline{g}\, q\, d^n\, x_1^{\omega_1} \cdots x_n^{\omega_n}\, z\,]_e^-$
$[_e\, b^k\, \overline{g}\, f\, d^n\, x_1^{\omega_1} \cdots x_n^{\omega_n}\, z\,]_e^+$
$\overline{g}\, [_e\, b^k\, q_0\,]_e^-$ (1)
$[_e\, b^k\, \overline{g}\, f_+\, d^{n-1}\, x_0^{\omega_1} \cdots x_{n-1}^{\omega_n}\, z_+\,]_e^0$ (2)

Fig. 3. Evolution scheme

In (2) we have $[_e\, \overline{b}^k\ \overline{g}\ f_+\ d^{n-1}\ x_0^{\omega_1} \cdots x_{n-1}^{\omega_n}\ ]_e^0$, case $i = 1$ and $l = 0$ of proposition 1, and from it we will obtain the set $\{[_e\, b^k\ q_0\ b_0^l\ ]_e^- \mid l = \omega(D)$ for $\emptyset \neq D \subseteq A\}$ of relevant membranes (the last one after $3(n-1+1)+2 = 3n+2$ steps) and $2^{n-1+1} - 1 = 2^n - 1$ objects $\overline{g}$ in membrane s.

Moreover, from (2) (by Proposition 1) we will also obtain the set of irrelevant membranes $\{[_e\, \overline{b}^k\ \overline{g}\ f_+\ \overline{b}_0^l\ ]_e^+ \mid l = \omega(D)$ for $D \subseteq A\}$ (the last one after $3(n-1+1)+2 = 3n+2$ steps.

Finally, as an object z_+ appears in (2), from the evolution of z_0 (again by Proposition 1) it will only remain as an object z in the last generated irrelevant membrane. □

Let us see now, given $0 \leq m \leq \min(k, l)$, the evolution of a relevant membrane of the form: $[_e\, q_0\ b^{k-m}\ b_0^{l-m}\ ]_e^-$

(a) Case: $m = k$, $m = l$
$[_e\, q_0\,]_e^- \Rightarrow [_e\, q_1\,]_e^- \Rightarrow [_e\, q_2\ c\,]_e^- \Rightarrow \#\ [_e\, q_2\,]_e^+ \Rightarrow [_e\, q_3\,]_e^+ \Rightarrow \text{yes}\ [_e\ ]_e^0$

(b) Case: $m < k$, $m = l$
$[_e\, q_0\ b^{k-m}\ ]_e^- \Rightarrow [_e\, q_1\ b^{k-m}\ ]_e^- \Rightarrow [_e\, q_2\ c\ b^{k-m}\ ]_e^- \Rightarrow \#\ [_e\, q_2\ b^{k-m}\ ]_e^+ \Rightarrow [_e\, q_3\ b^{k-(m+1)}\ ]_e^- \Rightarrow \#\ [_e\, b^{k-(m+1)}\ ]_e^0$ (objects b will be consumed in the following step).

(c) Case: $m = k$, $m < l$
$[_e\, q_0\ b_0^{l-m}\ ]_e^- \Rightarrow \#\ [_e\, q_1\ b_0^{l-(m+1)}\ ]_e^+ \Rightarrow [_e\, q_0\ b_0^{l-(m+1)}\ ]_e^+ \Rightarrow$
$\#\ [_e\, b_0^{l-(m+1)}\ ]_e^0$ (objects b_0 will be consumed in the following step).

(d) Case: $m < k$, $m < l$
$[_e\, q_0\ b^{k-m}\ b_0^{l-m}\ ]_e^- \Rightarrow \#\ [_e\, q_1\ b^{k-m}\ b_0^{l-(m+1)}\ ]_e^+ \Rightarrow$
$\#\ [_e\, q_0\ b^{k-(m+1)}\ b_0^{l-(m+1)}\ ]_e^-$ (that will continue evolving as shown)

Beginning with $m = 0$, pairs of objects b_0 and b are sent out (case (d)) until both of them are finished (case (a)) or a lack of any of them is detected (cases (b) and (c)).

At the end, the membrane will be neutrally charged and ready to admit one object g, after that it will remain inactive.

The only irrelevant membrane that continues evolving is the one with an object z_+ that will be sent out (so this object will appear in membrane s after the last object $\overline{g}$ has also appeared).

Object z in membrane s will activate the evolution of membrane r:

$\mathsf{z}\,[_{\mathsf{r}}\,\overline{\mathsf{h}}_1\,]^0_{\mathsf{r}} \Rightarrow \bar{\mathsf{z}}\,\bar{\mathsf{z}}\,[_{\mathsf{r}}\,\overline{\mathsf{h}}_1\,]^0_{\mathsf{r}} \Rightarrow [_{\mathsf{r}}\,\overline{\mathsf{h}}_1\,\bar{\mathsf{z}}\,]^+_{\mathsf{r}}$ (also an object $\bar{\mathsf{z}}$ is sent out membrane s changing its charge to positive) $\Rightarrow [_{\mathsf{r}}\,\mathsf{h}_1\;\mathsf{p}\,]^+_{\mathsf{r}}$ (in membrane s each object $\bar{\mathsf{g}}$ evolves to an object g).

In membrane r begins now a process to detect if there is any object g in membrane s (remember that 2^n relevant membranes are at the checking stage and that when they finish they will admit one object g). Figure 4 shows a scheme of the process.

When this process finishes an object t has appeared in membrane s and two cases can take place: there is an object yes in membrane s or there is not.

(a) $[_{\mathsf{s}}\,\mathsf{t}\;\mathsf{yes}\;\overline{\mathsf{no}}\,]^+_{\mathsf{s}} \Rightarrow [_{\mathsf{s}}\,\mathsf{yes}\;\overline{\mathsf{no}}\,]^-_{\mathsf{s}} \Rightarrow \mathsf{yes}\,[_{\mathsf{s}}\,\mathsf{no}\,]^0_{\mathsf{s}}$
(b) $[_{\mathsf{s}}\,\mathsf{t}\;\overline{\mathsf{no}}\,]^+_{\mathsf{s}} \Rightarrow [_{\mathsf{s}}\,\overline{\mathsf{no}}\,]^-_{\mathsf{s}} \Rightarrow [_{\mathsf{s}}\,\mathsf{no}\,]^-_{\mathsf{s}} \Rightarrow \mathsf{no}\,[_{\mathsf{s}}\;]^0_{\mathsf{s}}$

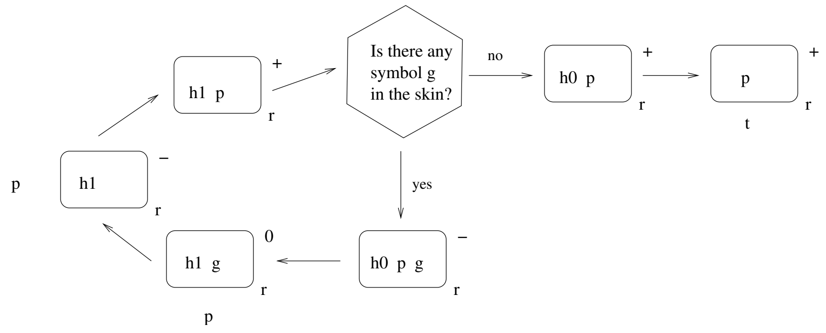


Fig. 4. Checking the existence of objects g in the skin

4 Conclusions

In this paper we have presented a family of recognizer P systems solving the Subset-Sum problem. Given a "size", n, member Π_n of the family solves all the instances of the SubsetSum problem for a finite set A with cardinal n. Each instance is determined by the value of k and the values of the weight function, ω_i for $i = 1, \ldots, n$.

This solution is more uniform than the one presented in [4], as the construction of the family only depends on n, the cardinality of A, and not on the parameter k.

On the other hand, we remark that the rules that control the generation, checking, and output stages depend neither on n nor on k. Only the number of rules that handle objects x_i (corresponding to the different elements of A) is determined by n.

Another interesting point is that the number of computation steps, in the cases where $\omega(\mathsf{A}) < k$, is smaller than in the solution given in [4].

This solution has also been adapted to other numerical NP-complete problems as the Knapsack or the Partition problems. Moreover, different approaches and skeleton designs to solve those problems have been studied. For example, in order to obtain 2^n membranes (each of them representing a subset of A) there is another solution that only generates those membranes (in the solution given in this paper, another 2^n irrelevant membranes are also generated).

Those results allow us to be optimistic about describing a "language" for the design of P systems to solve relevant numerical problems and, why not, other kinds of problems.

Another issue related to the present paper is the computer simulation of P systems. An implementation *in silico* (in CLIPS) for P systems with active membranes has been developed by the Research Group on Natural Computing from the University of Seville [5]. This simulation has helped us to debug some errors in the formal design and verification of P systems, and a feedback process also exists, as running simulations of already verified P systems can detect possible bugs in the implementation.

The CLIPS code of the simulator, some instructions of use and some examples (including the problem presented in this paper) are available on the Web at `http://www.gcn.us.es`.

References

1. M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, and A. Ricos-Núñez. Towards a programming language in cellular computing. *Electronic Notes in Theoretical Commputer Science*, 123(1):93–110, 2005.
2. Gh. Păun. *Membrane Computing: An Introduction.* Springer-Verlag, Berlin, 2002.
3. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. and Turku Center for Computer Science-TUCS Report No 208.
4. M. J. Pérez-Jiménez and A. Riscos-Núñez. Solving the subset-sum problem by p systems with active membranes. *New Generation Computing, Springer–Verlag, Tokyo*, to appear.
5. M. J. Pérez-Jiménez and F. J. Romero-Campero. A CLIPS simulator for recognizer p systems with active membranes. In Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, and F. Sancho-Caparrini, editors, *Proceedings of the Second Brainstorming Week on Membrane Computing*, RGNC Report (01/04), pages 387–413, 2004.
6. M. J. Pérez-Jiménez, Á. Romero-Jiménez, and F. Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–285, September 2003.
7. A. Riscos-Núñez. *Programación celular: resolución eficiente de problemas numéricos NP-completos.* PhD thesis, University of Seville, 2004.
8. C. Zandron. *A Model for Molecular Computing: Membrane Systems.* PhD thesis, Universit degli Studi di Milano, 2001.

P Systems with Active Membranes, Without Polarizations and Without Dissolution: A Characterization of P

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez,
Agustín Riscos-Núñez, and Francisco J. Romero-Campero

Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
{magutier, marper, ariscosn, fran}@us.es

Abstract. We study the computational efficiency of recognizer P systems with active membranes without polarizations and without dissolution. The main result of the paper is the following: the polynomial computational complexity class associated with the class of recognizer P systems is equal to the standard complexity class **P**.

1 Introduction

The *theory of computation* deals with the *mechanical solvability* of problems, that is, searching solutions that can be described by a finite sequence of elementary processes or instructions. The first goal of this theory is general problem solving; that is, develop principles and methods that are able to solve any problem from a certain class of questions.

A *computational model* tries to capture those aspects of mechanical solutions of problems that are relevant to these solutions, including their inherent limitations. In some sense, we can think that computational models design machines according to certain necessity.

If we have a mechanically solvable problem and we have a specific algorithm solving it that can be implemented in a real machine, then it is very important to know how much computational resources (time or memory) are required for a given instance, in order to recognize the limitations of the real device.

Thus, one of the main goals of the *theory of computational complexity* is the study of the efficiency of algorithms and their data structures through the analysis of the resources required for solving problems (that is, according to their intrinsic computational difficulty). This theory provides a classification of the *abstract problems* that allows us to detect their inherent complexity from the computational solutions point of view.

Many interesting problems of the real world are presumably intractable and hence it is not possible to execute algorithmic solutions in an electronic computer when we deal with instances of those problems whose size is large. The theoretical limitations of the Turing machines in terms of computational power are also practical limitations to the digital computers.

Natural Computing is a new computing area inspired by nature, using concepts, principles and mechanisms underlying natural systems. *Evolutionary Computation* uses computational models of evolutionary processes as key elements in the design and implementation of computer–based problem solving systems [18]. *Neural Networks* are inspired in the structures of the brain and nervous system. *DNA Computing* is based on the computational potential of DNA molecules and on the capacity to handle them. *Membrane Computing* is inspired by the structure and functioning of living cells, and it is a cross-disciplinary field with contributions by computer scientists, biologists, formal linguists and complexity theoreticians, enriching each others with results, open problems and promising new research lines.

This emergent branch of Natural Computing was introduced by Gh. Păun in [8]. Since then it has received important attention from the scientific community. In fact, Membrane Computing has been selected by the Institute for Scientific Information, USA, as a fast *Emerging Research Front* in Computer Science, and [6] was mentioned in [19] as a highly cited paper in October 2003.

This new non-deterministic model of computation starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called *P systems.*

Roughly speaking, a P system consists of a cell-like membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous non-deterministic maximally parallel manner[1].

Inspired in living cells, P systems abstract the way of obtaining new membranes. These processes are basically two: *mitosis* (membrane division) and *autopoiesis* (membrane creation). Both ways of generating new membranes have given rise to different variants of P systems: *P systems with active membranes*, where the new workspace is generated by membrane division, and *P systems with membrane creation*, where the new membranes are created from objects.

Both models are universal from a computational point of view, but technically, they are pretty different. In fact, nowadays there does not exist any theoretical result which proves that these models can simulate each other in polynomial time.

P systems with active membranes have been successfully used to design solutions to well-known **NP**-complete problems, as SAT [16], Subset Sum [13], Knapsack [14], Bin Packing [15] and Partition [3], but as Gh. Păun pointed in [10] *"membrane division was much more carefully investigated than membrane creation as a way to obtain tractable solutions to hard problems"*. Recently, the first results related to the power and design of algorithms to solve NP problems in these model have arisen (see [4,5]).

P systems with active membranes were introduced in [7] with the membranes having polarizations, one of the "electrical charges" $0, -, +$, and several times the problem was formulated whether or not these polarizations are necessary in

[1] A layman-oriented introduction can be found in [9] and further bibliography at [20].

order to obtain polynomial solutions to **NP**–complete problems. The last current result is that from [1], where one proves that two polarizations suffice.

The present paper is both a contribution to this problem, and a contribution to another interesting problem in membrane computing, namely, of characterizing classic complexity classes, such as **P** and **NP**, by means of membrane computing complexity classes.

Specifically, we prove that **P** is equal to the family of problems which can be solved in a polynomial time by P systems with membrane division, without polarizations and *without dissolution*. At this moment, we do not know whether this last condition can be avoided, but either result would be of a great interest: if our result would remain true also when using membrane dissolution, then we would have the possitive answer to the problem of removing polarization; the other possibility would indicate a surprising role of the –apparently "innocent"– operation of membrane dissolution, as it will make the difference between efficiency and non–efficiency for P systems with membrane division and without polarization.

2 Preliminaries

2.1 The Reachability Problem

The Reachability Problem is the following: *given a (directed or undirected) graph, G, and two nodes a, b, determine whether or not the node b is reachable from a, that is, whether or not there exists a path in the graph from a to b.*

This problem belongs to the complexity class **P**. Indeed, it is very easy to design an algorithm running in polynomial time solving it. For example, given a (directed or undirected) graph, G, and two nodes a, b, we consider a depth–first–search with source a, and we check if b is in the tree of the computation forest whose root is a. The total running time of this algorithm is $O(|V| + |E|)$, that is, in the worst case is quadratic in the number of nodes. Morover, this algorithm needs to store a linear number of items (it can be proved that there exists another polynomial time algorithm which uses $O(\log^2(|V|))$ space).

2.2 Recognizer P Systems

In the structure and functioning of a cell, biological *membranes* play an essential role. The cell is separated from its environment by means of a *skin membrane*, and it is internally compartmentalized by means of *internal membranes.*

The main *syntatic* ingredients of a cell–like membrane system (P system) are the *membrane structure*, the *multisets*, and the *evolution rules.*

– A *membrane structure* consists of several membranes arranged hierarchically inside a main membrane (the *skin*), and delimiting *regions* (the space in–between a membrane and the immediately inner membranes, if any). Each membrane identifies a region inside the system. A membrane structure can be considered as a rooted tree.

– Regions defined by a membrane structure contain objects corresponding to chemical substances present in the compartments of a cell. The objects can be described by symbols or by strings of symbols, in such a way that *multiset of objects* are placed in the regions of the membrane structure.
– The objects can evolve according to given *evolution rules*, associated with the regions (hence, with the membranes).

The *semantics* of the cell–like membrane systems is defined through a non–deterministic and synchronous model (a global clock is assumed) as follows:

– A *configuration* of a cell–like membrane system consists of a membrane structure and a family of multisets of objects associated with each region of the structure. At the beginning, there is a configuration called the *initial configuration* of the system.
– In each time unit we can transform a given configuration in another configuration by applying the evolution rules to the objects placed inside the regions of the configurations, in a non–deterministic, and maximally parallel manner (the rules are chosen in a non–deterministic way, and in each region all objects that can evolve must do it). In this way, we get *transitions* from one configuration of the system to the next one.
– A *computation* of the system is a (finite or infinite) sequence of configurations such that each configuration –except the initial one– is obtained from the previous one by a transition.
– A computation which reaches a configuration where no more rules can be applied to the existing objects, is called a *halting computation.*
– The result of a halting computation is usually defined through the multiset associated with a specific output membrane (or the environment) in the final configuration.

That is, a computation in a P system is structured as follows: it starts with the initial configuration of the system, then the computation proceeds, and when it stops the result is to be found in the output membrane.

In this paper we use membrane computing as a framework to attack the resolution of decision problems. In order to solve this kind of problems and having in mind the relationship between the solvability of a problem and the recognition of the language associated with it, we consider P systems as *recognizer language* devices.

Definition 1. *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where: (a)* Π *is a P system, with working alphabet* Γ*, with* p *membranes labelled by* $1, \ldots, p$*, and initial multisets* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *associated with them; (b)* Σ *is an (input) alphabet strictly contained in* Γ *and the initial multisets are over* $\Gamma - \Sigma$*; (c)* i_Π *is the label of a distinguished (input) membrane.*

The computations of a P system with input in the form of a multiset over Σ are defined in a natural way, but the initial configuration of (Π, Σ, i_Π) must be the initial configuration of the system Π to which we add the input multiset.

Definition 2. *Let (Π, Σ, i_Π) be a P system with input. Let Γ be the working alphabet of Π, μ the membrane structure, and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of Π. Let m be a multiset over Σ. The* initial configuration of (Π, Σ, i_Π) with input m *is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$.*

Let (Π, Σ, i_Π) be a P system with input. Let Γ be the working alphabet of Π, μ the membrane structure and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of Π. Let m be a multiset over Σ. Then we denote $\mathcal{M}_j^* = \{(a, j) : a \in \mathcal{M}_j\}$, for $1 \leq j \leq p$, and $m^* = \{(a, i_\Pi) : a \in m\}$.

Let us recall that a decision problem X is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a predicate (a total boolean function) over I_X.

Definition 3. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer P systems with input. A* polynomial encoding *from X to $\mathbf{\Pi}$ is a pair (cod, s) of polynomial time computable functions over I_X such that for each instance $w \in I_X$, $s(w)$ is a natural number and $cod(w)$ is an input multiset for the system $\Pi(s(w))$.*

It is easy to prove that polynomial encodings are stable under polynomial time reductions.

Proposition 1. *Let X_1, X_2 be decision problems. Let r be a polynomial time reduction from X_1 to X_2. Let (cod, s) be a polynomial encoding from X_2 to $\mathbf{\Pi}$. Then $(cod \circ r, s \circ r)$ is a polynomial encoding from X_1 to $\mathbf{\Pi}$.*

Definition 4. *A* recognizer P system *is a P system with input and external output such that:*

1. *The working alphabet contains two distinguished elements* yes *and* no.
2. *All computations halt.*
3. *If $\mathcal{C}$ is a computation of the system, then either object* yes *or object* no *(but not both) must have been released into the environment, and only in the last step of the computation.*

In recognizer P systems, we say that a computation $\mathcal{C}$ is an *accepting computation* (respectively, *rejecting computation*) if the object *yes* (respectively, *no*) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$. Hence, these devices send to the environment an accepting or rejecting answer, in the end of their computations.

2.3 A Polynomial Complexity Class in Recognizer P Systems

Definition 5. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer P systems* with input. *Let (cod, s) be a polynomial encoding from X to $\mathbf{\Pi}$.*

– *We say that the family* $\mathbf{\Pi}$ *is sound with regard to* (X, cod, s) *if the following is true: for each instance of the problem* $w \in I_X$*, if there exists an accepting computation of* $\Pi(s(w))$ *with input* $cod(w)$*, then* $\theta_X(w) = 1$.
– *We say that the family* $\mathbf{\Pi}$ *is complete with regard to* (X, cod, s) *if the following is true: for each instance of the problem* $u \in I_X$*, if* $\theta_X(u) = 1$ *then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting computation.*

Next, we propose to solve a decision problem through a family of P systems constructed in polynomial time by a Turing machine, and verifying that each element of the family processes, in a specified sense, all the instances of *equivalent size*. We say that these solutions are *uniform solutions*.

Definition 6. *Let* $\mathcal{R}$ *be a class of recognizer P systems with input membrane. A decision problem* $X = (I_X, \theta_X)$ *is solvable in polynomial time by a family* $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$*, of P systems from* $\mathcal{R}$*, and we denote this by* $X \in \mathbf{PMC}_{\mathcal{R}}$*, if the following is true:*

– *The family* $\mathbf{\Pi}$ *is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system* $\Pi(w)$ *from the instance* $w \in I_X$.
– *There exists a polynomial encoding* (cod, s) *from* I_X *to* $\mathbf{\Pi}$ *such that*
  - *The family* $\mathbf{\Pi}$ *is polynomially bounded with regard to* (X, cod, s)*, that is, there exists a polynomial function* p*, such that for each* $u \in I_X$ *every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is halting and, moreover, it performs at most* $p(|u|)$ *steps.*
  - *The family* $\mathbf{\Pi}$ *is sound and complete with regard to* (X, cod, s).

It is easy to see that the class $\mathbf{PMC}_{\mathcal{R}}$ is closed under polynomial–time reduction and complement (see [11] for details).

3 Recognizer P Systems with Active Membranes Without Polarizations and Without Dissolution

A particularly interesting class of cell–like membrane systems are the systems with active membranes, where the membrane division can be used in order to solve computationally hard problems, e.g., **NP**-complete problems, in polynomial or even linear time, by a space–time trade-off.

In this paper we work with a variant of P systems with active membranes that does not use electrical charges or dissolution rules.

Definition 7. *A recognizer P system with active membranes without polarizations and without dissolution is a recognizer P system* (Π, Γ, i_Π)*, where the rules of the associated P system are of the following forms:*

(a) $[a \to u]_h$ *for* $h \in H$*,* $a \in \Gamma$*,* $u \in \Gamma^*$*: This is an object evolution rule, associated with a membrane labelled with h: an object* $a \in \Gamma$ *belonging to that membrane evolves to a string* $u \in \Gamma^*$.

(b) $a\,[\ ]_h \rightarrow [\,b\,]_h$ *for* $h \in H$, $a, b \in \Gamma$*: An object from the region immediately outside a membrane labelled with h is introduced in this membrane, possibly transformed into another object.*

(c) $[\,a\,]_h \rightarrow b\ [\ ]_h$ *for* $h \in H$, $a, b \in \Gamma$*: An object is sent out from membrane labelled with h to the region immediately outside, possibly transformed into another object.*

(d) $[\,a\,]_h \rightarrow [\,b\,]_h\ [\,c\,]_h$ *for* $h \in H$, $a, b, c \in \Gamma$*: An elementary membrane can be divided into two membranes with the same label, possibly transforming some objects.*

These rules are applied according to the following principles:

– All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non–deterministic way), but any object which can evolve by one rule of any form, must evolve.
– If at the same time a membrane labelled by h is divided by a rule of type (d) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. Of course, this process takes only one step.
– The rules associated with membranes labelled by h are used for all copies of this membrane. At one step, a membrane can be the subject of *only one* rule of types $(b), (c), (d)$.

In this framework we work without cooperation, without priorities, with cell division rules for elementary membranes, and without changing the labels of membranes.

We denote by $\mathcal{AM}^0_{-d}$ the class of all recognizer P systems with active membranes without polarizations and without dissolution.

4 Dependency Graph of a Recognizer P System with Active Membranes

Let Π be a recognizer P systems with active membranes without polarizations and without dissolution. Let R be the set of rules associated with Π.

Each rule can be considered, in a certain sense, as a *dependency* between the object triggering the rule and the object or objects produced by its application.

We can consider a general format of all kinds of rules of such systems as follows: $(a, h) \rightarrow (a_1, h')(a_2, h') \ldots (a_s, h')$, according to the following criterion:

– The rules of type (a) correspond to the case $h = h'$ and $s \geq 1$.
– The rules of type (b) correspond to the case $h = f(h')$ and $s = 1$.
– The rules of type (c) correspond to the case $h' = f(h)$ and $s = 1$.
– The rules of type (d) correspond to the case $h = h'$ and $s = 2$.

If h is the label of a membrane, then $f(h)$ (respectively, $ch(h)$) denotes the label of the father (resp. a child) of the membrane labelled by h. We adopt the convention that the father of the skin membrane is the environment.

For example, let us consider a general rule $(a,h) \to (a_1,h')(a_2,h')\dots(a_s,h')$. Then we can interpret that from the object a in membrane labelled by h we can *reach* the objects $a_1,\dots,a_s$ in membrane labelled by h'.

Next, we formalize these ideas in the following definition.

Definition 8. *Let Π be a recognizer P systems with active membranes without polarizations and without dissolution. Let R be the set of rules associated with Π. The dependency graph associated with Π is the directed graph $G_\Pi = (V_\Pi, E_\Pi)$ defined as follows:*

$$
\begin{aligned}
V_\Pi &= VL_\Pi \cup VR_\Pi,\\
VL_\Pi &= \{(a,h) \in \Gamma \times H:\ \exists u \in \Gamma^*\ ([a \to u]_h \in R) \vee\\
&\qquad \exists b \in \Gamma\ ([a]_h \to [\ ]_h b \in R) \vee\\
&\qquad \exists b \in \Gamma\ \exists h' = ch(h)\ (a[\ ]_{h'} \to [b]_{h'} \in R) \vee\\
&\qquad \exists b, c \in \Gamma\ ([a]_h \to [b]_h[c]_h \in R)\},\\
VR_\Pi &= \{(b,h) \in \Gamma \times H:\ \exists a \in \Gamma\ \exists u \in \Gamma^*\ ([a \to u]_h \in R \wedge b \in alph(u)) \vee\\
&\qquad \exists a \in \Gamma\ \exists h' = ch(h)\ ([a]_{h'} \to [\ ]_{h'} b \in R) \vee\\
&\qquad \exists a \in \Gamma\ (a[\ ]_h \to [b]_h \in R) \vee\\
&\qquad \exists a, c \in \Gamma\ ([a]_h \to [b]_h[c]_h \in R)\},\\
E_\Pi &= \{((a,h),(b,h')) : \exists u \in \Gamma^*\ ([a \to u]_h \in R \wedge b \in alph(u) \wedge h = h') \vee\\
&\qquad ([a]_h \to [\ ]_h b \in R\ \wedge\ h' = f(h)) \vee\\
&\qquad (a[\ ]_{h'} \to [b]_{h'} \in R\ \wedge h\ = f(h')) \vee\\
&\qquad \exists c \in \Gamma\ ([a]_h \to [b]_h[c]_h \in R \wedge h = h')\}.
\end{aligned}
$$

Proposition 2. *Let Π be a recognizer P systems with active membranes without polarizations and without dissolution. There exists a Turing machine that constructs the dependency graph, G_Π, associated with Π, in polynomial time (that is, in a time bounded by a polynomial function depending on the total number of rules and the maximum length of the rules).*

Proof. A deterministic algorithm that, given a P system Π with the set R of rules, constructs the corresponding dependency graph, is the following:

```
Input: $(\Pi, R)$
$V_\Pi \leftarrow \emptyset$; $E_\Pi \leftarrow \emptyset$
for each rule $r \in R$ of $\Pi$ do
  if $r = [a \to u]_h \wedge \mathtt{alph}(u) = \{a_1, \dots, a_s\}$ then
    $V_\Pi \leftarrow V_\Pi \cup \bigcup_{j=1}^{s} \{(a,h),(a_j,h)\}$; $E_\Pi \leftarrow E_\Pi \cup \bigcup_{j=1}^{s} \{((a,h),(a_j,h))\}$
  if $r = [a]_h \to [\ ]_h b$ then
    $V_\Pi \leftarrow V_\Pi \cup \{(a,h),(b,f(h))\}$;
    $E_\Pi \leftarrow E_\Pi \cup \{((a,h),(b,f(h)))\}$
  if $r = a[\ ]_h \to [b]_h$ then
    $V_\Pi \leftarrow V_\Pi \cup \{(a,f(h)),(b,h)\}$;
    $E_\Pi \leftarrow E_\Pi \cup \{((a,f(h)),(b,h))\}$
  if $r = [a]_h \to [b]_h[c]_h$ then
    $V_\Pi \leftarrow V_\Pi \cup \{(a,h)),(b,h),(c,h)\}$;
    $E_\Pi \leftarrow E_\Pi \cup \{((a,h)),(b,h)),((a,h),(c,h))\}$
```

The running time of this algorithm is bounded by $O(|R| \cdot q)$, where q is the value $\max\{length(r) : r \in R\}$. □

Proposition 3. *Let $\Pi = (\Gamma, \Sigma, H, \mathcal{M}_1, \dots, \mathcal{M}_p, R_1, \dots, R_p, i_\Pi)$ be a recognizer P systems with active membranes without polarizations and without dissolution. Let Δ_Π be defined as follows:*

$$\Delta_\Pi = \{(a,h) \in \Gamma \times H : \textit{there exists a path (within the dependency graph)} \\ \textit{from } (a,h) \textit{ to } (\mathtt{yes}, \textit{environment})\}$$

Then, there exists a Turing machine that constructs the set Δ_Π in polynomial time (that is, through a polynomial function depending on the total number of rules and the maximum length of the rules).

Proof. We can construct the set Δ_Π from Π as follows:

– We construct the dependency graph G_Π associated with Π.
– Then we consider the following algorithm:

```
Input: $G_\Pi = (V_\Pi, E_\Pi)$
  $\Delta_\Pi \leftarrow \emptyset$
  for each $(a,h) \in V_\Pi$ do
    if **reachability** $(G_\Pi, (a,h), (\mathtt{yes}, environment)) = \mathtt{yes}$ then
      $\Delta_\Pi \leftarrow \Delta_\Pi \cup \{(a,h)\}$
```

The running time of this algorithm is of the order $O(|V_\Pi| \cdot |V_\Pi|^2)$, hence it is of the order $O(|\Gamma|^3 \cdot |H|^3)$. □

Proposition 4. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ be a family of recognizer P systems with input membrane solving X, according to Definition 6. Let (cod, s) be the polynomial encoding associated with that solution. Then, for each instance w of the problem X the following assertions are equivalent:*

(a) $\theta_X(w) = 1$ (that is, the answer to the problem is `yes` *for w).*

(b) $\Delta_{\Pi(s(w))} \cap ((cod(w))^* \cup \bigcup_{j=1}^{p} \mathcal{M}_j^*) \neq \emptyset$, where $\mathcal{M}_1, \dots, \mathcal{M}_p$ are the initial multisets of the system $\Pi(s(w))$.

Proof. Let $w \in I_X$. Then $w \in L_X$ if and only if there exists an accepting computation of the system $\Pi(s(w))$ with input multiset $cod(w)$. But this condition is equivalent to the following: in the initial configuration of $\Pi(s(w))$ with input multiset $cod(w)$ there exists an object $a \in \Gamma$ in a membrane labelled by h such that in the dependency graph the node $(\texttt{yes}, environment)$ is reachable from (a, h).

Hence, $\theta_X(w) = 1$ if and only if $\Delta_{\Pi(s(w))} \cap \mathcal{M}_1^* \neq \emptyset$, or $\dots$, or $\Delta_{\Pi(s(w))} \cap \mathcal{M}_p^* \neq \emptyset$, or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$. □

Theorem 1. $\mathbf{PMC}_{\mathcal{AM}^0_{-d}} = \mathbf{P}$.

Proof. We have $\mathbf{P} \subseteq \mathbf{PMC}_{\mathcal{AM}^0_{-d}}$ because the class $\mathbf{PMC}_{\mathcal{AM}^0_{-d}}$ is closed under polynomial time reduction. Next, we show that $\mathbf{PMC}_{\mathcal{AM}^0_{-d}} \subseteq \mathbf{P}$. For that, let $X \in \mathbf{PMC}_{\mathcal{AM}^0_{-d}}$. Let $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ a family of recognizer P systems with input membrane solving X, according to Definition 6. Let (cod, s) be the polynomial encoding associated with that solution.

We consider the following deterministic algorithm:

```
Input: An instance $w$ of $X$
  - Construct the system $\Pi(s(w))$ with input multiset $cod(w)$.
  - Construct the dependency graph $G_{\Pi(s(w))}$ associated with $\Pi(s(w))$.
  - Construct the set $\Delta_{\Pi(s(w))}$ according to Proposition 3
    $answer \leftarrow$ No; $j \leftarrow 1$
    while $j \leq p \ \wedge \ answer = No$ do
        if $\Delta \cap M_j^* \neq \emptyset$ then
            $answer \leftarrow$ yes
        $j \leftarrow j + 1$
    endwhile
    if $\Delta \cap (cod(w))^* \neq \emptyset$ then
        $answer \leftarrow$ yes
```

On one hand, the answer of this algorithm is `yes` if and only if there exists a pair (a, h) belonging to $\Delta_{\Pi(s(w))}$ such that in the membrane labelled by h in the initial configuration (with input the multiset $cod(w)$) appears the symbol a.

On the other hand, a pair (a, h) belongs to $\Delta_{\Pi(s(w))}$ if and only if there exists a path from (a, h) to $(\texttt{yes}, \mathit{environment})$; that this, if and only if we can obtain an accepting computation of $\Pi(s(w))$ with input $\mathit{cod}(w)$. Hence, the algorithm above described solves the problem X.

The cost to determine whether or not $\Delta \cap M_j^* \neq \emptyset$ (or $\Delta \cap (\mathit{cod}(w))^* \neq \emptyset$) is of the order $O(|\Gamma|^2 \cdot |H|^2)$.

Hence, the running in time of this algorithm can be bounded as $f(|w|) + O(|R| \cdot q) + O(p \cdot |\Gamma|^2 \cdot |H|^2)$, where f is the (total) cost of a polynomial encoding from X to $\mathbf{\Pi}$, R the set of rules of Π, and $q = \max\ \{\mathit{length}(r) :\ r \in R\}$. But from Definition 6 we have that all involved parameters are polynomials in $|w|$. That is, the algorithm is polynomial in the size $|w|$ of the input. $\square$

5 Conclusions

Dependency graphs associated with a variant of recognizer P systems with active membranes are introduced. This concept allows us to characterize the accepting computations of these systems through the reachability of a distinguished node of the graph from other nodes associated with the initial configuration.

In this paper, we have showed that in the framework of P systems with active membranes if we remove electrical charges and dissolution, then it is possible to solve in polynomial time only problems which are tractable in the standard sense.

But what happens if in this framework we consider dissolution rules? Will be possible to solve **NP**–complete problems? If the answer is yes, then this result will provide a negative answer to the P–conjecture ($\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0}$, where $\mathcal{AM}^0$ is the class of all recognizer P systems with active membranes, without polarization, using dissolution rules and cell division rules only for elementary membranes).

Acknowledgement

The authors wish to acknowledge the support of the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds.

References

1. A. Alhazov, R. Freund, Gh. Păun: P systems with active membranes and two polarizations. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, 2004, 20–35.
2. M.R. Garey, D.S. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.
3. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A fast P system for finding a balanced 2-partition. *Soft Computing*, in press.

4. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving SAT with membrane creation. *CiE 2005: New Computational Paradigms, Amsterdan* (S. Barry Cooper, B. Lowe, L. Torenvliet, eds.), Report ILLC X-2005-01, University of Amsterdam, 82–91.
5. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution for QSAT with Membrane Creation. Submitted, 2005.
6. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3 (2002), 295–305
7. Gh. Păun, Computing with membranes: Attacking **NP**–complete problems. In *Unconventional Models of Computation* (I. Antoniou, C.S. Calude, M.J. Dinneen, eds.), 2000, 94–115.
8. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
9. Gh. Păun, M.J. Pérez-Jiménez: Recent computing models inspired from biology: DNA and membrane computing. *Theoria*, 18, 46 (2003), 72–84.
10. Gh. Păun: Further open problems in membrane computing. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds.), Report RGNC 01/04, University of Seville, 2004, 354–365.
11. M.J. Pérez–Jiménez: An approach to computational complexity in Membrane Computing. En G. Mauri, Gh. Păun, M. J. Pérez-Jiménez, Gr. Rozenberg, A. Salomaa (eds.) *Membrane Computing, 5th International Workshop, WMC5, Revised Selected and Invited Papers*. LNCS 3365 (2005), 85-109.
12. M.J. Pérez–Jiménez, A. Riscos–Núñez: A linear time solution to the Knapsack problem using active membranes. *Membrane Computing* (C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, eds.). LNCS 2933 (2004), 250–268.
13. M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset-Sum problem by active membranes. *New Generation Computing*, in press.
14. M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear solution for the Knapsack problem using active membranes. In *Membrane Computing* (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.). LNCS 2933 (2004), 250–268.
15. M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving the BIN PACKING problem by recognizer P systems with active membranes. *Proceedings of the Second Brainstorming Week on Membrane Computing* (Gh. Păun, A. Riscos, A. Romero and F. Sancho, eds.), Report RGNC 01/04, University of Seville, 2004, 414–430.
16. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: A polynomial complexity class in P systems using membrane division. *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003* (E. Csuhaj-Varjú, C. Kintala, D. Wotschke, G. Vaszil, eds.), 2003, 284-294.
17. P. Sosik: The computational power of cell division. *Natural Computing*, 2, 3 (2003), 287–298.
18. W.M. Spears: *Evolutionary Algorithms. The role of mutation and recombination.* Natural Computing Series, Springer, 2000.
19. ISI web page `http://esi-topics.com/erf/october2003.html`
20. P systems web page `http://psystems.disco.unimib.it/`

Discrete State Transition Systems on Continuous Space-Time: A Theoretical Model for Amorphous Computing

Masami Hagiya

Department of Computer Science,
Graduate School of Information Science and Technology,
University of Tokyo and NTT Communication Science Laboratories

Abstract. This paper deals with a kind of hybrid system that is obtained by making cellular automata infinitely small. Each point on a continuous space is defined as a state transition system with a finite number of discrete states that changes its state according to its previous state and the states of its neighbors. Time is also defined as continuous, so that state transitions may propagate through the space continuously over time. Discrete state transitions can be made instantaneously, as in the case of ordinary hybrid systems. It turned out that even defining the global behaviors that satisfy local transition rules is not trivial. The framework that we propose here can be regarded as a framework for amorphous computing.

1 Introduction

A hybrid system has a set of continuous parameters that may change continuously over time and affect its discrete state transitions [1]. This paper deals with another kind of hybrid system, one that is obtained by making cellular automata infinitely small. Each point on a continuous space, such as two-dimensional Euclidean space, is defined as a state transition system with a finite number of discrete states. It changes its state according to its previous state and the states of its neighbors. Time is also defined as continuous, so that state transitions may propagate through the space continuously over time.

Using the method called *ultradiscretization*, some cellular automata can be obtained from partial differential equations [6]. In ultradiscretization, space and time in a partial differential equation are first discretized. Then, the functional value is discretized to yield the corresponding cellular automata. Since the discretization of the functional value is possible only after space and time are discretized, it appears difficult to capture the kind of hybrid system mentioned above using ultradiscretization.

Berec classifies individual-based models for population dynamics according to whether population size, space, and time are discrete or continuous [3]. He also gives a model in which population size is discrete, while space and time are continuous. However, individuals are countable and scattered over space. In this paper, all points in the space are considered automata. Therefore, individuals are densely distributed over space in contrast to Berec's classification.

In this paper, we try to define hybrid systems of the above kind in the style of ordinary state transition systems, i.e., by giving transition rules for each point. As in the case of ordinary hybrid systems, discrete state transitions can be made instantaneously. Due to this kind of transition and the form of transition rules, it transpires that even defining the global behaviors that satisfy local transition rules is not trivial.

The framework that we propose here can be regarded as a framework for amorphous computing [2]. In amorphous computing, *computational particles* are scattered over a continuous space in an amorphous fashion, and communicate with each other locally through message passing.

A neural network of densely distributed spiking neurons, such as in [4], is also an example of a state transition system of this kind, because each neuron makes discrete state transitions, while space and time are continuous.

The rest of this paper is organized as follows. The next section presents the basic definitions, including that of transition rules, with some examples. In Section 3, we introduce two conditions, i.e., realizability and earliness, in order to define correct executions. Section 4 then gives a sufficient condition for satisfying the two conditions. A condition that ensures the uniqueness of an execution is also given.

2 Basic Definitions

Let T denote the time line. In this paper, we assume that T is simply the set of all non-negative real numbers. Let X be a metric space. Again, we assume that X is the n-dimensional Euclidean space $\mathbf{R}^n$.

Let Q be a finite set of states. At each point on the metric space X, we allocate a finite or infinite sequence of state transitions. Let σ denote a function from X to $(Q \times T)^+ \cup (Q \times T)^\omega$, i.e., for each point $x \in X$, $\sigma(x)$ is a nonempty finite or infinite sequence of pairs from $Q \times T$. Moreover, σ should satisfy the following conditions for any $x \in X$.

- $\sigma(x) = (q_0, t_0), (q_1, t_1), \cdots$
- $t_0 = 0$
- $t_0 \leq t_1 \leq \cdots$
- If $\sigma(x)$ is infinite, then $\lim_{i\to\infty} t_i = \infty$.

If σ satisfies these conditions, we call σ a *trajectory*. The set of all trajectories is denoted by $Tr(T, X, Q)$.

Note that adjacent state transitions may occur instantaneously, i.e., t_i may equal t_{i+1}. Since the last condition forbids the Zeno behavior, such instantaneous transitions are not repeated infinitely.

If $\sigma(x)$ is finite and $\sigma(x) = (q_0, t_0), (q_1, t_1), \cdots, (q_n, t_n)$, we define $t_{n+1} = \infty$ for brevity of later definitions and discussions.

A *transition rule* has the form

$$q \to q' \textbf{ if } A,$$

where $q, q' \in Q$ and A is a formula as defined below. The rule $q \to q'$ **if** A means that the state of a point is allowed to change from q to q' if the condition expressed by formula A is satisfied at that point. (We later impose a condition that a point should change its state if it can do so.)

Formulas are defined as follows:

$$A \ ::= q \ \mid \ A \wedge A \ \mid \ A \vee A \ \mid \ \Diamond_v A \ \mid \ \neg A$$

A state $q \in Q$ is an atomic formula. A formula of the form $\Diamond_v A$ means that in the neighborhood specified by v, there exists a point that satisfies A, where v is a non-negative real number denoting the propagation speed of state transitions. More precisely, in order for $\Diamond_v A$ to hold at a pair (x, t), there should exist a pair (x', t') such that $d(x', x) \leq v(t - t')$ and A holds at (x', t'). Moreover, such t' should be arbitrarily close to (x, t). Formally, for any $\epsilon > 0$, there should exist (x', t') such that $t - \epsilon < t' < t$ and $d(x', x) \leq v(t - t')$. This interpretation of the formula $\Diamond_v A$ is the starting point of the following formalism.

The interpretation $[\![A]\!]$ of formula A is defined as a function from $X \times T$ to $\{\top, \bot\}$. Note that it depends on the trajectory σ under consideration.

- $[\![q]\!](x, t) = \top$ if and only if there exists i such that $q_i = q$, and either $t = t_i$ or $t_i < t < t_{i+1}$, where $\sigma(x) = (q_0, t_0), (q_1, t_1), \cdots$. Note that if $\sigma(x)$ is finite and (q_i, t_i) is the last transition of $\sigma(x)$, the condition $t_i < t < t_{i+1}$ is equivalent to $t_i < t$ since we have defined $t_{i+1} = \infty$. Note also that the disjunction "$t = t_i$ or $t_i < t < t_{i+1}$" is not equivalent to $t_i \leq t < t_{i+1}$ since the latter requires $t_i < t_{i+1}$.
- $[\![A_1 \wedge A_2]\!](x, t) = \top$ if and only if $[\![A_1]\!](x, t) = \top$ and $[\![A_2]\!](x, t) = \top$.
- $[\![A_1 \vee A_2]\!](x, t) = \top$ if and only if $[\![A_1]\!](x, t) = \top$ or $[\![A_2]\!](x, t) = \top$.
- $[\![\Diamond_v A]\!](x, t) = \top$ if and only if for any $\epsilon > 0$, there exist $t' \in T$ and $x' \in X$ such that $t - \epsilon < t' < t$, $d(x', x) \leq v(t - t')$, and $[\![A]\!](x', t') = \top$, where $d(\cdot, \cdot)$ denotes the metric of the metric space X. Note that this condition is not equivalent to the following: for any $t' < t$, there exists $x' \in X$ such that $d(x', x) \leq v(t - t')$ and $[\![A]\!](x', t') = \top$.
- $[\![\neg A]\!](x, t) = \top$ if and only if $[\![A]\!](x, t) = \bot$.

When $[\![A]\!](x, t) = \top$, we write $(x, t) \models A$.

The following condition **(soundness)** is a natural requirement for a trajectory σ.

> **(soundness)** Let $x \in X$ and assume $\sigma(x) = (q_0, t_0), (q_1, t_1), \cdots$. For each $i > 0$, there exists a transition rule of the form $q_{i-1} \to q_i$ **if** A such that $(x, t_i) \models A$ holds.

As an initial condition, we give the first state of each point, i.e., a function f_0 from X to Q. A trajectory σ satisfies f_0, if for any $x \in X$, $q_0 = f_0(x)$ holds. Henceforth, we assume that a trajectory satisfies a given initial condition.

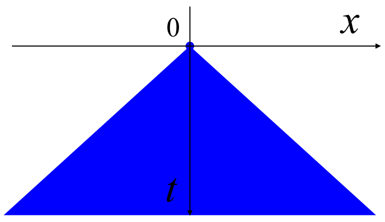


Fig. 1. A trajectory for Example 1

Example 1: Consider the following transition rule.

$$\text{white} \rightarrow \text{blue } \textbf{if } \Diamond_1 \text{ blue}$$

The suffix 1 in $\Diamond_1$blue denotes the speed of propagation. By this rule, blue points propagate with speed 1.

Let X be the one-dimensional Euclidean space, i.e., $X = \mathbf{R}$. With the initial condition f_0, where

$$f_0(x) = \text{blue if } x = 0, \text{ and}$$
$$f_0(x) = \text{white otherwise,}$$

we obtain the following trajectory σ (Figure 1).

$$\sigma(0) = (\text{blue}, 0)$$
$$\sigma(x) = (\text{white}, 0), (\text{blue}, |x|) \quad (x \neq 0)$$

It is obvious that for each state transition $(\text{blue}, |x|)$ in $\sigma(x)$, the judgment $(x, |x|) \models \Diamond_1$blue holds.

Example 2: Consider the following set of transition rules.

$$\text{white} \rightarrow \text{red } \textbf{if } \Diamond_1 \text{ red} \wedge \neg\Diamond_1 \text{ green}$$
$$\text{white} \rightarrow \text{green } \textbf{if } \Diamond_1 \text{ green} \wedge \neg\Diamond_1 \text{ red}$$
$$\text{white} \rightarrow \text{yellow } \textbf{if } \Diamond_1 \text{ red} \wedge \Diamond_1 \text{ green}$$
$$\text{red} \rightarrow \text{yellow } \textbf{if } \Diamond_1 \text{ yellow}$$
$$\text{green} \rightarrow \text{yellow } \textbf{if } \Diamond_1 \text{ yellow}$$

In this example, a white point becomes red if a neighboring red point exists and no green point exists in its neighborhood. Similarly, green points also propagate. If a white point is adjacent to both red and green points, it becomes yellow. Yellow points then propagate into red and green regions.

The upper panel of Figure 2 shows one red point at 0 and one green point at 1 propagating in one-dimensional space. The lower panel of Figure 2 also shows one red point and one green point, propagating in two-dimensional space (see Section 5).

Example 3: Consider the following set of transition rules

$$\text{white} \rightarrow \text{red } \textbf{if } \Diamond_1 \text{ red} \wedge \neg\Diamond_1 \text{ green}$$
$$\text{white} \rightarrow \text{green } \textbf{if } \Diamond_1 \text{ green} \wedge \neg\Diamond_1 \text{ red}$$

white $\longrightarrow$ yellow **if** $\Diamond_1$ red $\wedge$ $\Diamond_1$ green
red $\longrightarrow$ yellow **if** $\Diamond_3$ yellow
green $\longrightarrow$ yellow **if** $\Diamond_3$ yellow

In this example, yellow points propagate three times faster than red and green points, as specified by $\Diamond_3$ yellow. As a result, a thin membrane made of

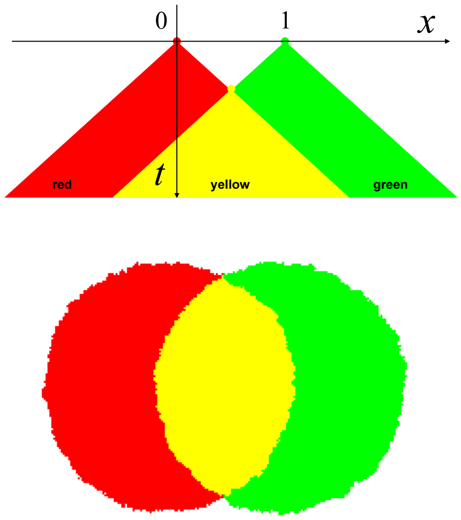


Fig. 2. Example 2

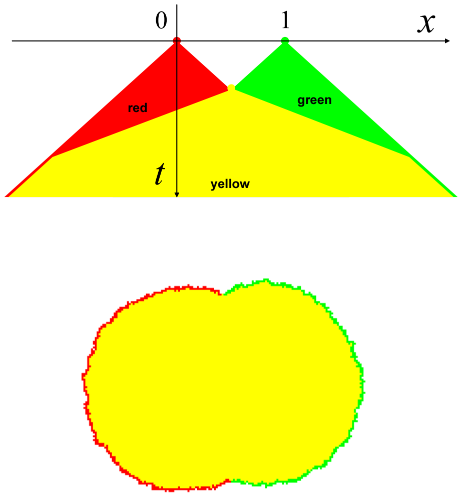


Fig. 3. Example 3

red and green points, enclosing yellow points, expands as in the two-dimensional case in the lower panel of Figure 3. The upper panel of Figure 3 shows the corresponding one-dimensional case, the trajectory σ of which is as follows.

$$\begin{array}{l}
\sigma(x) = (\text{white}, 0), (\text{red}, -x), (\text{yellow}, -x) \quad (x \leq -1) \\
\sigma(x) = (\text{white}, 0), (\text{red}, |x|), (\text{yellow}, 2/3 - x/3) \quad (-1 < x < 1/2, x \neq 0) \\
\sigma(0) = (\text{red}, 0), (\text{yellow}, 2/3) \\
\sigma(1/2) = (\text{white}, 0), (\text{yellow}, 1/2) \\
\sigma(x) = (\text{white}, 0), (\text{green}, |1-x|), (\text{yellow}, 1/3+x/3) \quad (1/2 < x < 2, x \neq 1) \\
\sigma(1) = (\text{green}, 0), (\text{yellow}, 2/3) \\
\sigma(x) = (\text{white}, 0), (\text{green}, x-1), (\text{yellow}, x-1) \quad (2 \leq x)
\end{array}$$

3 Realizability and Earliness

In Example 1 in the previous section, we had the following trajectory (Figure 1).

$$\begin{array}{l}
\sigma(0) = (\text{blue}, 0) \\
\sigma(x) = (\text{white}, 0), (\text{blue}, |x|) \quad (x \neq 0)
\end{array}$$

This simple example shows that the requirement **(soundness)** given in the previous section is insufficient to characterize a trajectory that satisfies the given transition rules.

Consider the following trajectory (Figure 4) for the transition rule in Example 1.

$$\begin{array}{l}
\sigma(0) = (\text{blue}, 0) \\
\sigma(x) = (\text{white}, 0), (\text{blue}, \min(|x|, |x-1|)) \quad (x \neq 0 \wedge x \neq 1) \\
\sigma(1) = (\text{white}, 0)
\end{array}$$

In this trajectory, point 1 is a source of blue points, although it is not blue initially. For example, consider the pair $(1+t, t)$ for $t > 0$. We have $(1+t, t) \models \Diamond_1\text{blue}$, because for any $\epsilon > 0$, there exists t' such that $t - \epsilon < t' < t$ and $(1+t', t') \models \text{blue}$.

For the same example, consider the following trajectory for $k > 1$.

$$\begin{array}{l}
\sigma(0) = (\text{blue}, 0) \\
\sigma(x) = (\text{white}, 0), (\text{blue}, k|x|) \quad (x \neq 0)
\end{array}$$

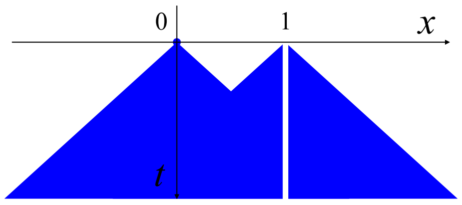


Fig. 4. Another trajectory for Example 1

In this trajectory, those points that could make a state transition did not do so. As an extreme case, even the following trajectory satisfies the initial condition and transition rule.

$$\begin{aligned}&\sigma(0) = (\text{blue}, 0)\\&\sigma(x) = (\text{white}, 0)\end{aligned}$$

There are two kinds of problem. The first one is the inverse of the Zeno problem. State transitions may occur without any source in the initial condition. As we noted above, we have $(1+t, t) \models \Diamond_1\text{blue}$, because $(1+t', t') \models \text{blue}$ holds for some $t' < t$. However, in order for this to hold, the state transition from white to blue should occur at $(1+t', t')$ and $(1+t', t') \models \Diamond_1\text{blue}$ should hold. Therefore, there should exist $t'' < t'$ such that $(1+t'', t'') \models \text{blue}$. In this way, we obtain an infinite sequence of state transitions without any source in the initial condition.

The second one is the opposite. Even if a point can make a state transition, there is no requirement to force it to do so.

To solve the first problem, we introduce the notion of realizability. For the second, we enforce the earliness condition as defined below.

A trajectory $\sigma \in Tr(T, X, Q)$ is called *realizable* if σ satisfies the following condition.

(realizability) For each $x \in X$ such that $\sigma(x) = (q_0, t_0), (q_1, t_1), \cdots$ and for each $i > 0$, there exists a transition rule of the form $q_{i-1} \to q_i$ **if** A such that for any $\epsilon > 0$, the judgment $(x, t_i) \vdash_\epsilon A$ is derivable using the inference rules introduced in Figure 5.

$$\frac{\begin{array}{c}\sigma(x) = (q_0, t_0), (q_1, t_1), \cdots \quad i \geq 0 \quad t = t_i \vee t_i < t < t_{i+1}\\ q_0 \to q_1 \ \mathbf{if}\ A_0 \quad (x, t_1) \vdash_\epsilon A_0\\ \cdots\\ q_{i-1} \to q_i \ \mathbf{if}\ A_{i-1} \quad (x, t_i) \vdash_\epsilon A_{i-1}\end{array}}{(x, t) \vdash_\epsilon q_i}$$

$$\frac{(x, t) \vdash_\epsilon A_1 \quad (x, t) \vdash_\epsilon A_2}{(x, t) \vdash_\epsilon A_1 \wedge A_2}$$

$$\frac{(x, t) \vdash_\epsilon A_1 \quad (x, t) \models \neg A_2}{(x, t) \vdash_\epsilon A_1 \wedge \neg A_2} \qquad \frac{(x, t) \vdash_\epsilon A_i}{(x, t) \vdash_\epsilon A_1 \vee A_2}$$

$$\frac{t - \epsilon < t' < t \quad d(x', x) \leq v(t - t') \quad (x', t') \vdash_\epsilon A \quad (x', t') \models A}{(x, t) \vdash_\epsilon \Diamond_v A}$$

Fig. 5. Inference rules for realizability

This condition roughly means that each transition can be traced back to the initial condition in a finite number of steps, uniformly with respect to $\epsilon > 0$. This finiteness is guaranteed by the finiteness of the derivation of $(x, t_i) \vdash_\epsilon A$. Note that the condition implies **(soundness)**.

The inference rules in Figure 5 assume that the formula A in a transition rule $q \to q'$ **if** A satisfies the following restriction:

$$A \ ::= q \ | \ A \wedge A \ | \ A \wedge \neg A_0 \ | \ A \vee A \ | \ \Diamond_v A,$$

where A_0 may be an arbitrary formula. This notion of realizability restricts occurrences of negative information (i.e., $\neg A_0$), and only requires the finiteness on the positive side.

In the first rule in Figure 5, the parameter i may be equal to 0. This is the base case, i.e., there is no premise of the form $(x, t_i) \vdash A_{i-1}$ if $i = 0$.

We then define an order between trajectories. This order represents the earliness condition, i.e., for $\sigma, \sigma' \in \mathit{Tr}(T, X, Q)$, the order $\sigma \leq \sigma'$ means that σ makes a state transition earlier than σ'. The order $\sigma \leq \sigma'$ holds if the following condition is met.

(earliness) For any $x \in X$, either $\sigma(x) = \sigma'(x)$ or there exists n such that
- for any $i < n$, $t_i = t'_i$ and $q_i = q'_i$, and
- $t_n < t'_n$, or (q'_{n-1}, t'_{n-1}) is the last transition of $\sigma'(x)$ while $\sigma(x)$ has the transition (q_n, t_n),

where $\sigma(x) = (q_0, t_0), (q_1, t_1), \cdots$ and s$\sigma'(x) = (q'_0, t'_0), (q'_1, t'_1), \cdots$.

A trajectory $\sigma \in \mathit{Tr}(T, X, Q)$ is said to be a *solution,* if σ is minimal with respect to $\leq$ among realizable trajectories.

4 A Sufficient Condition for Solutions

In this section, we give a sufficient condition for a trajectory to be a solution.

First, we explain the notions needed to define the condition, using Example 1. In order to show that the trajectory in Figure 1 is a solution, it suffices to check that the pair (x, t) does not allow any state transition if $|x| > t$, because its realizability can be easily shown. We call such (x, t) inactive. In the following, we introduce the notion *strong inactiveness*, which is a sufficient condition for (x, t) to be inactive. This notion is defined as the non-existence of a certain path from (x, t) to some pair (x_n, t_n).

In this example, we consider a path $(x_0, t_0), (x_1, t_1), \cdots, (x_n, t_n)$ satisfying the following conditions:

- $(x_0, t_0) = (x, t)$,
- $t_i > 0$ for $0 \leq i < n$,
- for $0 \leq i < n$, $t_i \geq t_{i+1}$,
- for $0 \leq i < n$, if $x_{i+1} \neq x_i$ then $0 < t_i - t_{i+1} < \epsilon$ and $d(x_{i+1}, x_i) \leq t_i - t_{i+1}$,

– for $0 \leq i < n$, $(x_i, 0) \models \text{white}$,
– $(x_n, t_n) \models \text{blue}$, and
– if $t_n > 0$, then there is no transition at any (x_n, t') such that $0 < t' \leq t_n$.

These conditions depend on $\epsilon > 0$. We call a path satisfying these conditions a $(\{\text{white}\}, 1, \text{blue}, \epsilon)$-path.

For (x, t) to make a transition, there should exist a $(\{\text{white}\}, 1, \text{blue}, \epsilon)$-path for each $\epsilon > 0$. Therefore, if there exists no such path for some $\epsilon > 0$, then (x, t) should be strongly inactive. In this example, the following stronger statement holds: there exists no such path for any $\epsilon > 0$ if $|x| > t$. Consequently, (x, t) is strongly inactive if $|x| > t$. This implies that the trajectory in Figure 1 is indeed a solution.

In general, we make the following definitions. First, we fix an infinite sequence $\{\tau_k\}_{k \geq 0}$ of time points, where $\tau_k \in T$, such that

– $\tau_0 = 0$
– $\tau_k < \tau_{k+1}$ for any $k \geq 0$, and
– $\lim_{k \to \infty} \tau_i = \infty$.

This sequence can be chosen in accordance with the trajectory under consideration. For simple cases, any sequence applies.

For formula A, the predicate **maybe**(A) denotes a necessary condition for A to hold. It is derived as follows.

$$\frac{(x, t) \vdash \mathbf{path}(\emptyset, 0, A)}{(x, t) \vdash \mathbf{maybe}(A)}$$

The predicate **path**(I, v, A) is derived using the inference rules in Figure 6. In **path**(I, v, A), I is a set of states and v is a non-negative real number, denoting the maximum speed of propagation. When A is a state q, the predicate **path**(I, v, q) holds at (x, t), if there exists an (I, v, q, ϵ)-*path* from (x, t) for any $\epsilon > 0$.

An (I, v, q, ϵ)-*path* from (x, t) is a sequence $\{(x_i, t_i)\}_{0 \leq i \leq n}$ of pairs such that

– $(x_0, t_0) = (x, t)$,
– if $t = 0$, then $n = 0$ and $t_0 = t_n = t = 0$,
– if $\tau_k < t \leq \tau_{k+1}$, then $t_i > \tau_k$ for $0 \leq i < n$, and $t_n > \tau_k - \epsilon$,
– for $0 \leq i < n$, $t_i \geq t_{i+1}$,
– for $0 \leq i < n$, if $x_{i+1} \neq x_i$ then $0 < t_i - t_{i+1} < \epsilon$ and $d(x_{i+1}, x_i) \leq v(t_i - t_{i+1})$,
– for $0 \leq i < n$, $(x_i, \tau_k) \models q'$ for some $q' \in I$,
– $(x_n, t_n) \models q$, and
– if $t_n > \tau_k$, then there is no transition at any (x_n, t') such that $\tau_k < t' \leq t_n$.

The above definition appears quite complicated, but in concrete examples, it usually reduces to a simple form. For Example 1, let $\tau_0 = 0$ and consider (x, t) such that $\tau_0 < t \leq \tau_1$ and $|x| > t$. The conditions on a $(\{\text{white}\}, 1, \text{blue}, \epsilon)$-path $\{(x_i, t_i)\}_{0 \leq i \leq n}$ from (x, t) are reduced to those mentioned at the beginning of this section.

$$\frac{\text{for any } \epsilon > 0, \text{ there exists an } (I, v, q, \epsilon)\text{-path from } (x, t)}{(x, t) \vdash \mathbf{path}(I, v, q)}(*)$$

$$\frac{\begin{array}{cc} & i \geq 0 \\ q_0 \to q_1 \ \mathbf{if}\ A_0 & (x, t) \vdash \mathbf{path}(I \cup \{q_0\}, v, A_0) \\ & \cdots \\ q_i \to q_{i+1} \ \mathbf{if}\ A_i & (x, t) \vdash \mathbf{path}(I \cup \{q_0\}, v, A_i) \end{array}}{(x, t) \vdash \mathbf{path}(I, v, q_{i+1})}$$

$$\frac{(x, t) \vdash \mathbf{path}(I, v, A_1) \quad (x, t) \vdash \mathbf{path}(I, v, A_2)}{(x, t) \vdash \mathbf{path}(I, v, A_1 \wedge A_2)}$$

$$\frac{(x, t) \vdash \mathbf{path}(I, v, A_1)}{(x, t) \vdash \mathbf{path}(I, v, A_1 \wedge \neg A_2)} \qquad \frac{(x, t) \vdash \mathbf{path}(I, v, A_i)}{(x, t) \vdash \mathbf{path}(I, v, A_1 \vee A_2)}$$

$$\frac{(x, t) \vdash \mathbf{path}(I, \max(v, v'), A)}{(x, t) \vdash \mathbf{path}(I, v, \Diamond_{v'} A)}$$

Fig. 6. Inference rules for paths

In the second rule of **path**, we enumerate the possibility of successive transitions $q_0, q_1, \cdots, q_i, q_{i+1}$ at some point in X. We can assume that there is no duplicate rule among $q_0 \to q_1$ **if** A_0, $\cdots$, $q_i \to q_{i+1}$ **if** A_i, without affecting the derivability.

Using the predicate **maybe**, we define the strong inactiveness as follows. For a trajectory $\sigma \in Tr(T, X, Q)$ and a point $x \in X$, let $\sigma(x) = (q_0, t_0), (q_1, t_1), \cdots$. In the case $t_i < t < t_{i+1}$, the pair (x, t) is said to be *strongly inactive*, if $(x, t) \vdash$ **maybe**(A) is not derivable for any transition rule of the form $q_i \to q$ **if** A. In the case $t = t_i$, the pair (x, t_i) is said to be strongly inactive if the following conditions are satisfied:

– The state transition (q_i, t_i) is the last one of $\sigma(x)$ or $t_i < t_{i+1}$.
– For any transition rule of the form $q_i \to q$ **if** A, $(x, t_i) \vdash$ **maybe**(A) is not derivable.

If each pair $(x, t) \in X \times T$ is strongly inactive, then σ is said to be *strongly inactive.*

For Example 1, if $|x| > t > 0$, then we can show that the pair (x, t) is strongly inactive. We check if $(x, t) \vdash$ **maybe**$(\Diamond_1\text{blue})$ is not derivable. This condition first reduces to **path**$(\emptyset, 1, \text{blue})$. Since there is only one rule, i.e., white $\to$ blue **if** $\Diamond_1$blue, **path**$(\emptyset, 1, \text{blue})$ can be derived only by **path**$(\{\text{white}\}, 1, \text{blue})$. We have already shown that there exists no $(\{\text{white}\}, 1, \text{blue}, \epsilon)$-path from (x, t).

Theorem 1. If $\sigma \in Tr(T, X, Q)$ is realizable and strongly inactive, then σ is a solution.

Proof sketch. If there exists a realizable trajectory σ' such that $\sigma' < \sigma$, there should exist a pair (x, t) at which σ' makes a state transition, but σ does not do so. Let the transition rule used by σ' at (x, t) be $q \rightarrow q'$ **if** A. We show by induction on k, there exists no such (x, t) that satisfies $t \leq \tau_k$. The case $k = 0$ is easy because $t = 0$. In the following, we assume $\tau_k < t \leq \tau_{k+1}$.

Let **maybe**$'$ and **path**$'$ be the predicates defined similarly to **maybe**$'$ and **path**$'$, except that the rule marked $(*)$ is replaced with the following one.

$$\frac{\text{there exists an } (I, v, q, m\epsilon)\text{-path from } (x, t)}{\mathbf{path}'(I, v, q)} (**)$$

In this rule, ϵ is fixed to some predetermined value, and m is the maximum depth of transition rules. The depth of a transition rule $q \rightarrow q'$ **if** A is defined as the maximum number of nestings of $\Diamond_v$ in A.

For each $\epsilon > 0$, from the derivation of $(x, t) \vdash_\epsilon A$ for σ', we can extract an $(I, v, q, m\epsilon)$-path from (x, t) to (x', t'), where $(x', t') \vdash_\epsilon q$. We can then construct a derivation of $(x, t) \vdash \mathbf{maybe}'(A)$ for σ together with an $(I, v, q, m\epsilon)$-path for the premise of $\mathbf{path}'(I, v, q)$ at each leaf of the derivation.

The derivation of $\mathbf{maybe}'(A)$ can be shrunken if $\mathbf{path}'(I, v, q)$ appears at a leaf or at an intermediate node of the derivation. Note that there are a finite number of such shrunken derivations. Therefore, there exists some derivation that is constructed for infinitely many values of ϵ that approach zero. For such a derivation, the rule $(**)$ can be replaced with the rule $(*)$, so the derivation of $(x, t) \vdash \mathbf{maybe}(A)$ can be obtained. □

Unfortunately, the theorem is not enough in the case of Example 3. The method we have proposed above shows the inactiveness of a trajectory for each interval between τ_i and τ_{i+1}. However, in cases like Example 3, it is necessary to show the inactiveness of some regions within an interval. As for Example 3, the white triangular region surrounded by the red and green ones should be first shown to be strongly inactive. Then it becomes possible to show the red and green regions strongly inactive. We should therefore modify the definition of an (I, v, q, ϵ)-path. We require that an (I, v, q, ϵ)-path $(x_0, t_0), (x_1, t_1), \cdots$ stop at (x_i, t_i) if (x_i, t_i) has already been shown strongly inactive.

Demonstrating the uniqueness of a solution is even more complicated [5]. A simple sufficient condition for uniqueness is obtained by modifying the strong inactiveness. As before, for a trajectory $\sigma \in Tr(T, X, Q)$ and a point $x \in X$, let $\sigma(x) = (t_0, q_0), (t_1, q_1), \cdots$. The state transition (q_i, t_i) $(i > 0)$ is said to be *strongly deterministic*, if the following condition holds.

– For a transition rule of the form $q_{i-1} \rightarrow q$ **if** A, if $(x, t_i) \vdash \mathbf{maybe}(A)$ is derivable, then $q = q_i$.

If all the transitions are strongly deterministic, σ is said to be *strongly deterministic.*

Theorem 2. If $\sigma \in \mathit{Tr}(T, X, Q)$ is realizable, strongly inactive, and strongly deterministic, then σ is a unique solution.

Unfortunately, since the predicate **path** throws away negative information, Examples 2 and 3 cannot be directly handled by this theorem. More concretely, the uniqueness of the transition at $(1/2, 1/2)$ should be shown using the definitions.

5 Amorphous Simulation

We have implemented a simple simulator of the framework. The two-dimensional Euclidean space is first divided into squares of the size $1/n \times 1/n$. Inside each square, a point is chosen randomly. At each time step, the condition of each transition rule is evaluated at each point and if the condition is true, the state of the point is changed according to the rule. A formula of the form $\Diamond_v A$ is evaluated straightforwardly. The formula A is evaluated for each neighbor within a distance v/m, and if A is true at some neighbor, then the formula A is evaluated as true. Note that this simple method works only if each point is chosen randomly on its square. As a result, the points are scattered over the space in an amorphous fashion. Figures 2 and 3 are obtained for $n = 2$ and $m = 1$. An actual implementation of amorphous simulation can be found at the following URL: `http://hagi.is.s.u-tokyo.ac.jp/members/hagiya/amorphous/`.

6 Related Work and Concluding Remarks

Recently, based on the author's suggestion, Takeuti formalized a similar kind of state transition system using well-established notions from general topology [5]. He enforced the minimum speed of propagation and proved the uniqueness of a solution. Our work is based on more naive notions, such as realizability and earliness, but uniqueness as well as the existence of a solution should be shown using specific properties of each transition system. Another difference is that his model cannot cope with instantaneous transitions, so Example 3 is not realizable. Furthermore, the simulation of cellular automata, which Takeuti shows in his model, can also be implemented in our model.

Although the speed of propagation is restricted to a constant speed here, the restriction may be relaxed so that the speed may depend on time and space. In this case, more complex spatiotemporal patterns can be generated.

There are, of course, cases for which no solution exists. For example, the following variant of Example 3 does not have a solution, because after red and green points encounter, their states cannot be determined.

white $\rightarrow$ red **if** $\Diamond_1$ red $\wedge \neg\Diamond_1$ green
white $\rightarrow$ green **if** $\Diamond_1$ green $\wedge \neg\Diamond_1$ red
white $\rightarrow$ yellow **if** $\Diamond_1$ red $\wedge$ $\Diamond_1$ green
red $\rightarrow$ green **if** $\Diamond_3$ green
green $\rightarrow$ red **if** $\Diamond_3$ red

How to cope with such cases is left for future work.

As mentioned in the Introduction, a densely distributed neural network is an example of the transition system discussed here. To realize such a neural network, it is necessary to introduce more operators for describing conditions on state transitions. In this paper, we introduced only the $\Diamond_v$ operator, which simply checks the existence of a certain point in the neighborhood. For a neural network, we need to introduce operators, such as one that integrates the time after the last transition (firing) in the neighborhood.

Acknowledgments

The author deeply thanks the anonymous reviewers for pointing out some crucial errors in the submitted version of the paper. He also thanks Yoshinori Tanabe for his valuable comments for improving the paper. This work is partially supported by Grand-in-Aid for Scientific Research on Priority Area No.14085202, Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. R. Alur, C. Courcoubetis, N. Halbwacks, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science,* Vol.138, pp.3–34, 1995.
2. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss: Amorphous computing, *Communications of the ACM,* Vol.43, No.5, pp.74–82, 2000.
3. L. Berec: Techniques of spatially explicit individual-based models: construction simulation, and mean-field analysis, *Ecological Modelling,* Vol.150, pp.55–81, 2002.
4. R. Osam, J. Rubin, R. Curtu, and B. Ermentrout: Traveling waves in a one-dimensional integrate-and-fire neural network with finite support connectivity, *Neurocomputing,* Vol.52–54, pp.869–875, 2003.
5. I. Takeuti: Transition systems over continuous time-space, *Electronic Notes in Theoretical Computer Science,* Vol.120, pp.173–186, 2005.
6. T. Tokihiro, D. Takahashi, J. Matsukidaira, and J. Satsuma: From soliton equations to integrable cellular automata through a limiting procedure, *Physical Review Letters,* Vol.76, No.18, pp.3247–3250, 1996.

On Reversible Cellular Automata with Finite Cell Array

Shuichi Inokuchi[1], Kazumasa Honda[2], Hyen Yeal Lee[3], Tatsuro Sato[4], Yoshihiro Mizoguchi[1], and Yasuo Kawahara[2]

[1] Faculty of Mathematics, Kyushu University 33, Fukuoka 812-8581, Japan
[2] Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
[3] School of Computer Engineering, Pusan National University, Pusan 609-735, Korea
[4] Oita National College of Technology, Oita, 870-0152, Japan

Abstract. Discrete quantum cellular automata are cellular automata with reversible transition. This paper deals with 1d cellular automata with finite cell array and triplet local transition rules. We present the necessary condition of local transition rules for cellular automata to be reversible, and prove the reversibility of some cellular automata.

1 Introduction

Since Feynman proposed the notion of "quantum computation", a lot of models of quantum computation have been investigated. Watrous [8] introduces the notion of quantum cellular automata(QCA, for short) of a kind of quantum computer and showed that any quantum Turing machines can be simulated by a partitioned QCA(PQCA) with constant slowdown. Moreover he presented necessary and sufficient conditions for the well-formedness of 1d PQCA. Watrous' QCA have infinite cell arrays. Inokuchi and Mizoguchi [4] introduced a notion of cyclic QCA with finite cell array, which generalises PQCA, and formulated sufficient condition for local transition functions to form QCA. Quantum computations are performed by means of applying unitary transformations for quantum states. So classical cellular automata(CA) with reversible transition functions are considered as a special type of QCA.

On the other hand CA have also been studied as models of universal computation and complex systems [2,3,6,10]. Reversible CA and the reversibility of CA were discussed by many researchers. Wolfram [10] investigated the reversibility of several models of CA and showed that only six CA, whose transition functions are identity function, right-shift function, left-shift function and these complement functions, of the 256 elementary CA with infinite cell array are reversible. Dow [1] investigated the injectivity(reversibility) of additive CA with finite cyclic cell array and infinite cell array, and showed the relation between injectivities of these additive CA. Morita and Harao [7] showed that for any reversible Turing machine there is a reversible CA that simulates it. Kobuchi and Nishio [5] investigated 1d CA with finite cell array. And they showed that the set of Garden-of-Eden configurations is regular set and it is decidable whether 1d CA with finite cell array is reversible or not.

This paper treats with 1d CA, denoted by $CA-R(n)$, with finite cell array, triplet local rule of rule number R and two states, and investigate the reversibility of $CA-R(n)$. We can easily observe dynamical behaviors of $CA-R(n)$ by computer simulations, and consequently get the following table which shows whether 1d cellular automata $CA-R(n)$ with finite and infinite cell array are reversible or not, according to five types of boundary conditions a-b, a-$*$, $*$-b, $*$-$*$ and $*$, which will be defined in the next section.

Rule numbers	a-b	a-$*$	$*$-b	$*$-$*$	$*$	∞
51, 204	○	○	○	○	○	○
15, 85, 170, 240					○	○
90, 165	△[1]	○	○			
60,195	○	○				
102,153	○		○			
150,105	△[2]	△[3]	△[4]	△[4]	△[5]	
166,180,154,210,89,75,101,45					△[5]	

Although for 1d CA with infinite cell array there exist only six trivial reversible CA, we will prove that there exist several non trivial reversible 1d CA with finite cell array.

2 Cellular Automata $CA-R$

Cellular automata treated in the paper have linearly ordered and finite number cells bearing with states 0 or 1. The next state of any cell depends upon the states of left cell, the cell itself and the right cell. In this section we will formally define cellular automata $CA-R(n)$ with rule number R of triplet local rule and n cell array.

Let Q be a state set $\{0, 1\}$ and n a positive integer. The complement of a state $a \in Q$ will be denoted by a^-, that is, $a^- = 1 - a$. The state set Q forms an additive group by the addition $+$ (modulo 2) (the exclusive logical sum), that is, $0 + 0 = 1 + 1 = 0$ and $0 + 1 = 1 + 0 = 1$. Remark that $a^- = 1 + a$ for all states $a \in Q$. The n-th cartesian product of Q is denoted by Q^n, in other words, Q^n is the set of all n-tuples consisting of 0 and 1. For example, $Q^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$. An n-tuple $x = x_1x_2 \cdots x_n \in Q^n$ may be called a word of length n over Q, or a configuration in a context of cellular automata. It is obvious that the n-th cartesian product Q^n also forms an additive group by the component-wise addition, that is, $x_1x_2 \cdots x_n + y_1y_2 \cdots y_n = (x_1 + y_1)(x_2 + y_2) \cdots (x_n + y_n)$ for all n-tuples $x_1x_2 \cdots x_n, y_1y_2 \cdots y_n \in Q^n$.

A *triplet local (transition) rule* is a function $f : Q^3 \to Q$ and the *rule number* $|f|$ of f is defined by

$$|f| = \textstyle\sum_{a,b,c \in Q} 2^{4a+2b+c} f(abc).$$

[1] $n = 0 \pmod 2$, [2] $n \neq 2 \pmod 3$, [3] $n \neq 1 \pmod 3$, [4] $n \neq 0 \pmod 3$, [5] $n = 1 \pmod 2$

Note that the rule number $|f|$ is a natural number with $0 \le |f| \le 255$. A triplet local rule with rule number R will be denoted by f_R, namely $|f_R| = R$.

Let $f : Q^3 \to Q$ be a triplet local rule. The *symmetric rule* $f^\sharp : Q^3 \to Q$ of f is defined by

$$f^\sharp(abc) = f(cba)$$

for all triples $abc \in Q^3$. It is trivial that $f^{\sharp\sharp} = f$ and

$$|f^\sharp| = |f| + 56(r_3 - r_6) + 14(r_1 - r_4)$$

where $r_{4a+2b+c} = f(abc)$ for all triples $abc \in Q^3$. The *complementary rule* $f^- : Q^3 \to Q$ of f is defined by

$$f^-(abc) = [f(abc)]^-$$

for all triples $abc \in Q^3$. Note that $f^{--} = f$ and $f^{-\sharp} = f^{\sharp -}$ and

$$|f^-| = 255 - |f|.$$

The *reverse rule* $f^\circ : Q^3 \to Q$ of f is defined by

$$f^\circ(abc) = [f(a^- b^- c^-)]^-$$

for all triples $abc \in Q^3$. It is trivial that $f^{\circ\circ} = f$, $f^{\sharp\circ} = f^{\circ\sharp}$ and

$$|f^\circ| = 255 - (128r_0 + 64r_1 + 32r_2 + 16r_3 + 8r_4 + 4r_5 + 2r_6 + r_7).$$

A *dynamical system* is a pair (X, δ) of a set X and a transition function $\delta : X \to X$.

Let $f : Q^3 \to Q$ be a triplet local rule, n a positive integer and $a, b \in Q$. By setting different boundary conditions we can define five transition functions $f_{(n),a-b} : Q^n \to Q^n$, $f_{(n),*} : Q^n \to Q^n$, $f_{(n),*-*} : Q^n \to Q^n$, $f_{(n),a-*} : Q^n \to Q^n$ and $f_{(n),*-b} : Q^n \to Q^n$ as follows.

$$f_{(n),a-b}(x_1x_2\cdots x_{n-1}x_n) = f(a\,x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-2}x_{n-1}x_n)f(x_{n-1}x_nb),$$

$$f_{(n),*}(x_1x_2\cdots x_{n-1}x_n) = f(x_nx_1x_2)f(x_1x_2x_3)\cdots f(x_{n-2}x_{n-1}x_n)f(x_{n-1}x_nx_1),$$

$$f_{(n),*-*}(x_1x_2\cdots x_{n-1}x_n) = f(x_1x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-2}x_{n-1}x_n)f(x_{n-1}x_nx_n),$$

$$f_{(n),a-*}(x_1x_2\cdots x_{n-1}x_n) = f(a\,x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-2}x_{n-1}x_n)f(x_{n-1}x_nx_n),$$

$$f_{(n),*-b}(x_1x_2\cdots x_{n-1}x_n) = f(x_1x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-2}x_{n-1}x_n)f(x_{n-1}x_nb)$$

for all n-tuples $x_1x_2\cdots x_{n-1}x_n \in Q^n$ and all states $a, b \in Q$.

Set rule number $R = |f|$ $(0 \le R \le 255)$. Cellular automata $CA{-}R_{a-b}(n)$ with fixed boundary $a{-}b$, $CA{-}R_*(n)$ with cyclic boundary, $CA{-}R_{*-*}(n)$ with free boundary, $CA{-}R_{a-*}(n)$ with right free boundary $a{-}*$ and $CA{-}R_{*-b}(n)$ with left free boundary $*{-}b$ are dynamical systems $(Q^n, f_{(n),a-b})$, $(Q^n, f_{(n),*})$,

$(Q^n, f_{(n),*-*})$, $(Q^n, f_{(n),a-*})$ and $(Q^n, f_{(n),*-b})$, respectively. This is denoted by the followings for short;

$$CA{-}|f|_{\{a-b,a-*,*-b,*-*,*\}}(n) = (Q^n, f_{(n),\{a-b,a-*,*-b,*-*,*\}}).$$

Transition functions $\delta : Q^n \to Q^n$ are not always bijections, but when it is the case we can regard them as discrete quantum automata.

Definition 1. 1. A triplet local rule $f : Q^3 \to Q$ is *additive* if $f(abc + a'b'c') = f(abc) + f(a'b'c')$ for all triples $abc, a'b'c' \in Q^3$.
2. A transition function $\delta : Q^n \to Q^n$ is *additive* if $\delta(x + x') = \delta(x) + \delta(x')$ for all configurations $x, x' \in Q^n$.

We now recall the basic fact on the reversibility of dynamical systems over Q^n.

Lemma 1. *1. A transition function $\delta : Q^n \to Q^n$ is bijective iff it is injective iff it is surjective.*
2. If a triplet local rule $f : Q^3 \to Q$ is additive, then so is the transition function $f_{(n),\{0-0,0-*,*-0,*-*.*\}} : Q^n \to Q^n$ for all positive integers n.
3. An additive transition function $\delta : Q^n \to Q^n$ is bijective iff $\delta(x) = 0^n$ implies $x = 0^n$ for all configurations $x \in Q^n$.

Proof. (1) It is trivial since the set Q^n is finite. Also (2) and (3) are clear.

3 Basic Results

In this section we show some general properties of cellular automata $CA{-}R(n)$. Let (X, δ) and (Y, γ) be two dynamical systems. An *isomorphism* $t : (X, \delta) \to (Y, \gamma)$ is a bijection $t : X \to Y$ rendering the following square commutative:

$$\begin{tikzcd} X \arrow[r, "t"] \arrow[d, "\delta"'] & Y \arrow[d, "\gamma"] \\ X \arrow[r, "t"'] & Y. \end{tikzcd}$$

We call (X, δ) and (Y, γ) isomorphic, denoted by $(X, \delta) \cong (Y, \gamma)$, if there exists an isomorphism between (X, δ) and (Y, γ). It is trivial that isomorphic dynamical systems are essentially the same ones.

Lemma 2. *The followings holds;*

1. $CA{-}|f^\sharp|_{\{a-b,a-*,*-b,*-*,*\}}(n) \cong CA{-}|f|_{\{b-a,*-a,b-*,*-*,*\}}(n)$.
2. $CA{-}|f^\circ|_{\{a-b,a-*,*-b,*-*,*\}}(n) \cong CA{-}|f|_{\{a^- -b^-,a^- -*,*-b^-,*-*,*\}}(n)$.

Remark 1. The equation 1 of above lemma asserts the following five statements:

1. $CA{-}|f^\sharp|_{a-b}(n) \cong CA{-}|f|_{b-a}(n)$,
2. $CA{-}|f^\sharp|_{a-*}(n) \cong CA{-}|f|_{*-a}(n)$,

3. $CA{-}|f^\sharp|_{*-b}(n) \cong CA{-}|f|_{b-*}(n)$,
4. $CA{-}|f^\sharp|_{*-*}(n) \cong CA{-}|f|_{*-*}(n)$,
5. $CA{-}|f^\sharp|_{*}(n) \cong CA{-}|f|_{*}(n)$.

The proof of the above lemma is omitted, but we can easily prove it.

Corollary 1.

$$\begin{aligned} CA{-}|f|_{\{a-b,a-*,*-b,*-*,*\}}(n) &\cong CA{-}|f^\sharp|_{\{b-a,*-a,b-*,*-*,*\}}(n) \\ &\cong CA{-}|f^\circ|_{\{a^- - b^-, a^- - *, * - b^-, * - *, *\}}(n) \\ &\cong CA{-}|f^{\sharp\circ}|_{\{b^- - a^-, * - a^-, b^- - *, * - *, *\}}(n). \end{aligned}$$

Thus a quartet $[|f|, |f^\sharp|, |f^\circ|, |f^{\sharp\circ}|]$ of rule numbers $|f|$, $|f^\sharp|$, $|f^\circ|$ and $|f^{\sharp\circ}|$ is an equivalence class of rule numbers which represent isomorphic triplet local rules. For example, $[102, 60, 153, 195]$ and $[89, 75, 101, 45]$.

Lemma 3. *Let k and n be positive integers.*

1. *$CA{-}|f^-|_{\{a-b,a-*,*-b,*-*,*\}}(n)$ is reversible iff $CA{-}|f|_{\{a-b,a-*,*-b,*-*,*\}}(n)$ is.*
2. *If $CA{-}|f|_{*-*}((2k+1)n)$ is reversible, then so is $CA{-}|f|_{*-*}(n)$.*
3. *If $CA{-}|f|_{*}(kn)$ is reversible, then so is $CA{-}|f|_{*}(n)$.*

Proof. It is trivial.

For a local transition rule $f : Q^3 \to Q$ and an integer $k \geq 1$ we define a natural number

$$c_k(f) = \sum_{x_1 x_2 \cdots x_{k+2} \in Q^{k+2}} f(x_1x_2x_3) \times f(x_2x_3x_4) \times \cdots \times f(x_k x_{k+1} x_{k+2}).$$

The following lemma is helpful for the proof of irreversibility.

Lemma 4. *Let n be an integer ≥ 3. If the transition function $\delta : Q^n \to Q^n$ determined by a local transition rule $f : Q^3 \to Q$ is reversible, then*

$$c_k(f) = 4$$

for each integer k with $1 \leq k \leq n$.

Proof. Set $x = x_1x_2 \cdots x_n \in Q^n$ and $y = y_1y_2 \cdots y_n = \delta(x) \in Q^n$. Assume that $\delta : Q^n \to Q^n$ is reversible. Then we have

$$\sum_{y \in Q^n} y_2 \times y_3 \times \cdots \times y_{k+1} = 2^{n-k}$$

and also

$$\begin{aligned} &\sum_{y \in Q^n} y_2 \times y_3 \times \cdots \times y_{k+1} \\ &= \sum_{x \in Q^n} f(x_1x_2x_3) \times f(x_2x_3x_4) \times \cdots \times f(x_k x_{k+1} x_{k+2}) \\ &= 2^{n-k-2} \sum_{x_1x_2\cdots x_{k+2} \in Q^{k+2}} f(x_1x_2x_3) \times f(x_2x_3x_4) \times \cdots \times f(x_k x_{k+1} x_{k+2}). \end{aligned}$$

As a result of calculation c_k for 256 local transition rules, 70 and 34 local rules satisfy $c_1(f) = 4$ and $c_1(f) = c_2(f) = 4$, respectively. And there are 30 local rules such that $c_1(f) = c_2(f) = c_3(f)$=4. Their rule numbers are 30, 86, 106, 120, 135, 149, 169, 225 and those which will be treated in the following sections. Because of $f_{30}(abc) = 1 - f_{225}(abc)$ for any $abc \in Q^3$, the reversibilities of 8 rules $[30, 86, 135, 149]$ and $[106, 120, 169, 225]$ are equivalent. $CA - 30$ with finite cell array are not reversible under all the 5 boundary conditions adopted in the paper since we can get Garden-of-Eden configurations. Wolfram [9] discussed 1d CA with rule 30 as a random sequence generator. $CA - 30$ with finite cyclic cell array have limit cycles of longer period length than those of CA treated in the following sections.

4 $CA-\{204, 51\}(n)$

Triplet local rules $f_{\{204,51\}} : Q^3 \to Q$ are given, defined by $f_{204}(abc) = b$ and $f_{51}(abc) = b^-$ for all triples $abc \in Q^3$. So it holds that $f_{204}^- = f_{51}$, $[204, 204, 204, 204]$ and $[51, 51, 51, 51]$. Hence equalities

$$f_{204(n),\{a-b,a-*,*-b,*-*,*\}} = \mathrm{id}_{Q^n} \quad \text{and} \quad f_{51(n),\{a-b,a-*,*-b,*-*,*\}} = c_n$$

hold. Thus all configurations in $CA-204(n)$ are fixed points, i.e. $CA-204(n) \cong 2^n\langle 1\rangle$, and all configurations in $CA - 51(n)$ lie on limit cycles of period 2, i.e. $CA - 51(n) \cong 2^{n-1}\langle 2\rangle$ where $\langle n\rangle$ denotes a limit cycle of period n.

Corollary 2. *$CA-\{204, 51\}_{\{a-b,a-*,*-b,*-*,*\}}(n)$ are reversible for all positive integers n.*

5 $CA-\{240, 170, 15, 85\}(n)$

Since $f_{240}(abc) = a$, $f_{170}(abc) = c$, $f_{15}(abc) = a^-$ and $f_{85}(abc) = c^-$ for all triples $abc \in Q^3$, we have $f_{240}^- = f_{15}$, $[240, 170, 240, 170]$ and $[15, 85, 15, 85]$. Thus it is trivial that $f_{\{240,170,15,85\}(n),*}$ are bijective for all positive integers n.

Corollary 3. *$CA - \{240, 170, 15, 85\}_*(n)$ are reversible for all positive integers n.*

Lemma 5. *1. $f_{15(n),\{a-b,a-*,*-b,*-*,*\}} = f_{240(n),\{a^--b,a^--*,*-b,*-*,*\}} \circ c_n$,*
2. $f_{85(n),\{a-b,a-*,*-b,*-*,*\}} = f_{170(n),\{a-b^-,a-*,*-b^-,*-*,*\}} \circ c_n$.

Proof. We will show only $f_{15(n),a-b} = f_{240(n),a^--b} \circ c_n$.

$$\begin{aligned} f_{15(n),a-b}(x_1x_2\cdots x_n) &= f_{15}(ax_1x_2)f_{15}(x_1x_2x_3)\cdots f_{15}(x_{n-1}x_nb) \\ &= a^-x_1^-x_2^-\cdots x_{n-1}^- \\ &= f_{240}(a^-x_1^-x_2^-)f_{240}(x_1^-x_2^-x_3^-)\cdots f_{240}(x_{n-1}^-x_n^-b) \\ &= f_{240(n),a^--b}(x_1^-x_2^-\cdots x_n^-) \\ &= (f_{240(n),a^--b} \circ c_n)(x_1x_2\cdots x_n). \end{aligned}$$

Remark. $(f_{240(n),*})^n = (f_{170(n),*})^n = \mathrm{id}_{Q^n}$ and $(f_{15(n),*})^n = (f_{85(n),*})^n = (c_n)^n$.

Lemma 6. *$CA-\{240,170,15,85\}_{\{a-b,a-*,*-b,*-*\}}(n)$ are not reversible for all positive integers n.*

Proof. It simply follows from

$$f_{240(n),\{a-b,a-*,*-b,*-*\}}(a^n) = f_{240(n),\{a-b,a-*,*-b,*-*\}}(a^{n-1}a^-) = a^n.$$

6 $CA-\{90,165\}(n)$

It is obvious that $f_{90}(abc) = a + c$ and $f_{165}(abc) = (a+c)^-$ for all triples $abc \in Q^3$, and $[90, 90, 165, 165]$. Hence by Corollary 1 we have

$$CA-165_{\{a-b,a-*,*-b,*-*,*\}}(n) \cong CA-90_{\{a^--b^-,a^--*,*-b^-,*-*,*\}}(n)$$

and so we will inspect only $CA-90(n)$.

Lemma 7. *Let $x = x_1x_2\cdots x_n \in Q^n$. If $f_{90(n),\{a-b,a-*,*-b,*-*,*\}}(x) = 0^n$, then $x_i = x_{i+2}$ for all $i = 1,2,\cdots,n-2$.*

Proof. Set $f = f_{90}$. The condition $f_{90(n),\{a-b,a-*,*-b,*-*,*\}}(x) = 0^n$ means

$$f(*\,x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-2}x_{n-1}x_n)f(x_{n-1}x_n\,*) = 0^n.$$

Hence we have $x_i + x_{i+2} = f(x_ix_{i+1}x_{i+2}) = 0$ for all $i = 1,2,\cdots,n-2$.

Lemma 8. *1. $CA-90_{a-b}(n)$ is reversible iff so is $CA-90_{0-0}(n)$.*
2. $CA-90_{0-0}(n)$ is reversible iff $n = 0 \pmod 2$.

Proof. 1. It is immediate from a fact that $f_{90(n),a-b}(x) = f_{90(n),0-0}(x) + a0^{n-2}b$ for all $x \in Q^n$.

2. First we will show that $f_{90(n),0-0}$ is injective for $n = 0 \pmod 2$, since $f_{90(n),0-0}$ is additive. (Cf. Lemma 1.)
(i) Set $f = f_{90}$. It holds that $f_{(2),0-0}(x_1x_2) = f(0x_1x_2)f(x_1x_20) = x_2x_1$. Hence $CA-90_{0-0}(2)$ is reversible.
(ii) Assume that $CA-90_{0-0}(n)$ is reversible for $n \geq 2$, i.e. $f_{(n),0-0}$ is injective. We will see that $f_{(n+2),0-0}$ is also injective.
Assume $f_{(n+2),0-0}(x_1x_2\cdots x_nx_{n+1}x_{n+2}) = 0^{n+2}$. Then we have

$$\begin{aligned} f_{(n),0-0}(x_1x_2\cdots x_n) &= f(0x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-1}x_n0) \\ &= 0^{n-1}f(x_{n+1}x_{n+2}0) \\ &= 0^n, \end{aligned}$$

since $x_{n-1} = x_{n+1}$ and $x_n = x_{n+2}$ by Lemma 7. Hence by the induction hypothesis we have $x_1x_2\cdots x_n = 0^n$ and consequently $x_{n+1} = x_{n+2} = 0$. Therefore $CA-90_{0-0}(n+2)$ is reversible. Finally we see that if $n = 1 \pmod 2$ then $f_{90(n),0-0}$ is not injective. This follows at once from a fact that

$$f_{90(2k-1),0-0}((10)^{k-1}1) = 0^{2k-1}$$

holds for all positive integers k.

Corollary 4. *$CA{-}90_{a-b}(n)$ is reversible iff $n = 0 \pmod 2$.*

In the same discussion as Lemma 8 the following lemma can be shown.

Lemma 9. *1. $CA{-}90_{a-*}(n)$ is reversible iff so is $CA{-}90_{0-*}(n)$.*
2. $CA{-}90_{0-*}(n)$ is reversible for all positive integers n.

Corollary 5. *$CA{-}90_{\{a-*,*-b\}}(n)$ are reversible for all positive integers n.*

Lemma 10. *$CA{-}90_{\{*-*,*\}}(n)$ are not reversible for all positive integers n.*

Proof. It directly follows from $f_{90(n),\{*-*,*\}}(1^n) = f_{90(n),\{*-*,*\}}(0^n) = 0^n$.

7 $CA{-}\{102, 60, 153, 195\}(n)$

It is obvious that $f_{102}(abc) = b + c$, $f_{60}(abc) = a + b$, $f_{153}(abc) = (b + c)^-$, $f_{195}(abc) = (a + b)^-$ for all triples $abc \in Q^3$, and $[102, 60, 153, 195]$. Hence by Corollary 1 cellular automata $CA{-}\{60, 153, 195\}(n)$ are isomorphic to $CA{-}102(n)$ and so we will inspect only $CA{-}102(n)$.

Lemma 11. *Let $x = x_1x_2 \cdots x_n \in Q^n$. If $f_{102(n),\{a-b,a-*,*-b,*-*,*\}}(x) = 0^n$, then $x_1 = x_2 = \cdots = x_{n-1} = x_n$.*

Proof. Set $f = f_{102}$. The condition $f_{(n),\{a-b,a-*,*-b,*-*,*\}}(x) = 0^n$ means that

$$f(*\,x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-2}x_{n-1}x_n)f(x_{n-1}x_n\,*) = 0^n.$$

Hence we have $x_i + x_{i+1} = f(x_{i-1}x_ix_{i+1}) = 0$ for all $i = 1, 2, \cdots, n-1$.

Lemma 12. *1. $CA{-}102_{\{a-b,*-b\}}(n)$is reversible iff so is $CA{-}102_{\{0-0,*-0\}}(n)$.*
2. $CA{-}102_{\{0-0,*-0\}}(n)$ are reversible for all positive integers n.

Proof. 1. It follows from a fact that $f_{102(n),\{a-b,*-b\}}(x) = f_{102(n),\{0-0,*-0\}}(x) + 0^{n-1}b$ holds.

2. Set $f = f_{102}$. Since $f_{(n),\{0-0,*-0\}} : Q^n \to Q^n$ is additive (modulo 2), we will show that $f_{(n),\{0-0,*-0\}}(x) = 0^n$ implies $x = 0^n$ in $CA{-}102_{\{0-0,*-0\}}(n)$ for all positive integers n.
(i) It holds that $f_{(1),\{0-0,*-0\}}(x_1) = f(*x_10) = x_1$ and $f_{(2),\{0-0,*-0\}}(x_1x_2) = f(*x_1x_2)f(x_1x_20) = (x_1 + x_2)x_2$. Hence $CA{-}102_{\{0-0,*-0\}}(n)$ is reversible for $n = 1, 2$.
(ii) Assume that $CA{-}102_{\{0-0,*-0\}}(n)$ for $n \geq 2$ is reversible, i.e. $f_{(n),\{0-0,*-0\}}(x_1 \cdots x_n) = 0^n$ implies $x_1 \cdots x_n = 0^n$. We now see that

$$f_{(n+1),\{0-0,*-0\}}(x_1 \cdots x_nx_{n+1}) = 0^{n+1} \text{ implies } x_1 \cdots x_nx_{n+1} = 0^{n+1}.$$

Assume $f_{(n+1),\{0-0,*-0\}}(x_1 \cdots x_nx_{n+1}) = 0^{n+1}$. Then we have

$$\begin{aligned} f_{(n),\{0-0,*-0\}}(x_1 \cdots x_n) &= f(*x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-1}x_n0) \\ &= 0^{n-1}f(x_nx_{n+1}0) \\ &= 0^n, \end{aligned}$$

since $x_{n-1}x_n = x_nx_{n+1}$ by Lemma 11. Hence by the induction hypothesis we have $x_1 \cdots x_n = 0^n$. Therefore $CA{-}102_{\{0-0,*-0\}}(n+1)$ is reversible.

Corollary 6. *$CA{-}102_{\{a-b,*-b\}}(n)$ are reversible for all positive integers n.*

Lemma 13. *$CA{-}102_{\{a-*,*-*,*\}}(n)$ are not reversible for all integers n.*

Proof. It is direct from $f_{102(n),\{a-*,*-*,*\}}(1^n) = f_{102(n),\{a-*,*-*,*\}}(0^n) = 0^n$.

8 $CA{-}\{150, 105\}(n)$

It is obvious that $f_{150}(abc) = a+b+c$, $f_{105}(abc) = (a+b+c)^-$ for all triples $abc \in Q^3$, $f_{150}^- = f_{105}$, $[150, 150, 150, 150]$ and $[105, 105, 105, 105]$. Hence by Lemma 3 the reversibility of $CA{-}105(n)$ and $CA{-}150(n)$ are equivalent and so we will inspect only $CA{-}150(n)$.

Lemma 14. *Let $x = x_1x_2\cdots x_n \in Q^n$. If $f_{150(n),\{a-b,a-*,*-b,*-*,*\}}(x) = 0^n$, then $x_i = x_{i+3}$ for all $i = 1, 2, \cdots, n-3$.*

Proof. Set $f = f_{150}$. The condition $f_{(n),\{a-b,a-*,*-b,*-*,*\}}(x) = 0^n$ means

$$f(*\,x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-2}x_{n-1}x_n)f(x_{n-1}x_n\,*) = 0^n.$$

Hence we have

$$x_i + x_{i+3} = f(x_ix_{i+1}x_{i+2}) + f(x_{i+1}x_{i+2}x_{i+3}) = 0 + 0 = 0$$

for all $i = 1, 2, \cdots, n-3$.

Corollary 7. *$CA{-}150_{\{*-*,*\}}(n)$ are reversible iff $n \neq 0 \pmod 3$.*

Proof. Set $f = f_{150}$. First we will show that $f_{(n),\{*-*,*\}}(x) = 0^n$ implies $x = 0^n$ in $CA{-}150_*(n)$ for $n \neq 0 \pmod 3$.
(i) It holds that $f_{(1),\{*-*,*\}}(x_1) = x_1$, $f_{(2),*-*}(x_1x_2) = x_2x_1$, $f_{(2),*}(x_1x_2) = x_1x_2$, $f_{(4),*-*}(x_1x_2x_3x_4) = x_2(x_1+x_2+x_3)(x_2+x_3+x_4)x_3$ and $(f_{(4),*})^2(x_1x_2x_3x_4) = x_1x_2x_3x_4$. Hence $CA{-}150_{\{*-*,*\}}(n)$ is reversible for $n = 1, 2, 4$.
(ii) Assume that $CA{-}150_{\{*-*,*\}}(n)$ is reversible for $n \geq 2$, i.e. $f_{(n),\{*-*,*\}}$ is injective. We will see that $f_{(n+3),\{*-*,*\}}$ is injective.
Assume $f_{(n+3),\{*-*,*\}}(x_1x_2\cdots x_nx_{n+1}x_{n+2}x_{n+3}) = 0^{n+3}$. Then we have

$$\begin{aligned} f_{(n),\{*-*,*\}}(x_1x_2\cdots x_n) &= f(\{x_1, x_n\}\,x_1x_2)f(x_1x_2x_3)\cdots f(x_{n-1}x_n\,\{x_n, x_1\}) \\ &= f(\{x_1, x_{n+3}\}\,x_1x_2)0^{n-2}f(x_{n+2}x_{n+3}\,\{x_{n+3}, x_1\}) \\ &= 0^n, \end{aligned}$$

since $x_n = x_{n+3}$ and $x_{n-1} = x_{n+2}$ by Lemma 14. Hence by the induction hypothesis we have $x_1x_2\cdots x_n = 0^n$ and so $x_{n+1} = x_{n+2} = x_{n+3} = 0$. Finally we see that if $n = 0 \pmod 3$ then $f_{150(n),\{*-*,*\}}$ is not injective. It follows at once from a fact that $f_{150(3k),\{*-*,*\}}((101)^k) = 0^{3k}$ holds for all positive integers k.

Lemma 15. *1. $CA{-}150_{a-b}(n)$ is reversible iff so is $CA{-}150_{0-0}(n)$.*
2. $CA{-}150_{a-*}(n)$ is reversible iff so is $CA{-}150_{0-*}(n)$.

3. $CA-150_{0-0}(n)$ is reversible iff $n \neq 2 \pmod 3$.
4. $CA-150_{0-*}(n)$ is reversible iff $n \neq 1 \pmod 3$.

Proof. (1,2) The result comes direct from $f_{150(n),a-b}(x) = f_{150(n),0-0}(x)+a0^{n-2}b$ and $f_{150(n),a-*}(x) = f_{150(n),0-*}(x)+a0^{n-1}$. (3,4) This can be shown in the same way as the proof of Corollary 7.

Corollary 8. *1. $CA-150_{a-b}(n)$ is reversible iff $n \neq 2$ (mod3).*
2. $CA-150_{a-*}(n)$ is reversible iff $n \neq 1$ (mod3).

9 $CA-\{166, 180, 154, 210, 89, 75, 101, 45\}(n)$

It is obvious that $f_{166}(abc) = (a+1)b+c$, $f_{180}(abc) = a+b(c+1)$, $f_{154}(abc) = a(b+1)+c$, $f_{210}(abc) = a+(b+1)c$, $f_{89}(abc) = (a+1)b+c+1$, $f_{75}(abc) = a+b(c+1)+1$, $f_{101}(abc) = a(b+1)+c+1$, $f_{45}(abc) = a+(b+1)c+1$ for all triples $abc \in Q^3$, $f_{\overline{166}} = f_{89}$, $[166, 180, 154, 210]$ and $[89, 75, 101, 45]$. Hence by Corollary 1 and Lemma 3 the reversibilities of cellular automata $CA-\{166, 180, 154, 210, 89, 75, 101, 45\}(n)$ are equivalent and so we will inspect only $CA-166(n)$.
We now use the following notation:

$$x_1^{(1)} x_2^{(1)} \cdots x_n^{(1)} = f_{166(n),*}(x_1 x_2 \cdots x_n),$$

$$x_1^{(k+1)} x_2^{(k+1)} \cdots x_n^{(k+1)} = f_{166(n),*}(x_1^{(k)} x_2^{(k)} \cdots x_n^{(k)})$$

for each configuration $x_1 x_2 \cdots x_n \in Q^n$. In other words,

$$x_1^{(k)} x_2^{(k)} \cdots x_n^{(k)} = (f_{166(n),*})^k (x_1 x_2 \cdots x_n),$$

where $(f_{166(n),*})^k$ is k-th composition of $f_{166(n),*}$. Also a configuration $x_1 x_2 \cdots x_n$ in $CA-166_*(n)$ is extended to an infinite configuration $(x_m)_{m \in \mathbb{Z}}$ such that $x_m = x_{m'}$ if $m = m' \pmod n$.

Lemma 16. *In $CA-166_*(n)$ an identity*

$$x_m^{(2^k)} = (x_{m-2^k} + 1) \Pi_{j=1}^{2^k} x_{m-2^k+2j-1} + x_{m+2^k}$$

holds for all natural numbers m and k.

Proof. (i) In the case of $k = 0$ an identity

$$x_m^{(1)} = (x_{m-1} + 1) x_m + x_{m+1}$$

holds, since $f_{166}(abc) = (a+1)b+c$.
(ii) Set $\delta = f_{166(n),*}$. Assume that the identity holds for a natural number k. Note that

$$\delta^{2^{k+1}}(x) = (\delta^{2^k} \delta^{2^k})(x) = \delta^{2^k}(x_1^{(2^k)} x_2^{(2^k)} \cdots x_n^{(2^k)}).$$

Hence by using the induction hypothesis twice we have

$$\begin{aligned}
x_m^{(2^{k+1})} &= (x_{m-2^k}^{(2^k)} + 1)\Pi_{j=1}^{2^k} x_{m-2^k+2j-1}^{(2^k)} + x_{m+2^k}^{(2^k)} \\
&= \{(x_{m-2^{k+1}} + 1)\Pi_{j=1}^{2^k} x_{m-2^{k+1}+2j-1} + x_m + 1\} \\
&\quad \Pi_{j=1}^{2^k}\{(x_{m-2^{k+1}+2j-1} + 1)\Pi_{i=1}^{2^k} x_{m-2^{k+1}+2j+2i-2} + x_{m+2j-1}\} \\
&\quad +(x_m + 1)\Pi_{j=1}^{2^k} x_{m+2j-1} + x_{m+2^{k+1}} \\
&\quad \{\ m - 2^{k+1} + 2j + 2i - 2 = m \text{ for } i = 2^k - j + 1\ \} \\
&= \{(x_{m-2^{k+1}} + 1)\Pi_{j=1}^{2^k} x_{m-2^{k+1}+2j-1} + x_m + 1\}\Pi_{j=1}^{2^k} x_{m+2j-1} \\
&\quad +(x_m + 1)\Pi_{j=1}^{2^k} x_{m+2j-1} + x_{m+2^{k+1}} \\
&= (x_{m-2^{k+1}} + 1)\Pi_{j=1}^{2^{k+1}} x_{m-2^{k+1}+2j-1} + x_{m+2^{k+1}},
\end{aligned}$$

which completes the proof.

Corollary 9. *$CA{-}166_*(n)$ is reversible iff $n = 1$ (mod 2)*

Proof. It is trivial that $f_{166(1),*}(x_1) = x_1$ and so $f_{166(1),*}$ is bijective. Next we will see that every transition function $f_{166(2n-1),*} : Q^{2n-1} \to Q^{2n-1}$ of $CA{-}166_*(2n-1)$ is bijective for all integers $n \geq 2$. Take a unique integer k such that $2^k < 2n-1 < 2^{k+1}$. By the virtue of the last lemma 16 the m-th state of $(f_{166(2n-1),*})^{2^k}(x)$ is given by

$$x_m^{(2^k)} = (x_{m-2^k} + 1)\Pi_{j=1}^{2^k} x_{m-2^k+2j-1} + x_{m+2^k}$$

Set $j = n$. (Remark $2 \leq n \leq 2^k$.) Then $m - 2^k + 2j - 1 = m - 2^k \pmod{2n-1}$ and so $(x_{m-2^k} + 1)x_{m-2^k+2j-1} = (x_{m-2^k} + 1)x_{m-2^k} = 0$. Hence we have

$$x_m^{(2^k)} = x_{m+2^k},$$

which proves the bijectivity of $f_{166(2n-1),*}$. Finally we see that every transition function $f_{166(2n),*}$ is not injective. It follows at once from

$$f_{166(2n),*}((01)^n) = f_{166(2n),*}((10)^n) = 0^{2n}.$$

This completes the proof.

Lemma 17. *1. $CA{-}166_{\{1-b,0-*,*-b\}}(n)$ are not reversible for all positive integers n.*
2. $CA{-}166_{\{0-b,1-*,*-*\}}(n)$ are not reversible for any positive integers $n \geq 2$.

Proof. It is direct from the following equations;

- $f_{166(n),0-b}(110^{n-2}) = f_{166(n),0-b}(0^n)$ for $n \geq 2$
- $f_{166(1),0-*}(x_1) = 0$
- $f_{166(2),0-*}(10) = f_{166(2),0-*}(01)$
- $f_{166(n),0-*}(110^{n-2}) = f_{166(n),0-*}(0^n)$ for $n \geq 3$
- $f_{166(1),1-b}(x_1) = b$
- $f_{166(n),\{1-b,1-*\}}(10^{n-1}) = f_{166(n),\{1-b,1-*\}}(0^n)$ for $n \geq 2$
- $f_{166(n),*-b}(10^{n-1}) = f_{166(n),*-b}(0^n)$
- $f_{166(n),*-*}(10^{n-1}) = f_{166(n),*-*}(0^n)$ for $n \geq 2$

10 Conclusion

We have proved that several 1d CA with finite cell array, including cyclic CA simulated in [4], are reversible. Wolfram showed that for 1d CA with infinite cell array there exist the only six reversible CA with trivial triplet local rules. However this paper presented some nontrivial 1d reversible CA with finite cell array. Because QCA has universal computability as same as quantum Turing machines [8], the study of QCA is useful for research of quantum Turing machines. The reversible CA dealt with in this paper are special type of QCA. It is difficult to create a unitary transition matrix of QCA generally. But we can get easily a unitary transition matrix of product of a rotation matrix and transition matrix of reversible CA [4]. So the result of this paper is valuable for the research of not only QCA, but also quantum Turing machines. Our future work is to investigate how these reversible $CA - R(n)$ can be extended to generalised PQCA introduced by [4].

References

1. R.A. Dow: Additive Cellular Automata and Global Injectivity, Physica D 110 (1997) 67-91.
2. E. Goles and S. Martinez, Neural and automata networks - dynamical behavior and applications -, Kluwer Academic Publishers, Dordrecht (1990).
3. S. Inokuchi et al., Computational analysis of cellular automata with triplet transition rule. Research Reports on Information Science and Electrical Engineering of Kyushu University Vol. 1(1) (1996) 79-84 (in Japanese).
4. S. Inokuchi and Y. Mizoguchi, Generalized Partitioned Quantum Cellular Automata and Quantization of Classical CA, Int. Journ. of Unconventional Computing, Vol. 1 (2005) 149-160.
5. Y. Kobuchi and H. Nishio, Some regular state sets in the system of one-dimensional iterative automata, Information Sciences, vol.5 (1973) 199-216.
6. H.-Y. Lee and Y. Kawahara, Transition diagrams of finite cellular automata. Bull. Inform. Cybernet. Vol. 28 (1996) 47-69.
7. K. Morita and M. Harao, Computation universality of one-dimensional reversible (injective) cellular automata, Transactions of the IEICE, Vol. E 72 (1989) 758-762.
8. J. Watrous, On one-dimensional quantum cellular automata, Proceedings on the 36th Annual Symposium on Foundations of Computer Science (1995) 528-537.
9. S. Wolfram, Random sequence generation by cellular automata, Advances in Applied Mathematics, Vol. 7 (1986) 123-169.
10. S. Wolfram, A New Kind of Science, Wolfram Media, Inc. (2002) 435-457, 1017-1021.

A Computational Model for Self-assembling Flexible Tiles

Nataša Jonoska and Gregory L. McColm

Department of Mathematics, University of South Florida Tampa, FL 33620
{jonoska, mccolm}@math.usf.edu

Abstract. We present a theoretical model for self-assembling tiles with flexible branches motivated by DNA branched junction molecules. We encode an instance of a “problem” as a pot of such tiles, and a “solution” as an assembled complete complex without any free sticky ends (called ports), whose number of tiles is within predefined bounds. We develop an algebraic representation of this self-assembly process and use it to prove that this model of self-assembly precisely captures NP-computability when the number of tiles in the minimal complete complexes is bounded by a polynomial.

1 Introduction

Many researchers use weak chemical bonds to design and “grow” self-assembled nanostructures. Under thermodynamic equilibrium, molecules assemble using several types of non-covalent intermolecular interactions (e.g., hydrogen or ionic bonds), evolving into relatively stable structures [23,24]. This paper is motivated mainly by DNA self-assembly, employing Watson-Crick complementarity and hydrogen bonding, as a case study for molecular self-organization due to weak bonding. The interest comes from several major experimental developments that use DNA for constructing nanostructures, for information processing, and as a material for nanodevices.

Nanostructures. An essential part of the chemical engineering of self-assembling structures is the design of the molecular building blocks that will bind into larger, and more complex structures. Although naturally-occurring DNA is a linear molecule (considering its helix axis as a segment of a curve), DNA molecules can be constructed as stable branch points [31,35] and ultimately as more complex structures with lateral fusion of DNA helices such as double DX and triple crossover TX molecules [11,22,32]. DNA and other molecules have been used for self-assembly of two dimensional arrays [36,37], three-dimensional graph-like structures [15,30] and regular polyhedra [6,40], including the octahedron (by DX and PX molecules) [33].

Algorithmic self-assembly. Beginning with the initial successful experiment by Adleman [1] and more recently one from the same group solving an instance of SAT with 20 variables [5], computations by biomolecular protocols include, among others, binary addition (simulation of XOR) using triple cross-over molecules (tiles) [25], a 9-bit instance of the "knight problem" using RNA [10], and a small instance of the maximal clique problem using plasmids [13]. Recently, Winfree [29] used algorithmic self-assembly to obtain cellular automata like two-dimensional arrays of the Sierpinski triangle.

Nanodevices. Based on a B-Z transition of the DNA helix, a nano-mechanical device was introduced in [26]. Soon after, "DNA fuel" strands based on Watson-Crick hydrogen bonding were used to produce devices whose activity were controlled by DNA strands [38], [39]. The device introduced in [38] has two distinct positions, each obtained by adding a pair of DNA strands that hybridize with the device so that the molecule is either in the first or in the second position.

Theoretical observations. Despite notable advances in experimental molecular self-assembly, the theoretical understanding of this process is lacking. Algorithmically, it has been observed that DNA tiles can simulate Wang tiles and as such, are capable of simulating the transitions of a Universal Turing machine [36]. However, there is a real need for understanding the limitations, boundaries, and complexity of the process of self-assembly. Only a few theoretical results have been obtained, primarily for DNA assemblies using rigid tiles. The complexity, measured as the number of tiles needed for a unique assembly of $n \times n$ squares, is considered in [28], where it was observed that only $O(\log n)$ molecules are needed for this task. Comparison of such "shape" complexity with Kolmogorov complexity is investigated in [34]. The same model was used to theoretically observe a possible two-dimensional tile self-assembly of a cube [21]. In [2], computing the minimal number of tiles needed for unique self-assembly in a given shape proved to be NP-hard, and $O(\log n)$-approximation of the concentration of the tiles needed for fast assembly in the desired structure is computed. The question of whether a given set of tiles arrange in an infinite ribbon-like shape was proved to be undecidable [3]. In [16] and [17], the tiles are "flexible" in that they have extended bendable branches, the branches being labeled so that the assembly process is guided by complementary Watson-Crick labels; the algorithmic view is that the input is encoded as a collection of tiles to use, and an output (if any) is a sufficiently small complete complex, i.e., a complex of tiles with no free ports. This flexible tile model has been investigated in [18], which presented a heuristic model to predict the distribution of products of the self-assembly. In [27], a step-wise assembly of junction molecules is investigated and the computational complexity of the problem, whether a complete complex is produced by a given pot of a certain form is considered.

In this paper, we take a somewhat reverse course from [27], and ask what are the problems solvable by flexible tile assembly; we find that these are precisely the NP-complete problems. Specifically, we find that all flexible tile DNA computations within a polynomial bound are reducible to the NP-computable integer programming problem of [4] and [20], and conversely, that any NPTIME computation can be simulated by a flexible tile computation. This note is an extended abstract of results appearing in [19].

2 The Theoretical Model

The main building blocks used in our assembly model are motivated by branched junction DNA molecules with junctions ending with single stranded portions (sticky ends) that can bind to their Watson-Crick complementary sequence. It is also assumed that the structure of the junctions (and possibly the branches) is such that the branches are rather flexible and various connections are possible. A schematic view of this process is presented in Figure 1. It has been shown theoretically that several NP-complete problems such as 3SAT and 3-vertex-colorability can be solved by self-assembly of DNA graphs [15,16,17]. The coding of the problems is such that a solution is obtained if and only if the graph can be assembled. Also, experimental confirmation of DNA graph self-assembly has been obtained by flexible tiles in [15,30]. In what follows we develop the theoretical model and prove these initial observations in a general setting.

Let $H \subset \Sigma^*$ be a finite set of words over alphabet Σ that we call *port (bonding) types* and let θ: $H \to H$ be an involution. We call $\theta(\mathbf{h}) \in H$ the *complementary* string to $\mathbf{h}$ such that ports of types $\mathbf{h}$ and $\theta(\mathbf{h})$ bond. For each $\mathbf{h} \in H$ we assume that $\theta(\mathbf{h}) \neq \mathbf{h} = \theta(\theta(\mathbf{h}))$.

Fix H and θ for the rest of this paper. To ease notation we write $\hat{\mathbf{h}}$ for $\theta(\mathbf{h})$.

Definition 1. *A* $\underline{\text{tile type}}$ *over* (H, θ) *is a function* $\mathbf{t}$: $H \to \mathbb{N}$.

A tile of type $\mathbf{t}$ will have $\mathbf{t}(\mathbf{h})$ ports of type $\mathbf{h}$. If $|H| = k$, then a tile type can be written as a k-dimensional vector with non-negative integer entries; alternatively, a tile type can be regarded as a multiset of port bonding types. We call $d = d(\mathbf{t}) = \sum_{\mathbf{h}} \mathbf{t}(\mathbf{h})$ the *degree* of tile type $\mathbf{t}$. In order to get the tiles themselves, we first get a collection of "prototiles." A *prototile corresponding to tile type* $\mathbf{t}$ is a star-like graph with one central vertex of degree $d(\mathbf{t})$ indicating the center of the prototile and $d(\mathbf{t})$ vertices of degree one labeled with ports indicating the branches seeking to bond with complementary ports. A tile is thus a copy of a prototile, so we say that a tile t that is a copy of a prototile of type $\mathbf{t}$ is thus itself of type $\mathbf{t}$, and we write: for each $\mathbf{h} \in H$, $t(\mathbf{h}) = \mathbf{t}(\mathbf{h})$ means

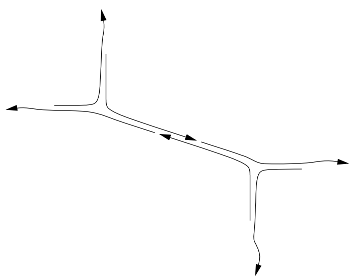


Fig. 1. Watson-Crick bonding of two DNA junction molecules. The double helix structure is not depicted for simplicity. The arrowhead indicates the $3'$ end also ending with a single stranded sticky end sequence.

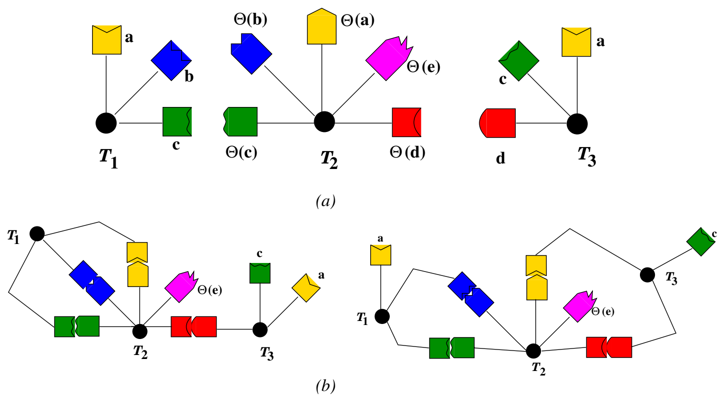


Fig. 2. Complexes. (a) Three tiles, with the central vertex indicated as a black circle, and the one-degree vertices with ports schematically presented with different colors and shapes. The complementary ports have compatible shapes, and same colors. (b) Two different (incomplete) complexes obtained by gluing the three tiles. They are of the same complex type despite their different structures, as they have the same multiset of free port types.

that t has exactly $\mathbf{t}(\mathbf{h})$ ports of type $\mathbf{h}$. Figure 2 (a) shows examples of three tiles with degrees 3, 5 and 3 respectively. The central vertex is represented with a black circle and ports are indicated with different colors and shapes.

We put tiles together to construct complexes. But first, we need to classify the pots that the complexes are assembled in.

Definition 2. *A* <u>pot type</u> *over* (H, θ) *is a set* $\mathbf{P}$ *of prototiles corresponding to tile types over* (H, θ) *such that for any* $\mathbf{h} \in H$ *and* $\mathbf{t} \in \mathbf{P}$*, if* $\mathbf{t}(\mathbf{h}) > 0$ *then there exists* $\mathbf{t}' \in \mathbf{P}$ *such that* $\mathbf{t}'(\hat{\mathbf{h}}) > 0$*.*

Thus no pot admits tiles with unattachable ports. A *pot* P is a set of tiles from $\mathbf{P}$.

Definition 3. *A* <u>complex</u> *over a pot type* $\mathbf{P}$ *is a pair* $\mathfrak{C} = \langle T, J \rangle$ *where* T *is a set of tiles with tile types in* $\mathbf{P}$ *and* J *is a set of unordered pairs* $e = \{(t, \mathbf{h}), (t', \mathbf{h}')\}$ *satisfying the following two properties:*

- *For each* $e = \{(t, \mathbf{h}), (t', \mathbf{h}')\} \in J$*,* $t, t' \in T$*,* $t(\mathbf{h}), t'(\mathbf{h}') > 0$*,* $\mathbf{h}' = \hat{\mathbf{h}}$ *(*e indicates the connection between two complementary ports)*, and*
- *the cardinality* $|\{e \mid (t, \mathbf{h}) \in e\}| \leq t(\mathbf{h})$ (this prevents the tile from making more connections than it has ports)*.*

The <u>type</u> *of a complex* $\mathfrak{C} = \langle T, J \rangle$ *is the function* $\text{type}(\mathfrak{C})\colon H \to \mathbb{N}$ *defined by*

$$\text{type}(\mathfrak{C})(\mathbf{h}) = \sum_{t \in T} t(\mathbf{h}) - |\{e \mid (t, \mathbf{h}) \in e\}|$$

Thus the type of $\mathfrak{C}$ indicates the types of its ports, free (i.e., those not appearing in J). Similarly, as with tile types, the complex type can be viewed as a k-dimensional vector with non-negative integer entries. Note that each tile can be considered as a complex where the set J is empty and the set T is a singleton. Then the type of tile t is equal to the type of the complex it represents. We would like to distinguish tiles from complexes merely to indicate the fact that complexes are assembled from tiles.

We assume that assembly occurs in an extremely dilute solution, so that when two complexes meet, *all* of their complementary free sticky ends join up so that there are no complementary free ports. (This is where the flexibility of the tiles is so critical.) Thus:

Definition 4. *A complex $\mathfrak{C}$ over (H, θ) is* <u>stable</u> *if, for each $\mathbf{h} \in H$, either* $\text{type}(\mathfrak{C})(\mathbf{h}) = 0$ *or* $\text{type}(\mathfrak{C})(\hat{\mathbf{h}}) = 0$.

In this paper, we assume that all complexes are stable unless otherwise indicated.

As an example, consider Figure 2 (b), in which the ports of three tiles are (maximally) connected to produce either of the two non-isomorphic complexes of the same type. The ports are shown with different colors and shapes.

More generally, we may join complexes to obtain bigger complexes: if $\mathfrak{C}_1 = \langle T_1, J_1 \rangle$ and $\mathfrak{C}_2 = \langle T_2, J_2 \rangle$ are two complexes, then we can *glue* them together by connecting up their complementary ports to get a complex $\mathfrak{C} = \langle T, J \rangle$. There may be several non-equivalent ways to do this. Let ΔJ be a set of unordered pairs $\{(t_1, \mathbf{h}_1), (t_2, \mathbf{h}_2)\}$ such that a free port of type $\mathbf{h}_1$ from tile $t_1 \in T_1$ connects to a free port of type $\mathbf{h}_2 = \hat{\mathbf{h}}_1$ from tile $t_2 \in T_2$. We have the following definition:

Definition 5. *We say that $\mathfrak{C} = \langle T, J \rangle$ is obtained by* gluing *complexes $\mathfrak{C}_1 = \langle T_1, J_1 \rangle$ and $\mathfrak{C}_2 = \langle T_2, J_2 \rangle$ if*

$$T = T_1 \cup T_2 \qquad \text{and} \qquad J = J_1 \cup J_2 \cup \Delta J,$$

with the restriction that for each $\mathbf{h} \in H$, as many ports of type $\mathbf{h}$ as possible are joined, i.e., for each $\mathbf{h}$, $\text{type}(\mathfrak{C})(\mathbf{h}) = |\text{type}(\mathfrak{C}_1)(\mathbf{h}) - \text{type}(\mathfrak{C}_2)(\hat{\mathbf{h}})|$.

A complex is called <u>*complete*</u> if it has no free ports, i.e., if for all bonding port types $\mathbf{h}$, $\overline{\text{type}(\mathfrak{C})(\mathbf{h}) = 0}$. For a pot P we denote with $\mathcal{C}(P)$ the set of all complete complexes that can be obtained by tiles from P, and for a pot type $\mathbf{P}$, let $\mathcal{C}(\mathbf{P}) = \bigcup_{P \in \mathbf{P}} \mathcal{C}(P)$. Note that if P is finite, so is $\mathcal{C}(P)$. The case when $\mathcal{C}$ is finite is discussed in [19].

3 Example: The 3SAT Problem

The *satisfiability problem* (SAT) asks whether for a given boolean formula φ, there is an assignment of {TRUE, FALSE} (or $\{T, F\}$) to the variables in φ that would assign to φ the value TRUE. (This is the archetypic NP-complete problem [7].) Writing "+" for "or" and concatenation for "and," a formula φ is in *conjunctive normal form* (CNF) if it can be expressed as $\varphi = c_1 c_2 \cdots c_r$ where each c_i is a clause of the form $(a_1 + \cdots + a_k)$, and each a_i is a *literal*, i.e., either a variable v or a negation of a variable $\bar{v}$. It is in *k-conjunctive normal form* (k-CNF) if each conjunctive clause contains at most k literals. The satisfiability problem for 3-CNF formulas is known as 3SAT [7]. Consider for example:

$$\varphi = (\bar{x} + y + \bar{z})(x + \bar{y} + z)(\bar{x} + \bar{y} + z). \tag{1}$$

This Formula 1 has value T for the assignments $(x, y, z) \in \{(F, F, F), (T, T, T), (T, F, F), (F, T, T), (F, F, T)\}$, and F for any other assignment.

For each 3-CNF formula φ we associate a pot type $\mathbf{P}$ with the following tiles. The alphabet from which the port types are taken is

$$H = \{T, F, \hat{T}, \hat{F}\} \cup \{x, \bar{x}, \hat{x}, \hat{\bar{x}} \mid x \text{ is a variable }\} \cup \{c, \hat{c} \mid c \text{ is a clause }\}.$$

The symbols with $\hat{}$ indicate the θ complements. The set of port types consists of three letter words which we write as ordered triples for easier reading.

$$H = \big\{(\iota, c, a), \theta(\iota, c, a) \mid \iota \in \{T, F\},\ c \text{ a clause, } a \text{ a literal }\big\}$$
$$\cup \big\{(x, \bar{x}, \iota), \theta(x, \bar{x}, \iota) \mid x \text{ is a variable, } \iota \in \{T, F\}\big\}.$$

The tiles differ in the assignment of the free ports. For each clause c_i there are seven tiles with three free ports of types (ι, c, a), a being a literal of c and ι being a truth assignment. Each tile corresponds to a truth assignment to the variables making the clause it belongs to true: its ports will be of types (ι, c, a), for a having truth assignment ι. (The truth assignment making the clause false gets no tile.) For example, the tile type corresponding to the clause $c_1 = (\bar{x}+y+\bar{z})$ for Formula 1 with assignment $(x \to F, y \to T, z \to T)$ has ports of types $(T, c_1, \bar{x})$, (T, c_1, y), and $(F, c_1, \bar{z})$ indicating $(\bar{x} \to T, y \to T, \bar{z} \to F)$. This is depicted in Figure 3 (a).

For each variable x, if x appears in s clauses, we associate x with two tiles, each with $s + 1$ ports. Each such tile corresponds to one of the two possible truth values of the variable, T or F. Each of s ports is complementary to the

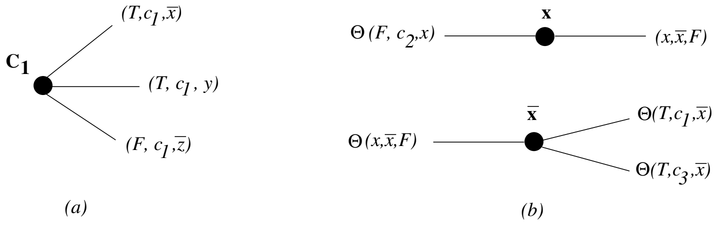


Fig. 3. (a) Tile for clause c_1 for Formula 1 with assignment (F, T, T). (b) Tiles corresponding to variable x and $\bar{x}$ (with value F for x) in 1.

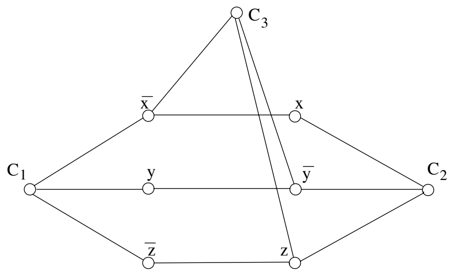


Fig. 4. A complete complex for Formula 1. Connections between complemenary ports are treated as edges in the graph.

corresponding port in one of the corresponding clause tiles (encoding θ(truth value, clause, variable)). The additional $(s+1)$st port is connected to the tile of the negation of the variable. Similarly, such tiles are encoded for the complements of the variables. The tiles corresponding to x and $\bar{x}$ in Formula 1 assigning value F to x are depicted in Figure 3 (b).

Any complete complex has to have at least one tile for each clause, and one for each variable (two if the variable and its negation both appear). A complete complex with exactly one tile for each clause represents a truth assignment satisfying the given 3-CNF formula (for details see [15]). Such a complete complex for the formula (1) is depicted in Figure 4.

On the other hand, if there is no satisfying truth assignment, complete complexes may still assemble. However the condition that for each clause and each variable there is precisely one corresponding tile in the complex will not be satisfied. Moreover, all such complexes contain larger (integer multiple) number of tiles than the number of clauses and variables. See [16] for an explanation of multiple covers.

Hence, a way to tell if the inputted 3-CNF formula is satisfiable is to determine if there are sufficiently small complete complexes in the pot. In the following we specify what "sufficiently small" means; we start with a system of indexes for guiding the construction of complexes.

Definition 6. *Fix a graph $\mathfrak{G} = \langle V, E \rangle$. A pot type* **P** *is* properly indexed *by $\mathfrak{G}$ if there is a complete complex $\mathfrak{C}$ that consists of tiles of distinct types, labeled by vertices $g \in V$, such that the map $t \mapsto \text{label}(t)$ is an isomorphism of $\mathfrak{C}$ onto $\mathfrak{G}$.*

Thus a complete complex of a properly indexed pot type (indexed by a graph $\mathfrak{G}$) is isomorphic to $\mathfrak{G}$, or of the form of several copies of $\mathfrak{G}$, all tangled together.

Thus in the case of 3SAT the pot type is properly indexed by the graph corresponding to the formula if and only if the formula has a solution. Moreover, the graph that properly indexes the pot is unique up to an isomorphism. The proof of the following proposition follows from basic topological properties of graphs, and we omit the proof.

Proposition 1. *Let* $\mathbf{P}$ *be a pot type and assume that there is precisely one graph* $\mathfrak{G} = \langle V, E\rangle$ *such that* $\mathbf{P}$ *is properly indexed by* $\mathfrak{G}$*. Suppose that* $\mathfrak{G}$ *is connected. Then any complete complex in* $\mathcal{C}(\mathbf{P})$ *has* $n|V|$ *tiles, for some integer* n*.*

We can design properly indexed pots so that a complete complex witnessing, say, the solution of a problem has $|V|$ tiles, while a spurious complete complex has at least $2|V|$ tiles. For example, a 3-CNF formula generates a pot, which provides a solution (i.e., a satisfying truth assignment) in a proper complex; if there is no satisfying truth assignments, there is no proper complex. Thus a formula of k clauses and m variables is satisfied by some truth assignment iff the corresponding pot admits a complete complex of $k + 2m$ tiles; any larger complexes will have at least $2k + 4m$ tiles.

Definition 7. *A pot type* $\mathbf{P}$ *is* weakly satisfiable within bound b *if it admits a complete complex of at most* b *tiles.*

4 Computing NP Problems by Flexible-Tile Assembly

This section shows how NPTIME computations can be accomplished by (flexible tile) assembly within given bounds. First we fix the nomenclature. Consider a finite alphabet Σ, with a symbol $\# \notin \Sigma$ for “blank.” We consider problems of recognizing a formal language L over alphabet Σ, i.e., for an alphabet Σ and a language $L \subseteq \Sigma^*$, given input $x \in \Sigma^*$, is $x \in L$?

Definition 8. *A* Deterministic Turing Machine *is a tuple* $(Q, \Sigma, \delta, q_0, \#)$*, such that:*

– *The set* Q *is the set of* states *of the machine;* q_0 *is the* initial state *and* q_F *is the* terminal state*.*
– *The finite set* Σ *is the* alphabet*, which does not contain the “blank” character* $\#$*.*
– *The* transition function δ *is a partial function* $(Q \times (\Sigma \cup \{\#\})) \to (Q \times \Sigma \times \{L, 0, R\})$*. If* $\delta(q, \sigma) = (q', \sigma', \xi)$*, and if the tapehead is reading* $\sigma \in \Sigma \cup \{\#\}$ *while the machine is in state* q*, it replaces* σ *with* σ' *and moves* L*eft if* $\xi = L$ *or* R*ight if* $\xi = R$ *(or for* $\xi = 0$*, does not move), and changes its state to* q'*. Each such transition is a* step *denoted* $(q, \sigma) \to (q', \sigma', \xi)$*. The machine* halts *in state* q*, reading symbol* σ*, if* $\delta(q, \sigma)$ *is undefined.*
– *The machine starts in the initial state* q_0*, with the tapehead at the leftmost square of the input string* $x \in \Sigma^*$*. If it halts, the input string is accepted.*

The machine is a Non-Deterministic Turing Machine (NTM) *if, for each state* q *and character* σ*,* $\delta(q, \sigma)$ *is a nonempty subset of* $Q \times \Sigma \times \{L, 0, R\}$ *such that a transition is a step* $(q, \sigma) \to (q', \sigma', \xi)$ *if* $(q', \sigma', \xi) \in \delta(q, \sigma)$*.*

In the case of an NTM we assume that the machine proceeds by choosing successive actions out of the available options.

A language $L \subseteq \Sigma^*$ is *PTIME computable* if there exists a DTM M and a polynomial p such that for any input $x \in \Sigma^*$, M halts within $p(|x|)$ steps if and only if $x \in L$. Call a language $L \subseteq \Sigma^*$ *NPTIME computable* if there exists an NTM M and a polynomial p such that for any input $x \in \Sigma^*$, M can make choices in its computation to eventually halt in accepting state within $p(|x|)$ steps if and only if $x \in L$. These machines are called *PTIME acceptors* or *NPTIME acceptors*, respectively, as they "accept" their respective languages. Incidentally, a PTIME *program* or *algorithm* for converting one type of problem into another is a DTM M (with associated polynomial p) such that for any input x, within $p(|x|)$ steps the content of the tape will be a representation of the desired output.

Definition 9. *The* <u>Flexible Tile Assembly Polynomial time</u> *(or FTAP) class of languages is the class of languages $L \subseteq \Sigma^*$ such that there is a polynomial p and a PTIME algorithm converting any $x \in \Sigma^*$ into a pair $(\mathbf{P}_x, b)$ where:*

– *the pot type $\mathbf{P}_x$ is weakly satisfiable within bound b if and only if $x \in L$, and*
– *the bound b satisfies $b \leq p(|x|)$.*

We use tile assembly to construct a complex representing an entire NTM computation.

A *configuration* of a TM is a tuple (q, m, s), where q is the state of the machine, m is the position of the tapehead, and s is the string currently on the tape. Thus a *computation* of a Turing Machine is a sequence of configuration $C_0, C_1, \ldots$, where C_0 is an initial configuration, and for each i, C_{i+1} is obtained from C_i by a transition step. The computation terminates if and only if machine halts.

Here is the first half of the main theorem.

Theorem 1. *If a language $L \subseteq \Sigma^*$ is NPTIME computable, then it is in FTAP.*

Sketch of proof. Fix a polynomial p, and let L be accepted by the NTM M, which accepts strings of length n within $p(n)$ steps – or not at all. We construct a pot type such that each prototile in the pot type represents one tape square at a position m at a time t. For each input string x, if M accepts x it must do so within $p(|x|)$ steps. Set $t_x = p(|x|)$. Then the number of squares on the tape of the machine that is visited by the head is bounded by $n = 2t_x + 1$. Denote $N_{\text{time}} = \{0, 1, \ldots, t_x\}$ and $N_{\text{tape}} = \{-t_x, -t_x + 1, \ldots, 0, 1, \ldots, t_x\}$.

The set of prototiles is a subset of $(\Sigma \cup \{\#\}) \times N_{\text{tape}} \times N_{\text{time}} \times X^3 \times (Q \cup \{0\})^2$ where $X = \{-1, 0, 1\}$. Note that the number of prototiles is bounded by a polynomial in $|x|$. Intuitively, at time t and position m, there is a symbol σ in the square. The prototile corresponding to this square is $(\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, q'))$. We define $\xi_0 = 1$ if the tapehead is at this square at time t, and $\xi_0 = 0$ otherwise. If $\xi_0 = 1$, we define ξ_- and ξ_+ to represent the previous and next moves of the tapehead, otherwise they remain 0. The head had moved from the left ($\xi_- = 1$) or the right ($\xi_- = -1$), and the head will move either to the left ($\xi_+ = -1$) or the right ($\xi_+ = 1$). And if the machine entered square m while in a state q at time t, then it will change to a state q' at time $t+1$. Finally, if the computation halted at time t, $q' = q$ and $\xi_+ = 0$. If the tapehead is not positioned at the square m at time t, then $(\xi_-, \xi_0, \xi_+) = (0, 0, 0)$ and $(q, q') = (0, 0)$.

Given a prototile $\alpha = (\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, q'))$ and a Turing Machine M having transition function δ, we say that α is *consistent* with M if:

- $\xi_0 = 0$, implies $\xi_- = \xi_+ = 0$ and $q = q' = 0$,
- $\xi_0 = 1$ if $\delta(q, \sigma) = (q', \sigma', \xi)$ for $\xi \in \{R, 0, L\}$ and some $\sigma' \in \Sigma$.

Our pot type $\mathbf{P} = \mathbf{P}_M$ consists of the maximal set of prototiles consistent with M such that their second and third entries satisfy $m \in N_{\text{tape}}$ and $t \in N_{\text{time}}$.

There are eight codes for ports available, of which any particular tile may have degree from two to six. The degree of a tile $\alpha = (\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, q'))$ is determined by

$$d(\alpha) = \begin{cases} 2 \text{ if } m \in \{0, n\} \text{ and } t \in \{0, t_x\} & \textsf{if a corner tile;} \\ 3 \text{ if } m \in \{0, n\} \text{ xor } t \in \{0, t_x\} & \textsf{if a side tile;} \\ 4 \text{ if } \xi_0 = 0 & \textsf{if a central tile with no tapehead;} \\ 5 \text{ if } q' = q_F & \textsf{if the computation halts;} \\ 6 \text{ if } \xi_0 = 1 \text{ and } q' \neq q_F & \textsf{if the tapehead is present, not halting.} \end{cases}$$

We note that there are other kinds of tiles that encode specific boundary conditions, such as the head reaching a corner during the computation, but we do not include these details. Recall that a tile represents a particular mth square of the Turing machine at a particular time t. The idea is that successive rows of tiles represent successive configurations of the Turing Machine, starting with the initial (northernmost) row, and heading south. Several typical tiles in the computation assembly are depicted in Figure 5. Comparing to the simulation of a TM by rigid tiles (see for ex. [36,34]) one has to be careful to "force" the assembly into a rectangular grid using port codes. Each code indicates the current position of a given tile at the computation process of the Turing Machine (i.e. position m and time step t) as well as when necessary additional information such as the symbol, state and movement of the machine at a particular instance. This is depicted in Figure 5. The complementary ports are indicated by θ. Note that when the complex assembles, the "top" row represents the initial configuration with tiles encoding x on squares $0, \ldots, |x| - 1$. The rest of the tiles in the "top" row carry the blank character $\#$, the head is on square 0 in state q_0.

The tiles assemble into a rectangular grid, with each horizontal row of tiles giving a configuration of the machine, with successive rows southward representing successive configurations, and each vertical column representing a particular square. The diagonal connections represent motions of the head. The tiles are designed so that there are no north, south, east, or west ports off the end of the complex, so the complex will be complete if and only if there are no diagonal ports sticking out the bottom, i.e., if and only if the computation has halted within t_x steps.

So suppose L is NPTIME computable, i.e., there is a polynomial p such that for any x, $x \in L$ if and only if the NTM accepting L halts in a terminal state within $p(|x|)$ steps. If $x \in L$, $t_x = p(|x|)$ gives us a complete complex of at most $b = (t_x + 1)(2t_x + 1) = 2p(|x|)^2 + 3p(|x|) + 1$ tiles, a polynomial bound. If $x \notin L$, there is no complete complex at all. ∎

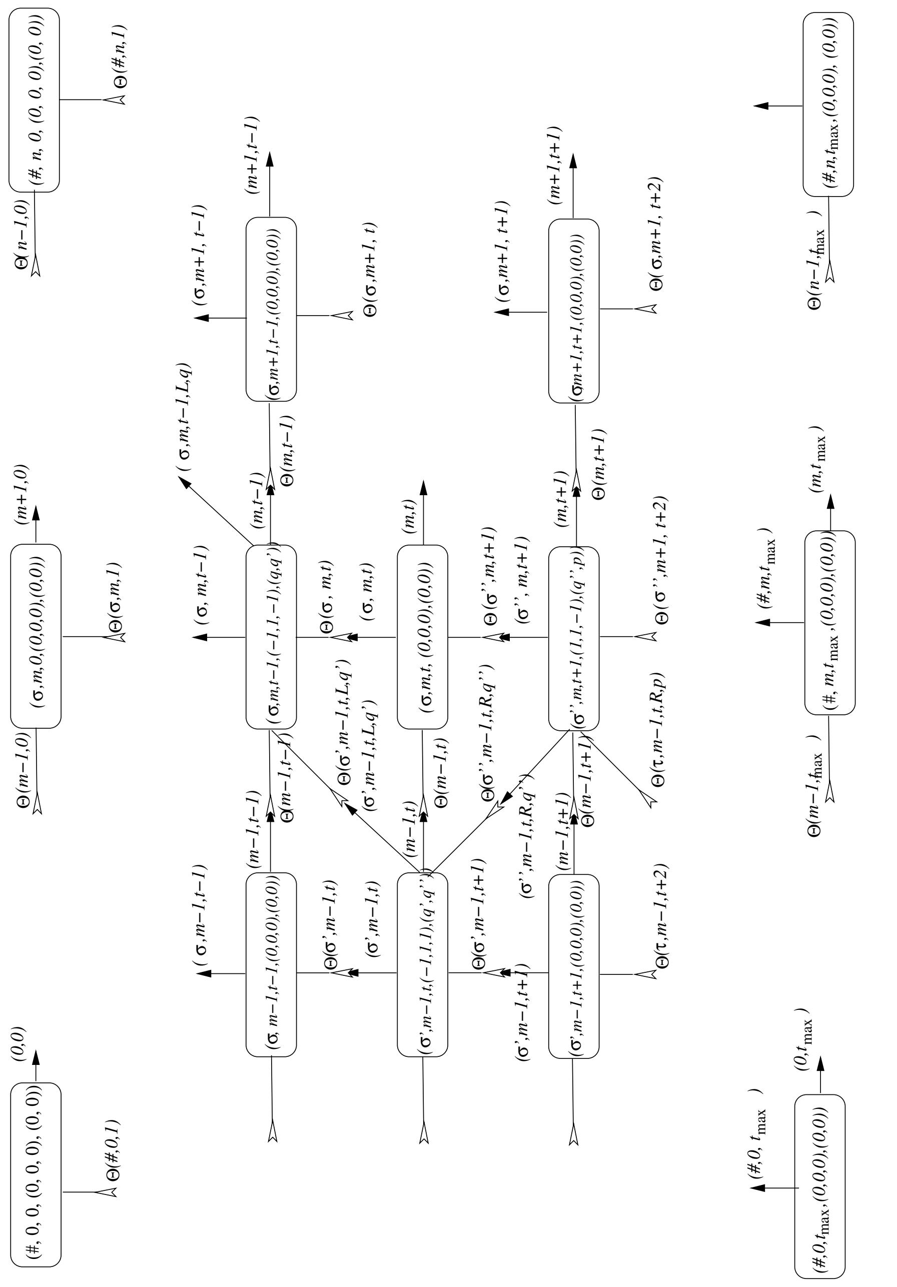


Fig. 5. Portions of assembly of tiles from P_M consistent with the Turing Machine M

5 Flexible-Tile Assembly Is Within NPTIME

We now go the other direction, proving that all weakly satisfiable problems within bounds are NPTIME-computable. We will use the Integer Programming Problem, called MP1 in [12], that was proved to be NP-computable by Borosh & Treybig [4] and NP-hard by Karp [20].

Definition 10. *This is the Integer Programming Problem (***MP1***).*

- INPUT:
  I1. A set finite $X \subseteq \mathbb{Z}^m \times \mathbb{Z}$ of $(m+1)$-tuples $(\boldsymbol{x}, b)$ of integers.
  I2. An m-tuple $\boldsymbol{c} \in \mathbb{Z}^m$.
  I3. An integer B.
- QUESTION: Does there exist an m-tuple $\boldsymbol{y} \in \mathbb{Z}^m$ such that the following is true?
  Q1. For each $(\boldsymbol{x}, b) \in X$, $\boldsymbol{x} \cdot \boldsymbol{y} \leq b$, where "$\cdot$" is vector dot or inner product.
  Q2. And $\boldsymbol{c} \cdot \boldsymbol{y} \geq B$.

Theorem 2. *All FTAP computable problems are NPTIME-computable.*

Idea of Proof. We prove that there is a PTIME reduction of pot types to MP1 problems that are solvable if and only if the original pot type was weakly satisfiable within bounds.

Given a pot type $\mathbf{P}$ over (H, θ), we will reduce the problem to whether there is a set of nonnegative integers (tile multiplicities) $\{y_{\mathbf{t}} \in \mathbb{N} \colon \mathbf{t} \in \mathbf{P}\}$, not all zero, giving us the number of each kind of tile in a complete complex. Considering each tile type as a vector with non-negative integer entries, we treat the pot $\mathbf{P}$ as a set of vectors of integers.

We need an input as in Definition 10 that is solvable iff there exists a system of integers $y_{\mathbf{t}}$ such that:

(i) Each $y_{\mathbf{t}}$ has $y_t \geq 0$, i.e., a number of tiles of a particular type.
(ii) For each $\mathbf{h}$, $\sum_{\mathbf{t} \in \mathbf{P}} y_{\mathbf{t}} \cdot \mathbf{t}(\mathbf{h}) = \sum_{\mathbf{t} \in \mathbf{P}} y_{\mathbf{t}} \cdot \mathbf{t}(\hat{\mathbf{h}})$, i.e., $\mathbf{P}$ can generate a complete complex.
(iii) $\sum_{\mathbf{t} \in \mathbf{P}} y_{\mathbf{t}} > 0$, i.e., the complete complex that is generated is non-trivial, contains at least one tile.
(iv) $\sum_{\mathbf{t} \in \mathbf{P}} y_{\mathbf{t}} \leq b$, i.e., the complete complex that is generated contains at most b tiles.

We rearrange the above inequalities to fit those that appear as input in Definition 10. ■

6 Final Remarks

This paper suggests at least two collections of problems that are yet to be considered.

First, while this model provides a method for computing NP problems in theory, it is not entirely clear how well it will work in practice. So far, laboratory experiments have determined that flexible tile computations would perform correctly for a 3-DNF propositional calculus formula of perhaps five clauses; it is not clear how well it would determine if a 5,000-clause formula is satisfiable (and a problem of this size is, in general, beyond our current computational capacities). The actual behavior of this model should be investigated further.

Second, this model suggests a set of logics describing computational complexity classes. The main theorem of this paper is a "representation theorem" similar to the main theorem of [9] (see [14] or [8], which presented a representation of NPTIME somewhat similar the one presented here. The combinatorial properties of this model, and related models, may provide additional tools for investigating some of the more obstinate classes in computational complexity theory.

Acknowledgement. This work has been supported in part by NSF Grants CCF #0432009 and EIA#0086015.

References

1. L. Adleman, *Molecular computation of solutions of combinatorial problems*, Science **266** (1994) 1021-1024.
2. L.M. Adleman, Q. Cheng, A. Goel, M-D. Huang, D. Kempe, P. Moisset de Espanés, P.W.K. Rothemund. *Combinatorial optimization problems in self-assembly, STOC'02 Proceedings*, Montreal Quebec, Canada, 2002.
3. L.M. Adleman, J. Kari, L. Kari, D. Reishus, *On the decidability of self-assembly of infinite ribbons*, *Proceedings of FOCS 2002*, IEEE Symposium on Foundations of Computer Science, Washington (2002) 530-537.
4. I. Borosh & L. B. Treybig, *Bounds on the positive integral solutions of linear Diophantine equations*, *Proc. Amer. Math. Soc.* **55** (1976) 299-304.
5. R. S. Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund, L. Adleman, *Solution of a 20-variable 3-SAT problem on a DNA Computer*, Science **296** (2002) 499-502.
6. J.H. Chen, N.C. Seeman, *Synthesis from DNA of a molecule with the connectivity of a cube, Nature* **350** (1991) 631-633.
7. S. Cook, *The complexity of theorem proving procedures*, *Proc. 3rd. ACM Ann. Symp. Theory of Comput.* (1971) 151-158.
8. H.-D. Ebbinghaus, J. Flum, *Finite Model Theory* (Springer, 2nd. ed., 1999).
9. R. Fagin, *Contributions to the Model Theory of Finite Structures*, UC Berkeley Ph.D. Thesis, 1973.
10. D. Faulhammer, A.R. Curkas, R.J. Lipton, L.F. Landweber, *Molecular computation: RNA solution to chess problems*, *PNAS* **97** (2000) 1385-1389.
11. T.J. Fu, N.C. Seeman, *DNA double crossover structures*, *Biochemistry* **32** (1993) 3211-3220.
12. M. R. Garey, D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman, 1979.
13. T. Head et.al, *Computing with DNA by operating on plasmids*, *BioSystems* **57** (2000) 87-93.
14. N. Immerman, *Descriptive Complexity Theory*, (Springer-Verlag, 1999).

15. N. Jonoska, P. Sa-Ardyen, N.C. Seeman, *Computation by self-assembly of DNA graphs*, *Genetic Programming and Evolvable Machines* **4** (2003) 123-137.
16. N. Jonoska, S. Karl, M. Saito, *Three dimensional DNA structures in computing, BioSystems* **52** (1999) 143-153.
17. N. Jonoska, S. Karl, M. Saito, *Creating 3-dimensional graph structures with DNA* in: *DNA based computers III* (H. Rubin, D. Wood, eds.), AMS DIMACS series **vol 48** (1999) 123-136.
18. N. Jonoska, G. McColm, A. Staninska, *Expectation and Variance of Self-Assembled Graph Structures*, *Proceedings of 11th International Meeting on DNA Computing*, London, Ontario, Canada, June 6-9 (2005) 49-58.
19. N. Jonoska, G. McColm, *On a Model of Computation by Self-Assembly*, in preparation.
20. R.M. Karp, *Reducibility among combinatorial problems*, in: *Complexity of Computer Computations* (R. E. Miller, W. E. Thatcher, eds.), Plenum Press (1972) 85-103.
21. M-Y. Kao, V. Ramachandran, *DNA Self-Assembly For Constructing 3D Boxes.* in: *Algorithms and Computation: ISAAC 2001 Proceedings*, Springer LNCS **2223** (2001) 429-440.
22. T.H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J.H. Reif, N.C. Seeman, *J. Am. Chem. Soc.* **122** (2000) 1848-1860.
23. J.M. Lehn, *Sopramolecular Chemistry*, *Science* **260** (1993) 1762-1763.
24. J.M. Lehn, *Toward complex matter: Supramolecular chemistry and self-organization*, *Proceedings of the National Academy of Science USA* **99** No.8 (2002) 4763-4768.
25. C. Mao, T.H. LaBean, J.H. Reif, N.C. Seeman, *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules*, *Nature* **407** (2000) 493-496.
26. C. Mao, W. Sun, Z. Shen, N.C. Seeman, *A nanomechanical device based on the B-Z transition of DNA*, *Nature* **397** (1999) 144-146.
27. J. H. Reif, S. Sahu, P. Yin, *Complexity of Graph Self-Assembly in Accretive Systems and Self-Destructive Systems*, *Proceedings of 11th International Meeting on DNA Computing*, London, Ontario, Canada, June 6-9 (2005) 101-112.
28. P.W.K. Rothemund, E. Winfree, *The Program-Size Complexity of Self-Assembled Squares*, *Proceedings of 33rd ACM meeting STOC 2001*, Portland, Oregon, May 21-23 (2001) 459-468.
29. P. Rothemund, N. Papadakis, E. Winfree, *Algorithmic Self-assembly of DNA Sierpinski Triangles*, PLoS Biology **2** (12) e424, 2004, (13 pages)
30. P. Sa-Ardyen, N. Jonoska, N. Seeman, *Self-assembly of graphs represented by DNA helix axis topology J. Am. Chem. Soc.* **126**(21) (2004) 6648-6657.
31. N.C. Seeman, *DNA junctions and lattices*, *J. Theor. Biol.* **99** (1982) 237-247.
32. N.C. Seeman, *DNA nicks and nodes and nanotechnology*, *NanoLetters* **1** (2001) 22-26.
33. W.M. Shihn J.D. Quispe, G.F. Joyce, *A 1.7-kilobase single-stranded DNA folds into a nano-scale octahedron*, *Nature* **427**, Feb. 12 (2004) 618-621.
34. D. Soloveichik, E. Winfree, *Complexity of Self-Assembled Shapes*, preprint at http://arxiv.org/abs/cs.CC/0412096.
35. Y. Wang, J.E. Mueller, B. Kemper, N.C. Seeman, *The assembly and characterization of 5-arm and 6-arm DNA junctions*, *Biochemistry* **30** (1991) 5667-5674.
36. E. Winfree, X. Yang, N.C. Seeman, *Universal computation via self-assembly of DNA: some theory and experiments*, in: *DNA based computers II* (L. Landweber, E. Baum eds.), AMS DIMACS series **44** (1998) 191-214.

37. E. Winfree, F. Liu, L.A. Wenzler, N.C. Seeman, *Design and self-assembly of two-dimensional DNA crystals*, Nature **394** (1998) 539-544.
38. H. Yan, X. Zhang, Z. Shen and N.C. Seeman, *A robust DNA mechanical device controlled by hybridization topology*, Nature **415** (2002) 62-65.
39. B. Yurke, A.J. Turberfield, A.P. Mills, F.C. Simmel Jr., *A DNA fueled molecular machine made of DNA*, Nature **406** (2000) 605-608.
40. Y. Zhang, N.C. Seeman, *The construction of a DNA truncated octahedron*, *J. Am. Chem. Soc.* **160** (1994) 1661-1669.

On Formulations of Firing Squad Synchronization Problems

Kojiro Kobayashi[1] and Darin Goldstein[2]

[1] Department of Information Systems Science, Faculty of Engineering, Soka University, 1-236 Tangi-cho, Hachioji-shi, Tokyo 192-8577, Japan
kobayasi@t.soka.ac.jp

[2] Department of Computer Engineering and Computer Science, California State University, Long Beach
daring@cecs.csulb.edu

Abstract. We propose a novel formulation of the firing squad synchronization problem. In this formulation we may use more than one general state and the general state to be used is determined by the boundary condition of the general. We show that the usual formulation and the new formulation yield different minimum firing times for some variations of the problem. Our results suggest that the new formulation is more suited for the general theory of the firing squad synchronization problem.

1 Introduction

The firing squad synchronization problem, or FSSP for short, is the following problem raised by J. Myhill in 1957 ([14]). Consider a finite automata A that has two input terminals, one from the left and the other from the right, and two output terminals, one to the left and the other to the right. The value of each output terminal at time t is the state of A at that time t. The state of A at a time $t+1$ is completely determined by the state and the values of the input terminals of A at time t. The set of the states of A includes at least three different states G, Q, F, called the *general state*, the *quiescent state*, and the *firing state*, respectively. For a number n (≥ 1) let N_n be the one-dimensional array of n nodes $p_1, p_2, \ldots, p_n$. Each node p_i is a copy of the automaton A, and the input terminals and the output terminals of adjacent nodes p_i, p_{i+1} are connected mutually ($1 \leq i < n$). See Figure 1. The values of the input terminal from the left of the leftmost node p_1 and the input terminal from the right of the rightmost node p_n are the special symbol # that indicates that the input terminal is open. The transition function of A must satisfy the following condition: if the state of A is Q and the value of each of the input terminals is either Q or # at a time t, then the state of A at the next time $t+1$ must be Q. We call the leftmost node p_1 the *root* of N_n. At time 0, the state of a node p_i is the general state G if the node is the root ($i = 1$) and the quiescent state Q otherwise ($i \geq 2$). Then, for each t (≥ 0) and i ($1 \leq i \leq n$), the state of the node p_i at time t is uniquely determined. The problem is to design a finite automaton A, a *solution* of FSSP, such that, for any n, all the nodes of N_n enter the firing state F simultaneously for the first time.

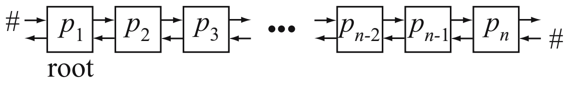


Fig. 1. The original FSSP

We can easily construct a solution having the firing time $3n$ for N_n. Moreover, we can easily show that the firing time of any solution for N_n cannot be smaller than $2n-2$. Hence, if a solution has the firing time $2n-2$ for all N_n, we may call it a *minimal-time solution.* Existence of minimal-time solutions was first shown by Goto ([7]), and later by Waksman ([20]).

After this original FSSP was introduced, many variations of FSSP have been proposed and studied ([13]). Suppose that a variation V of FSSP has a solution. For each problem instance N of V, by the *minimum firing time* of N of the variation V we mean the minimum of the firing times of solutions A of V for N, where A ranges over all solutions of V. If the firing time of a solution $\tilde{A}$ of V for N is the minimum firing time of N of V for all problem instances N of V, we call $\tilde{A}$ a *minimal-time solution* of the variation V.

In this paper we propose a modification of the formulation of FSSP and study how the modification influences the minimum firing times of various variations of FSSP. The modification is as follows. First, instead of having one unique general state G, we allow a finite automaton to have more that one general state $\mathrm{G}_1, \mathrm{G}_2, \ldots, \mathrm{G}_s$. The general state to be used is uniquely determined by the boundary condition of the root. Here, by the *boundary condition* of a node of a problem instance, we mean the information of which input terminals and output terminals of the node are open. Second, instead of having one unique firing state F, we specify a set $\mathcal{F}$ of states as the set of firing states. For a finite automaton to be a solution, all nodes of the network must enter some firing state simultaneously for the first time. Different nodes may enter different firing states. A general state G_i may be also a firing state.

This modification implies the following for designing solutions of FSSP. First, the root can send its boundary condition to adjacent nodes at time 0. Hence a node adjacent to the root can use the boundary condition of the root in determining its state at time 1. Second, the general state that should be used when the boundary condition of the root is "all terminals are open" may be a firing state. Hence, if the problem instance has only one node, the root can fire at time 0 and hence the firing time can be be 0.

We call the usual formulation and the new formulation of FSSP the *traditional model* and the *boundary sensitive model*, respectively. There are two motivations for using the boundary sensitive model.

The main motivation is that the boundary sensitive model simplifies the analysis of minimum firing time and allows us to understand the essential structures of minimal-time solutions. We explain why the boundary sensitive model is more suitable through examples.

Another reason concerns the recent change of the motivation for studying FSSP. Recently, the FSSP for directed networks has been utilized as one of the basic protocols for designing network algorithms (for example, [6]). In such applications, a node is a circuit or a computer in a network, and connections between nodes are network connections. In this case, the time for a node to check its boundary condition is negligibly smaller than the time for information exchange between nodes. Hence it is natural to assume that the root (the initiator of the protocol) can send its boundary condition at time 0. The "firing" of the network generally means that the network simultaneously takes some action. If the network has only one node the root can know this at time 0 and start the action promptly. Hence it is natural to assume that if the network has only one node the firing time is 0.

For a variation V of FSSP and a problem instance N of V, let $\mathrm{mft}_{V,\mathrm{tr}}(N)$ and $\mathrm{mft}_{V,\mathrm{bs}}(N)$ denote the minimum firing times of V for N of the traditional model and the boundary sensitive model, respectively. We are interested in the relation between $\mathrm{mft}_{V,\mathrm{tr}}(N)$ and $\mathrm{mft}_{V,\mathrm{bs}}(N)$. We always have $\mathrm{mft}_{V,\mathrm{tr}}(N) > \mathrm{mft}_{V,\mathrm{bs}}(N)$ for the problem instance N that has only one node because the first value is at least 1 and the second value is 0. Hence, in the remainder of the paper we consider only problem instances that have at least two nodes.

The main technical results of the paper are twofold. First we show that for many variations the two models give the same minimum firing time. Second we show that for some variations the two models give different minimum firing times and in the traditional model the determination of the minimum firing time is unnecessarily complicated due to unnaturalness of the model.

We should mention that the formulation of FSSP that uses more than one firing state has been used by Imai and Morita ([8]) to study FSSP by reversible cellular automata.

2 FSSP That Have Known Minimal-Time Solutions

In this section we consider variations of FSSP for which we know minimal-time solutions. The following lists some of these variations:

- The original FSSP of the one-dimensional line of length n (Fig. 1): The minimum firing time is $2n-2$ ([7], [20], [1]);
- The one-dimensional line of length n such that the root may be at any position: The minimum firing time is $2n-2-\min\{p-1, n-p\}$, where p is the position of the root ($1 \le p \le n$) ([15]);
- The one-dimensional line of length n with k roots such that the roots may be at any positions: The minimum firing time is $2n-2-\min\{\max_i(p_i-1), \max_i(n-p_i)\}$, where p_i is the position of the ith root ($1 \le p_i \le n$, $1 \le i \le k$) ([18]);
- The square of size $n \times n$: The minimum firing time is $2n-2$ ([19]);
- The rectangle of size $m \times n$: The minimum firing time is $m+n+\max\{m,n\}-3$ ([19]);
- The cube of size $n \times n \times n$: The minimum firing time is $3n-3$ ([19]);

- The ring of size n: The minimum firing time is n ([3], [2]);
- The ring of size n with one-way information flow: The minimum firing time is $2n - 1$ ([10], [12]).

For all of these variations V we can show $\mathrm{mft}_{V,\mathrm{tr}}(N) = \mathrm{mft}_{V,\mathrm{bs}}(N)$ for any N. The proofs are essentially the same. As an example, suppose that V is the original FSSP. For this V there exists a solution for the traditional model that shows $\mathrm{mft}_{V,\mathrm{tr}}(N_n) \le 2n - 2$. Moreover, we can prove that the firing time of any solution A of V for N_n cannot be smaller than $2n-2$ by formalizing the intuitive reasoning that it takes at least $2n - 2$ time for the root to know the position of the rightmost node. But this proof is also true for the boundary sensitive model. Hence we have $2n - 2 \le \mathrm{mft}_{V,\mathrm{bs}}(N_n) \le \mathrm{mft}_{V,\mathrm{tr}}(N_n) \le 2n - 2$.

3 FSSP of General Networks

Next we consider variations of FSSP for general networks. Of these variations two are the most basic. One is the FSSP of directed networks and the other is the FSSP of bilateral networks. We abbreviate these two variations to DN and BN respectively.

In DN, an automaton A has a input terminals and b output terminals, where a, b (≥ 1) are implicit parameters. A problem instance N of DN is a network that is obtained from copies of A by connecting some of the outputs to some of the inputs. Each output of a node is either open or is connected to a single input of another node, and hence the "fan-out" is at most one. Each automaton A knows whether its jth output is open or not for each j ($1 \le j \le b$). One node is specified as the root. Moreover, the network N must be strongly connected, that is, there must be a directed path of connections from v to v' for each pair (v, v') of nodes.

A network N of DN is a *bilateral* network if $a = b$ and the following condition is satisfied: if the ith output of a node v is connected to the jth input of a node v' then the jth output of v' is connected to the ith input of v. BN is the the variation such that the problem instances are all bilateral networks. Note that all the variations mentioned in Section 2 are subproblems of BN except the last one, the FSSP of rings with one-way information flow.

For both DN and BN we do not know minimal-time solutions. The best known solution of BN is by Nishitani and Honda ([16]) and its firing time is $3r - 1$, where r is the radius of the network. A solution of DN was first found by Kobayashi ([9]). Its firing time was an exponential function of the number n of nodes. The firing time has been improved to $O(n^2)$ by Even, Litman and Winkler ([4]) and then to $O(nd)$ by Ostrovsky and Wilkerson ([17]), where d is the diameter of the network.

Claim 1. *For both of* BN *and* DN *the two formulations have the same minimum firing time.*

Proof. Let $N = (V, E)$ be a directed network or a bilateral network, where V is the set of nodes and E is the set of connections. For $v, v' \in V$ and $e \in E$, let

$d(v, v')$ and $d_e(v, v')$ respectively denote the length of a shortest path from v to v' (or between v and v' if N is bilateral) and the length of a shortest path from v to v' (or between v and v' if N is bilateral) that passes through e, respectively. Moreover let $f(N)$ denote the value $\max_{e \in E, v \in V} d_e(v_{\mathrm{g}}, v)$, where v_{g} denotes the root. Then we have

$$\mathrm{mft}_{\mathrm{DN,tr}}(N) = \mathrm{mft}_{\mathrm{DN,bs}}(N) = f(N),$$
$$\mathrm{mft}_{\mathrm{BN,tr}}(N) = \mathrm{mft}_{\mathrm{BN,bs}}(N) = f(N).$$

We will very briefly explain the idea for proving these characterizations of minimum firing time only for DN.

First we show $\mathrm{mft}_{\mathrm{DN,bs}} \geq f(N)$ using the network N shown in Fig. 2 as an example. For this network N we have $f(N) = 6$ and the e, v that realize this maximum value 6 is $e = (p_3, 1, 1, p_4), v = p_3$, where the symbol (v, i, j, v') denotes the connection from the ith output of v to the jth input of v'. Let N' be the network shown in Fig. 2 and let $\tilde{t}$ be any time such that $\tilde{t} \leq 5$. Then, at time $\tilde{t}$, the states of p_3 in N and p_3 in N' are the same and the the state of p_9 in N' is the quiescent state Q. Hence, if a solution A of DN of the boundary sensitive model fires at $\tilde{t}$ on N, at that time the states of nodes in N' contain both of a firing state and Q. This contradicts our assumption that A is a solution of DN. Hence the firing time of A cannot be $\tilde{t}$. This proves $\mathrm{mft}_{\mathrm{DN,bs}}(N) \geq 6 = f(N)$.

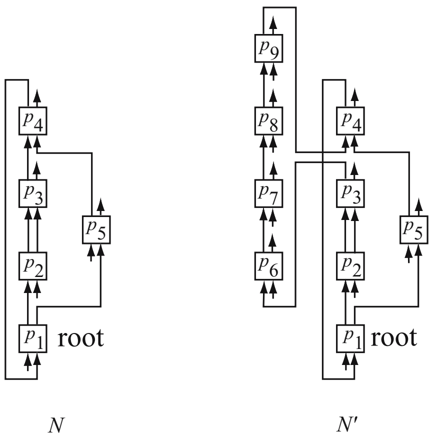


Fig. 2. FSSP of directed networks

Next we show $\mathrm{mft}_{\mathrm{DN,bs}}(N) \leq f(N)$. We select one directed network N and fix it. We show how to construct a solution A_{bs} of DN of the boundary sensitive model whose firing time for N is at most $f(N)$. The structure of A_{bs} essentially depends on the fixed network N.

A_{bs} simulates two finite automata $A_{1,\mathrm{bs}}$, $A_{2,\mathrm{bs}}$ of the boundary sensitive model and fires when at least one of them fires. $A_{1,\mathrm{bs}}$ may be any solution of

DN. $A_{2,\mathrm{bs}}$ is a finite automaton such that all the nodes collaborate to check that the given network is N. If the given network is N then all the nodes know it before or at $f(N)$, and fire at $f(N)$. Otherwise each node never fires. Hence A_{bs} is a solution.

The details of the behavior of $A_{2,\mathrm{bs}}$ are as follows. For each node $v \in V$ we fix one shortest path from v_{g} to v and use that path to uniquely specify v. For example, if we select the path $(p_1, 1, 1, p_2)$, $(p_2, 2, 2, p_3)$ for p_3 in the network N shown in Fig. 2, all the nodes refer to p_3 of N as "the node that is arrived at when we proceed from v_{g} along connections $(p_1, 1, 1, p_2)$, $(p_2, 2, 2, p_3)$."

For each pair (v', v) of nodes of N, $A_{2,\mathrm{bs}}$ uses a signal to teach v that v' really exists in the network, and also the boundary condition of v'. The time needed for this is $d(v_{\mathrm{g}}, v') + d(v', v)$ because we are using the boundary sensitive model. Moreover, for each pair (e, v) of $e \in E$ and $v \in V$, $A_{2,\mathrm{bs}}$ uses a signal to teach v that e really exists in the network. The time needed for this is $d(v_{\mathrm{g}}, v') + 1 + d(v'', v) = d_e(v_{\mathrm{g}}, v)$, where v', v'' are nodes such that e is from v' to v''.

Hence, if the given network is N, using these signals all the nodes know this before or at time

$$\max\{\max_{v',v\in V}(d(v_{\mathrm{g}}, v') + d(v', v)), \max_{e\in E, v\in V} d_e(v_{\mathrm{g}}, v)\} = \max_{e\in E, v\in V} d_e(v_{\mathrm{g}}, v) = f(N)$$

and $A_{2,\mathrm{bs}}$ at any node can fire at time $f(N)$. If the network is not N, $A_{2,\mathrm{bs}}$ at any node never fires. Hence, the firing time of the solution A_{bs} for N is at most $f(N)$, and hence $\mathrm{mft}_{\mathrm{DN,bs}}(N) \le f(N)$.

The above idea cannot be used directly for the traditional model because in the model the time for v to know the boundary condition of v' is not $d(v_{\mathrm{g}}, v') + d(v', v)$ for $v' = v_{\mathrm{g}}$. This complicates the analysis of $\mathrm{mft}_{\mathrm{DN,tr}}(N)$. However, if we note that $\max_{v\in V} d(v_{\mathrm{g}}, v) + 1 \le f(N)$, we can construct a solution A_{tr} of DN of the traditional model whose firing time for N is at most $f(N)$.

A_{tr} is obtained from A_{bs} by modifying its components $A_{1,\mathrm{bs}}$ and $A_{2,\mathrm{bs}}$ to automata $A_{1,\mathrm{tr}}$ and $A_{2,\mathrm{tr}}$ of the traditional model as follows. $A_{1,\mathrm{tr}}$ may be any solution of DN of the traditional model. There are 2^{a+b} boundary conditions. $A_{2,\mathrm{tr}}$ simulates the behaviors of $A_{2,\mathrm{bs}}$ for all of these boundary conditions simultaneously. At the same time, at time 1 the root broadcasts the correct boundary condition to all nodes. A node fires when it has received the correct boundary condition and the simulated $A_{2,\mathrm{bs}}$ for that boundary condition fires. The condition $\max_{v\in V} d(v_{\mathrm{g}}, v) + 1 \le f(N)$ guarantees that each node knows the correct boundary condition before or at $f(N)$. Hence $A_{2,\mathrm{tr}}$ fires at $f(N)$ if the given network is N.

Hence the firing time of A_{tr} for N is at most $f(N)$, and hence $\mathrm{mft}_{\mathrm{DN,tr}}(N) \le f(N)$. This, together with $f(N) \le \mathrm{mft}_{\mathrm{DN,bs}}(N) \le \mathrm{mft}_{\mathrm{DN,tr}}(N)$, shows $\mathrm{mft}_{\mathrm{DN,tr}}(N) = f(N)$.

The proof of $\mathrm{mft}_{\mathrm{BN,tr}}(N) = \mathrm{mft}_{\mathrm{BN,bs}}(N) = f(N)$ is similar. □

4 FSSP of Paths and Regions in $\mathbb{Z}^2$ and $\mathbb{Z}^3$

The final variations we consider are paths and regions in the two-dimensional grid space $\mathbb{Z}^2$ and the three-dimensional grid space $\mathbb{Z}^3$. First we explain the variations for $\mathbb{Z}^2$.

We say that two points $p = (x, y)$, $p' = (x', y')$ in $\mathbb{Z}^2$ are *adjacent* if either $x = x'$ and $|y - y'| = 1$ or $|x - x'| = 1$ and $y = y'$. By a *path* in $\mathbb{Z}^2$, or simply a *path*, we mean a sequence $p_1 p_2 \dots p_n$ of points in $\mathbb{Z}^2$ $(n \geq 1)$ such that p_i and p_j are adjacent if and only if $|i - j| = 1$ $(1 \leq i \leq n,\ 1 \leq j \leq n)$.

The FSSP of paths in $\mathbb{Z}^2$, or 2PATH for short, is the FSSP such that problem instances are paths $p_1 p_2 \dots p_n$ in $\mathbb{Z}^2$ and p_1 is the root of each path $p_1 p_2 \dots p_n$. Another variation, the FSSP of generalized paths in $\mathbb{Z}^2$, or g-2PATH for short, is the FSSP such that problem instances are paths $p_1 p_2 \dots p_n$ in $\mathbb{Z}^2$ and the root may be at any position. Finally, the FSSP of regions in $\mathbb{Z}^2$, or 2REG for short, is the FSSP such that problem instances are nonempty finite subsets X of $\mathbb{Z}^2$ such that any two points p, p' in X are connected with a path in X, and the root may be any point in X.

We can define similar variations 3PATH, g-3PATH, 3REG for $\mathbb{Z}^3$. In Fig. 3 (a) and (b) we show examples of problem instances of 2PATH and 2REG, respectively.

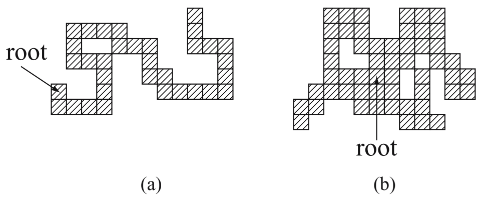


Fig. 3. Examples of 2PATH and 2REG

For each of these three variations, a finite automata A that is used to construct a solution has four inputs and four outputs, each corresponding to the direction of one of the four adjacent positions. Two copies of A at adjacent points p, p' are mutually connected with the corresponding input and output. Hence, all of these variations are subproblems of BN.

At present we know no minimal-time solutions for these six variations. However, in [5] we showed that if P $\neq$ NP then 3PATH, g-3PATH and 3REG have no minimal-time solutions. Hence these three variations are highly unlikely to have minimal-time solutions. (In [5] we showed the result only for 3PATH. However the proof also applies to g-3PATH and 3REG with slight modifications.)

It is evident that $\mathrm{mft}_{\mathrm{2PATH,tr}}(N) = \mathrm{mft}_{\mathrm{2PATH,bs}}(N)$ and $\mathrm{mft}_{\mathrm{3PATH,tr}}(N) = \mathrm{mft}_{\mathrm{3PATH,bs}}(N)$ for any path $N = p_1 p_2 \dots p_n$ in $\mathbb{Z}^2$ and $\mathbb{Z}^3$. Hence we are interested in how $\mathrm{mft}_{V,\mathrm{tr}}(N)$ and $\mathrm{mft}_{V,\mathrm{bs}}(N)$ are related for V = g-2PATH, 2REG, g-3PATH, 3REG. We consider the problem only for g-2PATH. All the results hold true for g-3PATH without any modification and for 2REG and 3REG with slight modifications.

In [11], a characterization of $\mathrm{mft}_{\mathrm{2PATH,tr}}(p_1p_2\ldots p_n)$ was obtained. We elaborate on this result in detail below.

For $1 \le i < n$, let $e(p_1p_2\ldots p_n, i)$ denote the length m of a longest extension of $p_1p_2\ldots p_i$ of the form $p_1p_2\ldots p_ip_{i+1}q_2\ldots q_m$. The value $e(p_1p_2\ldots p_n, i)$ may be ∞. For $i = n$, we define $e(p_1p_2\ldots p_n, n)$ to be 0. Let i_0 be the value defined by $i_0 = \min\{i | 1 \le i \le n, i \ge e(p_1p_2\ldots p_n, i)\}$. Let $f(p_1p_2\ldots p_n)$ be $2i_0 - 1$ if $i_0 = e(p_1p_2\ldots p_n, i_0)$ and $2i_0 - 2$ if $i_0 > e(p_1p_2\ldots p_n, i_0)$.

Lemma 1 ([11]).

$$\mathrm{mft}_{\mathrm{2PATH,tr}}(p_1p_2\ldots p_n) = \mathrm{mft}_{\mathrm{2PATH,bs}}(p_1p_2\ldots p_n) = f(p_1p_2\ldots p_n).$$

Proof. Only an outline of the proof is given. As we have already mentioned, we can easily show $\mathrm{mft}_{\mathrm{2PATH,tr}}(p_1p_2\ldots p_n) = \mathrm{mft}_{\mathrm{2PATH,bs}}(p_1p_2\ldots p_n)$. Hence we only show $\mathrm{mft}_{\mathrm{2PATH,bs}}(p_1p_2\ldots p_n) = f(p_1p_2\ldots p_n)$. We assume that $i_0 < n$. The proof for the case $i_0 = n$ is simpler.

First we show that the firing time of any solution A for $\alpha = p_1p_2\ldots p_n$ cannot be smaller than $f(p_1p_2\ldots p_n)$. Let $\tilde{t}$ be a time such that $\tilde{t} < f(p_1p_2\ldots p_n)$.

Suppose that $i_0 = e(p_1p_2\ldots p_n, i_0)$. Then $\tilde{t} < f(p_1p_2\ldots p_n) = 2i_0 - 1$. There is a path of the form $\alpha' = p_1p_2\ldots p_{i_0}p_{i_0+1}q_2\ldots q_{i_0}$. At time $\tilde{t}$, the state of p_1 in α and the state of p_1 in α' are the same and the state of q_{i_0} in α' is Q. Hence A cannot fire on α at time $\tilde{t}$.

Suppose that $i_0 > e(p_1p_2\ldots p_n, i_0)$. Then $0 \le \tilde{t} < f(p_1p_2\ldots p_n) = 2i_0 - 2$ and hence $2 \le i_0$. We have $i_0 \le e(p_1p_2\ldots p_n, i_0 - 1)$. Hence there is a path of the form $\alpha' = p_1p_2\ldots p_{i_0-1}p_{i_0}q_2\ldots q_{i_0}$. At time $\tilde{t}$, the state of p_1 in α and the state of p_1 in α' are the same and the state of q_{i_0} in α' is Q. Hence A cannot fire on α at time $\tilde{t}$.

Next we construct a solution A whose firing time for $p_1p_2\ldots p_n$ is at most $f(p_1p_2\ldots p_n)$. A simulates two finite automata A_1, A_2. The structure of A_2 essentially depends on the path $p_1p_2\ldots p_n$. A fires when at least one of A_1, A_2 fires. A_1 may be any solution of 2PATH. A_2 checks that the given path starts with $p_1p_2\ldots p_{i_0}p_{i_0+1}$. If the check succeeds, A_2 at any node fires at time $f(p_1p_2\ldots p_n)$. If the check fails, A_2 at any node never fires. Hence A is a solution.

The details of the behavior of A_2 is as follows. At time 0, A_2 sends a check signal from the root to the node p_{i_0} along the path $p_1p_2\ldots p_{i_0}$. If check succeeds, the check signal knows it at p_{i_0} at time $i_0 - 1$ and then the check signal broadcasts the order "fire at time $f(p_1p_2\ldots p_n)$" to all the nodes. If $i_0 = e(p_1p_2\ldots p_n, i_0)$ all the nodes receive the order before or at time $(i_0 - 1) + \max\{i_0 - 1, i_0\} = 2i_0 - 1 = f(p_1p_2\ldots p_n)$. If $i_0 > e(p_1p_2\ldots p_n, i_0)$ all the nodes receive the order before or at time $(i_0 - 1) + \max\{i_0 - 1, i_0 - 1\} = 2i_0 - 2 = f(p_1p_2\ldots p_n)$. Hence, in any case, if the check succeeds all the nodes receive the order "fire at time $f(p_1p_2\ldots p_n)$" before or at time $f(p_1p_2\ldots p_n)$ and hence can fire at that time.

Hence the firing time of A for $p_1p_2\ldots p_n$ is at most $f(p_1p_2\ldots p_n)$. $\square$

In Fig. 4 we show an example of a path. For this path $p_1p_2\ldots p_{22}$ we have $e(p_1p_2\ldots p_{22}, 18) = \infty$, $e(p_1p_2\ldots p_{22}, 19) = 4$, and hence $i_0 = 19$, $19 >$

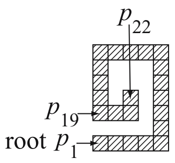


Fig. 4. An example of paths

$e(p_1p_2\dots p_{22}, 19)$, $f(p_1p_2\dots p_{22}) = 2{\cdot}19{-}2 = 36$. Hence $\mathrm{mft}_{\mathrm{2PATH,tr}}(p_1p_2\dots p_{22}) = \mathrm{mft}_{\mathrm{2PATH,bs}}(p_1p_2\dots p_{22}) = 36$.

A path $p_1p_2\dots p_n$ such that p_1 is the root is also a problem instance of g-2PATH. For this problem instance we have the following results.

Theorem 1. $\mathrm{mft}_{\mathrm{g-2PATH,bs}}(p_1p_2\dots p_n) = f(p_1p_2\dots p_n)$.

Proof. In the proof of Lemma 1 the check signal of A_2 checked that the given path starts with $p_1p_2\dots p_{i_0}p_{i_0+1}$. As a solution of g-2PATH, in addition to this the check signal should also check that the root is at the end. However, if we use the boundary sensitive model the check signal can check it without any additional time. Hence we can construct a solution A of g-2PATH of the boundary sensitive model whose firing time for $p_1p_2\dots p_n$ is at most $f(p_1p_2\dots p_n)$. □

Theorem 2. *If* $i_0 = e(p_1p_2\dots p_n, i_0)$ *and there is a path of the form*

$$r_{2i_0+1}\dots r_3r_2p_1p_2\dots p_{i_0}p_{i_0+1}q_2\dots q_{i_0},$$

then

$$\mathrm{mft}_{\mathrm{g-2PATH,tr}}(p_1p_2\dots p_n) = f(p_1p_2\dots p_n) + 1.$$

Proof. We have $\mathrm{mft}_{\mathrm{g-2PATH,tr}}(p_1p_2\dots p_n) \le \mathrm{mft}_{\mathrm{g-2PATH,bs}}(p_1p_2\dots p_n) + 1 = f(p_1p_2\dots p_n) + 1$.

Suppose that there is a solution A of g-2PATH of the traditional model whose firing time $\tilde{t}$ for $p_1p_2\dots p_n$ is at most $f(p_1p_2\dots p_n) = 2i_0 - 1$.

Suppose that we run A on the three paths $\alpha = p_1p_2\dots p_n$, $\alpha' = p_1p_2\dots p_{i_0} p_{i_0+1}q_2\dots q_{i_0}$, $\alpha'' = r_{2i_0+1}\dots r_3r_2p_1p_2\dots p_{i_0}p_{i_0+1}q_2\dots q_{i_0}$. In all of these paths p_1 is the root. Consider the states of nodes in these three paths at time $\tilde{t}$. All the nodes in α are F because A on α fires at $\tilde{t}$. The state of p_1 in α and the state of p_1 in α' are the same. Hence the state of p_1 in α' is F and hence the state of q_{i_0} in α' is also F. But the state of q_{i_0} in α' and the state of q_{i_0} in α'' are the same because A is a solution of the traditional model. Hence the state of q_{i_0} is also F. But the state of r_{2i_0+1} in α'' is Q. This is a contradiction. Hence we have $\mathrm{mft}_{\mathrm{g-2PATH,tr}}(p_1p_2\dots p_n) \ge f(p_1p_2\dots p_n) + 1$. □

In Fig. 5 we show an example of paths $\alpha_1 = p_1p_2\dots p_{108}$ that satisfies the condition of Theorem 2. For this path α_1 we have $e(\alpha_1, 106) = \infty$, $e(\alpha_1, 107) =$

107, $i_0 = 107$, and hence $f(\alpha_1) = 2i_0 - 1 = 213$. The equation $e(\alpha_1, 107) = 107$ was checked by the exhaustive search by computers. The path α_2 shown in Fig. 5 is one of the path of the form $\alpha_2 = p_1p_2 \dots p_{107}p_{108}q_2 \dots q_{107}$ found by the search. From this α_2 we can easily construct a path of the form $r_{215}r_{214} \dots r_3r_2p_1p_2 \dots p_{107}p_{108}q_2 \dots q_{107}$. Hence, by Theorem 2 we have $\mathrm{mft}_{\mathrm{g-2PATH,tr}}(\alpha_1) = f(\alpha_1) + 1 = 214$ while $\mathrm{mft}_{\mathrm{g-2PATH,bs}}(\alpha_1) = f(\alpha_1) = 213$.

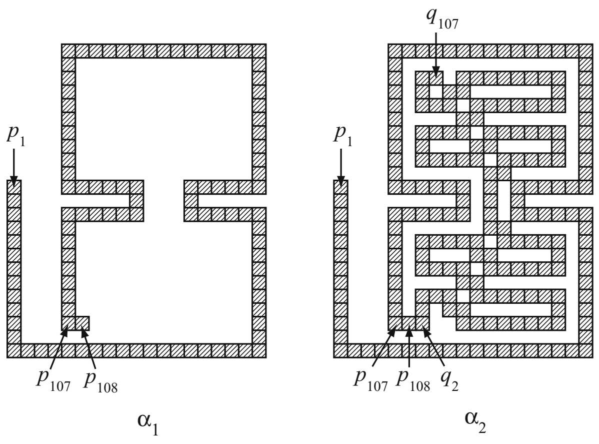


Fig. 5. Two paths α_1, α_2

The proofs of the following two theorems are not difficult and we omit them.

Theorem 3. *If* $i_0 - 1 = e(p_1p_2 \dots p_n, i_0)$ *and there is a path of the form*

$$r_{2i_0}r_{2i_0-1} \dots r_3r_2p_1p_2 \dots p_{i_0}p_{i_0+1}q_2 \dots q_{i_0-1},$$

then

$$\mathrm{mft}_{\mathrm{g-2PATH,tr}}(p_1p_2 \dots p_n) = f(p_1p_2 \dots p_n) + 1.$$

Theorem 4. *If* $i_0 - 2 \geq e(p_1p_2 \dots p_n, i_0)$ *then*

$$\mathrm{mft}_{\mathrm{g-2PATH,tr}}(p_1p_2 \dots p_n) = f(p_1p_2 \dots p_n).$$

From Theorems 2, 3 we are tempted to conjecture that if $i_0 - 1 \leq e(p_1p_2 \dots p_n, i_0)$ then $\mathrm{mft}_{\mathrm{g-2PATH,tr}}(p_1p_2 \dots p_n) = f(p_1p_2 \dots p_n) + 1$. However, this is not true. Suppose that we construct a path α_3 shown in Fig. 6 from the path α_1 shown in Fig. 5 by bending its beginning. For α_3, we have $i_0 = e(p_1p_2 \dots p_n, i_0)$ and $\mathrm{mft}_{\mathrm{g-2PATH,tr}}(p_1p_2 \dots p_n) = f(p_1p_2 \dots p_n)$. In α_3, the check signal of the traditional model that starts at p_1 at time 0 knows the boundary condition of p_1 as soon as it arrives at p_{12}, and hence it needs no extra time to check the boundary condition of p_1.

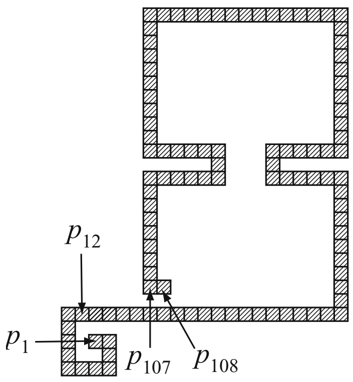


Fig. 6. A modified path α_3

5 Conclusions

The two models give the same minimum firing time for the variations considered in Sections 2, 3. However, they give different minimum firing time for g-2PATH (and 2REG, g-3PATH, 3REG). For these FSSP, the minimum firing time of the boundary sensitive model has a very simple characterization shown in Theorem 1 for paths with the roots at the end. However, Theorems 2, 3, 4 and the phenomenon mentioned in Fig. 6 show that the analysis of the minimum firing time of these FSSP of the traditional model is very complicated and moreover it is due to the unnaturalness of the model.

Hence, if our goal is to construct a general theory of minimum firing time of FSSP, our results suggest that the boundary sensitive model is “the” model to be used.

References

1. R. Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10:22-42, 1967.
2. A. Berthiaume, T. Bittner, L. Perković, A. Settle, and J. Simon. Bounding the firing synchronization problem on a ring. *Theoretical Computer Science*, 320:213-228, 2004.
3. K. Culik. Variations of the firing squad problem and applications. *Information Processing Letters*, 30:153-157, 1989.
4. S. Even, A. Litman, and P. Winkler. Computing with snakes in directed networks of automata. *J. Algorithms*, 24(1):158-170, 1997.
5. D. Goldstein and K. Kobayashi. On the complexity of network synchronization. In *Proc. of 15th International Symposium, ISAAC 2004*, Lecture Notes in Computer Science, vol. 3341, 496-507, Dec. 20-22, 2004.
6. D. Goldstein and N. Meyer. The wake up and report problem is asymptotically time-equivalent to the firing squad synchronization problem. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 578-587, San Francisco, CA, 6-8 January 2002.

7. E. Goto. A minimal time solution of the firing squad problem. *Course Notes for Applied Mathematics 298, Harvard University*, pages 52-59, 1962.
8. K. Imai and K. Morita. Firing squad synchronization problem in reversible cellular automata. *Theoretical Computer Science*, 165:475-482, 1996.
9. K. Kobayashi. The firing squad synchronization problem for a class of polyautomata networks. *J. Comput. Systems Sci.*, 17(3):300-318, 1978.
10. K. Kobayashi. A minimal time solution to the firing squad synchronization problem of rings with one-way information flow. *Tokyo Institute of Technology, Department of Information Sciences, Research Reports on Information Sciences*, No. C-8, September 1976.
11. K. Kobayashi. On time optimal solutions of the firing squad synchronization problem for two-dimensional paths. *Theoretical Computer Science*, 259:129-143, 29 May 2001.
12. S. LaTorre, M. Napoli, and M. Parente. Synchronization of one-way connected processors. *Complex Systems*, 10:239-255, 1996.
13. J. Mazoyer. An overview of the firing squad synchronization problem. In *Automata Networks* (C. Choffrut, ed.), Lecture Notes in Computer Science, vol. 316, 82-94, 1986.
14. E. F. Moore. *Sequential Machines, Selected Papers*, Addison Wesley, Reading, MA, 1962.
15. F. R. Moore and G. G. Langdon. A generalized firing squad problem, *Information and Control*, 12:212-220, 1968.
16. Y. Nishitani and N. Honda. The firing squad synchronization problem for graphs. *Theoretical Computer Science*, 14(1):39-61, 1981.
17. R. Ostrovsky and D. Wilkerson. Faster computation on directed networks of automata. In Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing, pages 38-46, Ottawa, Ontario, Canada, 2-23 August 1995.
18. H. Schmid and T. Worsch. The firing squad synchronization problem with many generals for one-dimensional CA. 3rd IFIP International Conference on Theoretical Computer Science, 111-124, 2004.
19. I. Shinahr. Two- and three-dimensional firing-squad synchronization problem, *Information and Control*, 24:163-180, 1974.
20. A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9:66-78, 1966.

Computation in One-Dimensional Piecewise Maps and Planar Pseudo-Billiard Systems

Oleksiy Kurganskyy[1] and Igor Potapov[2,*]

[1] Institute of Applied Mathematics and Mechanics,
Ukrainian National Academy of Sciences,
74 R. Luxemburg St, Donetsk, Ukraine
kurgansk@gmx.de

[2] Department of Computer Science,
University of Liverpool, Chadwick Building,
Peach St, Liverpool L69 7ZF, U.K.
igor@csc.liv.ac.uk

Abstract. The computation in low-dimensional system is related to many long standing open problems. In this paper we show the universality of a one-dimensional iterative map defined by elementary functions. The computation in iterative maps have a number of connections with other unconventional models of computations. In particular, one-dimensional iterative maps can be simulated by a planar pseudo-billiard system. As a consequence of our main result we show that a planar pseudo-billiard system is not only can demonstrate a chaotic behaviour, but also has ability of universal computation.

1 Introduction

Most of the real dynamical systems, such as the model systems in physics, chemistry, biology, ecology, economics etc, are described by equations, where the system evolves through a set of discrete time steps. The simplest models of these equations are the iterative maps because the future value of some variable at time $n+1$ depends only on its value at the present time n (here n is an integer). An iterative map is of the form $x_{n+1} = f(x_n)$ where $f(x)$ is called the mapping function.

The significant property of iterative maps and real dynamical systems is that the slightest uncertainty about initial state leads to very big uncertainty after some time. Iterative maps have been extensively studied in dynamical systems literature [9,10,7], and have exhibited very rich behaviour.

A lack of methods for the predictability of iterative maps leads to the questions about their computational complexity. It is interesting that even simple piecewise affine maps in dimension two [15] or closed-form analytic maps in dimension one [14] can simulate a Turing machine. The current research in this area is focused on analysing the complexity of iterative maps in low dimensions.

* Work partially supported by The Nuffield Foundation grant NAL/00684/G.

Along this line we obtained a new result about universality of piecewise iterative maps defined by a class of elementary functions.

Although we consider a class of function which is wider than a class of linear or affine functions studied in dimensions two and three [6,7,15] we show that piecewise iterative maps defined by a very restricted basis of elementary functions:

$$\{x^2, x^3, \sqrt[2]{x}, \sqrt[3]{x}, x \pm 1, 10 \cdot x\}$$

can simulate a Minsky machine even in dimension one.

Another interesting aspect of this result is related to the fact that a one dimensional iterative map can be simulated by a planar pseudo-billiard system (PBS). Planar PBS is a system of plain curve borders and assigned to them vector fields. The computation in such system is performed by placing a particle on one of the borders that moves according to it vector field. The dynamics of the particle in such environment is controlled by collisions with borders and it changes instantaneously at the moment of a collision with the boundary to the velocity defined by a given vector field on the boundary of the region.

On the one hand the computation in PBS is an abstraction that was defined on the basis of models in manufacturing, called switching arrival and switched server systems [5,8,11,12] that in one's turn describe the blocks in logistics, where one production units has to load/unload a number of subsequent/foregoing production units. On another hand this phenomena can theoretically arises for billiards in a strong magnetic or gravitational field, where only the angle with the field matters [5].

It has been shown in [5] that pseudo-billiard systems can demonstrate chaotic behaviour that indicates the potential presence of undecidability or even universality [2]. In this paper we support this idea by showing the universal computation ability of the planar PBS with a finite number of non-linear borders. The question of the computation power of planar PBS with finite number of polygon regions (i.e. with linear borders) is related to another open problem about computation in low dimensional systems like predictability of one-dimensional piecewise affine iterative maps [15].

2 Preliminaries

In what follows we use traditional denotations $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ for the sets of naturals (non-negative integers), integers, rationals and real respectively.

Definition 1. *A Piecewise (one-dimensional) elementary function* [1] *is a function that is defined on a sequence of disjoint intervals $I_i = [l_i, u_i]$ with l_i,$u_i \in \mathbb{Q}$, $i = 1..k$ and uses different elementary formulas for different parts of its domain $I = \{I_1 \cup \ldots \cup I_k\}$.*

[1] Piecewise liner functions, piecewise affine functions or piecewise polynomial functions are defined in the same obvious way.

Definition 2. *Let us consider the iterative map*

$$x_{n+1} = f(x_n)$$

where the function $f(x)$ is used to compute the next x with x_n as a starting point. A piecewise elementary map (PEM) is an iterative map where $f(x)$ is a piecewise elementary function.

The computation in the above system can be understand as a generation of sequence of points. One of the obvious problem that arises in such systems is a reachability problem that can be formulated as follows:

Problem 1. Given two points $x, y \in \mathbb{Q}$ and a one-dimensional piecewise elementary map P. Decide whether y is reachable from x in P.

As a main result of this paper we show universality of a one-dimensional piecewise elementary map for a case where the set of elementary functions is restricted by using x^2, x^3, $\sqrt[2]{x}$, $\sqrt[3]{x}$, $x \pm 1$, $10 \cdot x$ and their composition. That gives us undecidability of the Problem 1 and universality of planar pseudo-billiard system with a finite number of non-linear borders which we discuss later.

3 Main Result

Let us consider a piecewise elementary map P which is defined by composition of x^2, x^3, $\sqrt[2]{x}$, $\sqrt[3]{x}$, $x \pm 1$ and $10 \cdot x$ for different parts of a piecewise function domain.

In order to prove the universality of one-dimensional PEM we construct a piecewise elementary map that can simulate the computation of any 2-counter machine (Minsky machine). Actually we need to show how the states, transition function and updates of counters can be simulated by a piecewise elementary map P.

Let A be a 2-counter machine with a set of states $S = \{1, 2, \dots, n\}$. The configuration of A is a triple $[k, l, s]$ where k and l are values of two counters and s is a current state of A. Let us define the mapping $\phi : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ that is an isomorphism between a configuration $[k, l, s]$ of A and a rational number $s.0^{2^k 3^l}1$:

$$\phi([k, l, s]) \to s + \frac{1}{10^{2^k \cdot 3^l + 1}}.$$

Instead of classical Minsky machine from now on we will consider a well-known equivalent model of two counter machine where one of the counters is used as a scratchpad. Another, counter holds an integer whose prime factorization is $2^c \cdot 3^d$. The exponents c, d can be thought of as two virtual counters that are being simulated. If the real counter is set to zero then incremented once, that is equivalent to setting all the virtual counters to zero. If the real counter is doubled, that is equivalent to incrementing c, and if it is halved, that is equivalent to decrementing c. By a similar procedure, it can be multiplied or divided by 3, which is equivalent to incrementing or decrementing d.

To check if a virtual counter such as c (d) is equal to zero, just divide the real counter by 2 (3), see what the remainder is, then multiply by 2 (3) and add back the remainder. That leaves the real counter unchanged. The remainder will have been nonzero if and only if c (d) was zero.

Let A be in state s and its configuration is represented by a number $s.0^m1$. Let us show that we can perform the operations of multiplication and division in a piecewise elementary map P.

First, we construct a system of intervals with elementary functions, associated to them, that allow us to check divisibility of m by 2 and 3 or in other words to perform a zero testing on counters of original Minsky machine. For each state s of a counter machine we define the following intervals and functions:

$$\begin{array}{c}
\text{If } x \in (s, s+1) \text{ then apply } (x-s)^2 + (x-s) + 2s \\
\text{If } x \in (2s, 2s+0.1) \text{ then apply } 10(x-2s) + 3s \\
\text{If } x \in (3s, 3s+0.1) \text{ then apply } 10(x-3s) + 4s \\
\text{If } x \in (4s, 4s+0.1) \text{ then apply } 10(x-4s) + 5s \\
\text{If } x \in (5s, 5s+0.1) \text{ then apply } 10(x-5s) + 6s \\
\text{If } x \in (6s, 6s+0.1) \text{ then apply } 10(x-6s) + 7s \\
\text{If } x \in (7s, 7s+0.1) \text{ then apply } 10(x-7s) + 2s
\end{array}$$

It is easy to see that in this system of intervals any point of the form $s.0^m1$ will be mapped to $2s + 0.0^m10^m1$ and then after m iterations to the point $i \cdot s + 0.10^m1$. Note that $(i-1)$ is divided by 2 (3), if and only if m is divided by 2 (3).

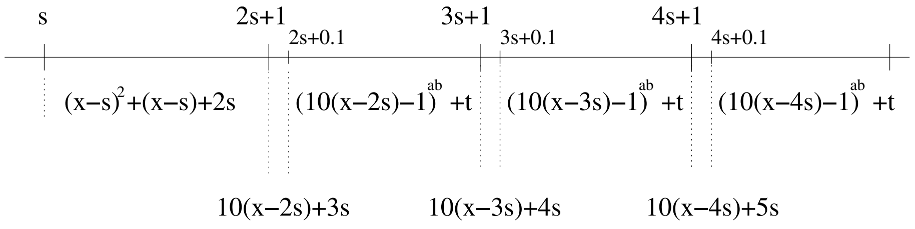


Fig. 1. A part of universal one-dimensional piecewise elementary map

Now we extend our piecewise function to simulate state transitions and update of counters by the following intervals and functions:

If $x \in (i \cdot s + 0.1, i \cdot s + 1)$, $i \in \{2, 3, 4, 5, 6, 7\}$ then apply $(10(x - i \cdot s) - 1)^{a \cdot b} + t$

where $a = 2$ ($a = \frac{1}{2}$) stands for increasing (decreasing) of the first counter by 1, and $b = 3$ ($a = \frac{1}{3}$) stands for increasing (decreasing) of the second counter by 1 (see Figure 1). Thus, this part of piecewise function express a transition of machine A from state s to state t and counters updates assuming that their values satisfy to divisibility by 2 and 3.

In order to finish the construction of 1-dimensional PEM that models machine A we redefine a set of states S by changing it to a set $S' = \{8 \cdot s | s \in S\}$

that gives us $7 \cdot |S|$ disjoint intervals in piecewise elementary function. Since the computation of a Minsky machine can be simulated by a specially designed PEM the following theorem holds:

Theorem 1. *One-dimensional piecewise elementary map is the universal model of computations.*

Corollary 1. *The reachability problem (Problem 1) for 1-dimensional PEM is undecidable.*

4 Computation in Pseudo-Billiard System

In this section we consider the pseudo billiard model that already appear in a different context and became an abstract framework for some practical problems. By the pseudo billiard as we defined before we understand a number of curve borders with assigned to them vector fields. The computation in this system can be described by the dynamics of the particle, which initially moves with the constant velocity (in a particular direction) inside a given region (not necessarily a polyhedron) and changes it instantaneously at the moment of a collision with the boundary to the velocity defined by a given vector field (not necessarily a constant one) on the boundary of the region. According to [5] this phenomena can arise for billiards in a strong magnetic or gravitational field, where only the angle with the field matters.

In general we can define a variety of models by making some of the model assumptions weaker or stronger. One of such submodel is the piecewise constant derivative (PCD) system studied in [1]. PCD system is the planar pseudo-billiard system with polyhedron regions such that in each region the particle can move in only one fixed direction. It was shown in [3] that the computation in such system is predictable and have effectively decidable reachability problem, but the model becomes universal in three-dimensional case. The pseudo-billiard planar system with predefined vectors assigned to the boarders of region is more complex then planar PCD system.

So, formally speaking, the planar pseudo-billiard system is defined by a finite set of borders $b_1, \ldots, b_h$ which are plane curves. Each border has assigned vector fields defined by two differential equations of the form

$$\dot{x} = k_x \ , \ \dot{y} = k_y$$

where k_x, k_y are integer constants, i.e. $k_x, k_y \in \mathbb{Z}$.

We show now that the pseudo-billiard planar system defined by non-linear borders with constant derivatives is the universal model of computations. The idea is based on simulation of one-dimensional piecewise elementary iterative map constructed in Section 3 by a planar pseudo-billiard system.

Theorem 2. *A planar pseudo-billiard system with a finite number of non-linear borders has ability of universal computation.*

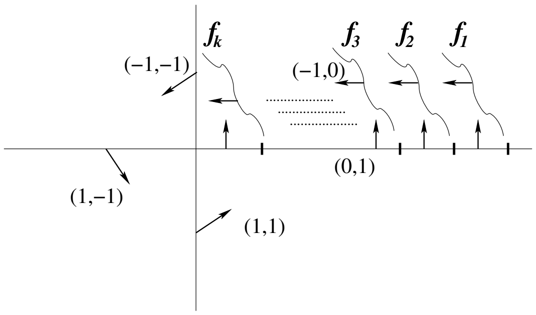


Fig. 2. Planar pseudo-billiard system

Proof. Let us construct a set of PBS borders with assigned to them vectors that can simulate a computation in a piecewise one-dimensional function. We assume that the computation starts from an initial point on the curve $b_1 = \{(x, y) \in \mathbb{R}^2 | x > 0, y = 0\}$ and the vector field assigned to b_1 is defined by

$$\dot{x} = 0 \ , \ \dot{y} = 1.$$

Let $f(x) = \{f_1, f_2, \ldots, f_k\}$ is a PEM that can simulate a Minsky Machine. The next border $b_2 = \{(x, y) \in \mathbb{R}^2 | y = f(x)\}$ of the planar PBS is exactly an image of piecewise one-dimensional function $f(x)$ on the plane, see Figure 2. For this border we assign a vector defined by

$$\dot{x} = -1 \ , \ \dot{y} = 0.$$

The idea of construction of other three borders b_3, b_4 and b_5 is to return the next computed value of one-dimensional map $f(x)$ to its initial domain on the border b_1. We define borders b_3, b_4 and b_5 and their vectors as follows:

$$\begin{array}{rl} b_3 = \{(x, y) \in \mathbb{R}^2 | x = 0, y > 0\} & \dot{x} = -1 \ , \ \dot{y} = -1, \\ b_4 = \{(x, y) \in \mathbb{R}^2 | x < 0, y = 0\} & \dot{x} = 1 \ , \ \dot{y} = -1, \\ b_5 = \{(x, y) \in \mathbb{R}^2 | x = 0, y < 0\} & \dot{x} = 1 \ , \ \dot{y} = 1. \end{array}$$

In principle one can define a similar system where the particle will not go through the walls. In order to do so we need to place each function f_i on a different level in such a way that each reachable point on the border b_3 will be associated to the reflection from the unique function $f_j \in \{f_1, f_2, \ldots, f_k\}$. Then we need to split b_3 on several partitions and define such vectors that will gather the particles at the same subpart of a border b_4. The rest of construction should be the same.

5 Conclusion

In this paper we have shown universality of one-dimensional piecewise iterative maps defined by a subclass of elementary functions and planar pseudo-billiard system with a finite number of non-linear borders. It would be interesting to check what could happened if we consider any other subclasses of piecewise elementary maps. For example, the computational power of one-dimensional piecewise elementary maps which are defined by polynomial functions is open. Another interesting direction of this research would be analysis of planar pseudo-billiard systems with linear borders which seems to be quite non-trivial problem.

References

1. E. Asarin, O. Maler, A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives, Theoretical Computer Science 138, (1995) 35-65
2. E. Asarin. Chaos and Undecidability. Invited talk at the workshop on Hybrid Systems at Grenoble, 1995.
3. E. Asarin, G. Schneider, S.Yovine. On the Decidability of the Reachability Problem for Planar Differential Inclusions. Hybrid Systems: Computation and Control, LNCS 2034 (2001) 89-104
4. E.Asarin, O.Maler. Discrete-continuous systems: an algorithmic aspect (in Russian). Avtomatika i Telemekhanika (1995) No 5, 124-137
5. M. Blank and L. Bunimovich. Switched flow systems: pseudo billiard dynamics. Dynamical Systems. V.19, No.4 (2004) 359-370
6. V.Blondel, O.Bournez, P.Koiran, Christos H. Papadimitriou, John N. Tsitsiklis: Deciding stability and mortality of piecewise affine dynamical systems. Theor. Comput. Sci. 255(1-2) (2001) 687-696
7. V.Blondel and J.Tsitsiklis. A survey of computational complexity results in systems and control, Automatica 36 (2000) pp. 1249-1274.
8. Chase C., Serrano J., and Ramadge P.J. Periodicity and chaos from switched flow systems:contrasting examples of discretely controlled continuous systems. IEEE Trans. Automatic Control, 38 (1993) 70-83
9. P.Collet and J. Eckmann. Iterated Maps on the Interval as Dynamical Systems. Boston, MA: Birkhuser, 1980.
10. J. Froyland, Introduction to chaos and coherence. Institute of Physics Publishing, Bristol and Philadelphia (1994), 130 p.
11. Horn C., Ramadge P.J. A topological analysis of a family of dynamical systems with non-standard chaotic and periodic behavior. Int. J. Control, 67, (1997) 979-996.
12. Katzorke I., Pikovsky A. Chaos and complexity in a simple model of production dynamics. Discrete Dynamics in Nature and Society, 5, (2000) 179-187.
13. P. Koiran. My Favourite Problems, http://www.ens-lyon.fr/∼koiran/problems.html
14. P. Koiran and C. Moore. Closed-form analytic maps in one and two dimensions can simulate Turing machines. Theoretical Computer Science 210(1) (1999) 217-223
15. Pascal Koiran, Michel Cosnard, Max H. Garzon. Computability with Low-Dimensional Dynamical Systems. Theor. Comput. Sci. 132(2) (1994) 113-128
16. Peters K., Parlitz U. Hybrid systems forming strange billiards. Int. J. of Bifurcations and Chaos, 19, (2003) 2575-2588.
17. Schurmann T., Hoffman I. The entropy of strange billiards inside n-simplexes. J. Phys. A 28, (1995) 5033-5039.

On the Importance of Parallelism for Quantum Computation and the Concept of a Universal Computer

Marius Nagy and Selim G. Akl

School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
{marius, akl}@cs.queensu.ca

Abstract. The role played by parallelism in the theory of computation depends on the particular paradigm or computational environment considered, but its importance has been confirmed with the emergence of each novel computing technology. In this paper we study the implications of parallelism in quantum information theory and show that a parallel approach can make the difference between success and failure when trying to distinguish among (entangled) quantum states. A (perhaps surprising) consequence of this fact is the impossibility of constructing a Universal Computer, as defined herein.

1 Introduction

Parallel computing was originally motivated by the need to speed up computation, especially for those tasks whose sequential running time is prohibitively long. This traditional view of the role played by parallelism in computation has since evolved dramatically, with implications almost impossible to foresee when the field originated.

We know today that there are tasks and computational paradigms for which a parallel approach offers much more than just a faster solution [4]. A real-time environment, constraining the input data provided and the output produced at various moments in time, can have drastic effects on the quality of the solution obtained for a certain problem, unless parallelism is employed [13,14,15,16]. A general framework is developed in [2] to show how a superlinear (with respect to the number of processors employed in the parallel approach) improvement in the quality of the solution computed to a real-time problem can be obtained.

In other cases, a sequential machine fails to tackle a certain task altogether, and parallelism is the only hope to see that task accomplished. Examples of this kind include measuring the parameters of a dynamical system [1,7] or setting them in such a way as to avoid pushing the system into a chaotic behavior [6]. Also, some geometric transformations can only be performed successfully if we act simultaneously on a certain number of objects [3].

Progress in science and technology influences the way computations are carried and the emergence of novel computational environments and paradigms continually broadens the applicability and importance of parallelism. In this paper we exhibit an example of a problem from quantum information theory that clearly emphasizes the role of parallelism in this relatively new field of computation governed by the principles of quantum mechanics. The example we present also reinforces the argument developed in [5] demonstrating the infeasibility of a Universal Computer obeying certain conditions.

The remainder of the paper is organized as follows. The next section is intended to make the reader familiar with the fundamental notions of quantum computation. Section 3 introduces the problem of distinguishing quantum states and analyzes the instance defined by the four Bell states. A generalization to an arbitrary number of qubits entangled together is developed in section 4. Section 5 discusses the relevance of the problem investigated, in the context of Universal Computation. The contributions of this paper, stressing the importance of parallelism in general and for the concept of a Universal Computer in particular, are summarized in the last section. We also mention a possible continuation of this work, with interesting implications.

2 Fundamentals of Quantum Computation and Quantum Information

This section introduces the basic elements of quantum computation and quantum information to the extent needed for a clear exposition of the main ideas presented in this paper.

Quantum information theory was developed much in analogy with classical information theory, enlarging the scope of the latter. Thus, quantum information theory deals with all the static resources and dynamical processes investigated by classical information theory, as well as additional static and dynamic elements that are specific to quantum mechanics.

2.1 The Qubit

Probably, the most fundamental quantum resource manipulated by quantum information theory is the quantum analogue of the classical bit, called the *qubit*.

Though it may have various physical realizations, as a mathematical object the qubit is a unit vector in a two-dimensional state space, for which a particular orthonormal basis, denoted by $\{|0\rangle, |1\rangle\}$ has been fixed. The two basis vectors $|0\rangle$ and $|1\rangle$ must be orthogonal (*i.e.* their inner product is zero) and normalized (*i.e.* of unit length each), hence the orthonormality requirement. The basis vectors correspond to the two possible values a classical bit can take. However, unlike classical bits, a qubit can also take many other values. In general, an arbitrary qubit $|\psi\rangle$ can be written as a linear combination of the computational basis states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1}$$

where α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$ (the normalization condition ensuring that $|\psi\rangle$ is a unit vector). In order to describe the state of a qubit or ensemble of qubits in a compact way, we have adopted here the well-established *bra/ket* notation introduced by Dirac [10]. According to his conventional notation, *kets* like $|x\rangle$ are simply column vectors, typically used to describe quantum states. Similarly, the matching *bra* $\langle x|$ is a row vector denoting the conjugate transpose of $|x\rangle$. Thus, the *ket* $|\psi\rangle$ in equation (1) is just a short notation for the complex column vector

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix},$$

while its corresponding *bra* $\langle\psi|$ denotes the row vector $[\alpha^* \; \beta^*]$, where α^* and β^* are the complex conjugates of the complex numbers α and β, respectively.

Equation (1) reflects the fundamental difference distinguishing quantum bits from classical ones and is a direct application of the quantum principle of superposition of states. The qubit $|\psi\rangle$ is in a superposition of $|0\rangle$ and $|1\rangle$, a state in which it is not possible to say that the qubit is definitely in the state $|0\rangle$, or definitely in the state $|1\rangle$.

For a single qubit, there is a very intuitive geometric representation of its state as a point on a sphere. Taking $\alpha = e^{i\gamma}\cos(\theta/2)$ and $\beta = e^{i\gamma}e^{i\varphi}\sin(\theta/2)$ in equation (1), we can rewrite the state of qubit $|\psi\rangle$ as

$$|\psi\rangle = e^{i\gamma}(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle), \tag{2}$$

where θ, φ and γ are real numbers. Note that this is always possible since $|\alpha|^2 + |\beta|^2 = 1$. Also, because a global phase factor like $e^{i\gamma}$ has no observable effects (*i.e.* it does not influence the statistics of measurement predicted for qubit $|\psi\rangle$), we can effectively ignore it. Consequently, the pair (θ, φ) uniquely identifies a point $(\cos\varphi\sin\theta, \sin\varphi\sin\theta, \cos\theta)$ on a unit three-dimensional sphere called the *Bloch sphere* [19,17].

Figure 1 depicts four possible states of a qubit using the Bloch sphere representation. Note that the states corresponding to the points on the equatorial circle have all equal contributions of 0-ness and 1-ness. What distinguishes them is the *phase*. For example, the two states displayed above, $1/\sqrt{2}(|0\rangle + |1\rangle)$ and $1/\sqrt{2}(|0\rangle - |1\rangle)$ are the same up to a relative phase shift of π, because the $|0\rangle$ amplitudes are identical and the $|1\rangle$ amplitudes differ only by a relative phase factor of $e^{i\pi} = -1$.

2.2 Measurements

We now turn our attention to the amount of information that can be stored in a qubit and, respectively, retrieved from a qubit. Since any point on the Bloch sphere can be characterized by a pair of real-valued parameters taking continuous values, it follows that, theoretically, a qubit could hold an infinite amount of information. As it turns out, however, we cannot extract more information from such a qubit than we are able to extract from a classical bit. The reason is

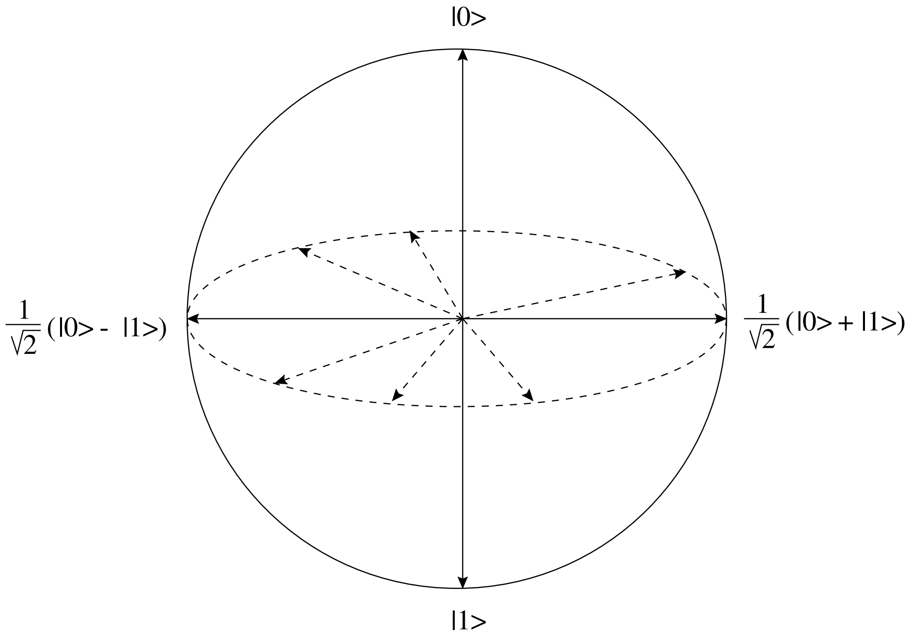

|0>
|1>
$\frac{1}{\sqrt{2}}$(|0> - |1>)
$\frac{1}{\sqrt{2}}$(|0> + |1>)

Fig. 1. The Bloch sphere representation of a qubit

that we have to *measure* the qubit in order to determine in which state it is. Yet, according to a fundamental postulate of quantum mechanics (Postulate 3 in [17]), the amount of information that can be gained about a quantum state through measurement is restricted. Thus, when we measure a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with respect to the standard basis for quantum computation $\{|0\rangle, |1\rangle\}$, we get either the result 0 with probability $|\alpha|^2$, or the result 1 with probability $|\beta|^2$. The condition that the probabilities must sum to one corresponds geometrically to the requirement that the qubit state be normalized to length 1, that is the inner product

$$\langle\psi|\psi\rangle = [\alpha^* \ \beta^*] \cdot \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \alpha\alpha^* + \beta\beta^*$$

equals 1.

Furthermore, measurement alters the state of a qubit, collapsing it from its superposition of $|0\rangle$ and $|1\rangle$ to the specific state consistent with the result of the measurement. For example, if we observe $|\psi\rangle$ to be in state $|0\rangle$ through measurement, then the post-measurement state of the qubit will be $|0\rangle$, and any subsequent measurements (in the same basis) will yield 0 with probability 1.

Naturally, measurements in bases other than the computational basis are always possible, but this will not help us in determining α and β from a single measurement. In general, measurement of a state transforms the state into one of the measuring device's associated basis vectors. The probability that the state is measured as basis vector $|u\rangle$ is the square of the norm of the amplitude of the component of the original state in the direction of the basis vector $|u\rangle$. Unless the basis is explicitly stated, we will always assume that a measurement is performed with respect to the standard basis for quantum computation.

2.3 Putting Qubits Together

Let us examine now more complex quantum systems, composed of multiple qubits. In classical physics, individual two-dimensional state spaces of n particles combine through the Cartesian product to form a vector space of $2n$ dimensions, representing the state space of the ensemble of n particles. However, this is not how a quantum system can be described in terms of its components. Quantum states combine through the tensor product to give a resulting state space of 2^n dimensions, for a system of n qubits.

For a system of two qubits, each with basis $\{|0\rangle, |1\rangle\}$, the resulting state space is the set of normalized vectors in the four dimensional space spanned by basis vectors $\{|0\rangle \otimes |0\rangle, |0\rangle \otimes |1\rangle, |1\rangle \otimes |0\rangle, |1\rangle \otimes |1\rangle\}$, where $|x\rangle \otimes |y\rangle$ denotes the tensor product between column vectors $|x\rangle$ and $|y\rangle$. For example,

$$|0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \times 0 \\ 1 \times 1 \\ 0 \times 0 \\ 0 \times 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} .$$

It is customary to write the basis in the more compact notation $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. This generalizes in the obvious way to an n-qubit system with 2^n basis vectors.

2.4 Entanglement

Similar to single qubits, multiple-qubit systems can also be in a superposition state. The vector

$$|\Psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \tag{3}$$

describes a superposition state of a two-qubit system in which all four components (corresponding to the four basis vectors) have equal amplitudes. What about the two qubits composing the system? Can we characterize their states individually? If we rewrite equation (3) in order to express $|\Psi\rangle$ as the tensor product

$$|\Psi\rangle = (\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) \otimes (\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) \tag{4}$$

then we can legitimately assert that each of the component qubits is also in a superposition state, perfectly balanced between $|0\rangle$ and $|1\rangle$. Now let us drop the two middle terms in equation (3) and consider the superposition state described by

$$|\Phi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \tag{5}$$

In this case it is no longer possible to find complex numbers α, β, γ and δ such that

$$\begin{aligned}
&(\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle) = \\
&= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle \\
&= \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle
\end{aligned} \tag{6}$$

The state of the system cannot be decomposed into a product of the states of the constituents. Even though the state of the system is well defined (through the state vector $|\Phi\rangle$), neither of the two component qubits is in a well-defined state. This is again in contrast to classical systems, whose states can always be broken down into the individual states of their components. Furthermore, if we try to measure the two qubits, the superposition will collapse into one of the two basis vectors contributing to the superposition, and the outcomes of the two measurements will always coincide. In other words, if one of the qubits is found to be in state $|0\rangle$, then the second one will necessarily be in the same state, while a state $|1\rangle$ observed through measurement will be shared by both qubits. Therefore, we say that the two qubits are *entangled*[1] and $|\Phi\rangle$ describes an entangled state of the system.

Entanglement defines the strong correlations exhibited by two or more particles when they are measured, and which cannot be explained by classical means. This does not imply that entangled particles will always be observed in the same state, as entangled states like

$$\frac{1}{\sqrt{2}}|01\rangle \pm \frac{1}{\sqrt{2}}|10\rangle \tag{7}$$

prove it. In this last example, a measurement will always reveal the two qubits to be in opposite states (when one is 0 the other is 1 and vice-versa). States like these or the one in equation (5) are known as *Bell states* or *EPR pairs* after some of the people [11,8] who pointed out their strange properties.

In some sense, we can say that superposition encompasses entanglement, since entanglement can be viewed as a special case of superposition. It is also interesting to make an analogy between entanglement and the concept of primality from number theory. Indeed, an entangled state of the system corresponds to a prime number, since it cannot be factored or decomposed as a product of subsystem states.

3 Quantum Distinguishability

We introduce the problem of distinguishing quantum states through a metaphor involving two prototypical characters, named Alice and Bob. Suppose we have a fixed set of quantum states described using the usual Dirac notation $|\Psi_i\rangle$ $(1 \leq i \leq n)$ known to both Alice and Bob. Alice randomly chooses a state from the set

[1] It was Schrödinger who actually named the phenomenon *entanglement* in 1935 [18].

and prepares a qubit (or set of qubits) in that particular state. She then gives the qubit(s) to Bob who is free to investigate them in any way he likes. To be more specific, Bob can apply any kind of measurement on the qubit(s) and possibly process and/or interpret the information acquired through measurement. In the end, his task is to identify the index i of the state characterizing the qubit(s) Alice has given him.

The only case in which a set of quantum states can be reliably (that is, 100% of the time) distinguished from one another is if they are pairwise orthogonal. For example, the four states $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ form an orthonormal basis (each vector is a unit vector and distinct vectors have a zero inner product) for the state space spanned by two qubits. Consequently, they can be reliably distinguished by an appropriate measurement. In this case, we can simply measure each qubit (sequentially) in the computational basis (defined by the basis vectors $|0\rangle$ and $|1\rangle$).

On the other hand, it is impossible to reliably distinguish $|0\rangle$ from $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. While the first state will consistently yield a 0 upon measurement, the second state also has a 50% chance to be observed as a 0. It is this component in the direction of the basis vector $|0\rangle$ which is present in both quantum states that prevents us from distinguishing them reliably. If the vectors describing the quantum states would be orthogonal, then a measurement basis would exist with respect to which the quantum states share no common components.

Consider now the case in which we try to distinguish among the four Bell states

$$\tfrac{1}{\sqrt{2}}|00\rangle + \tfrac{1}{\sqrt{2}}|11\rangle,\ \tfrac{1}{\sqrt{2}}|00\rangle - \tfrac{1}{\sqrt{2}}|11\rangle,\ \tfrac{1}{\sqrt{2}}|01\rangle + \tfrac{1}{\sqrt{2}}|10\rangle,\ \tfrac{1}{\sqrt{2}}|01\rangle - \tfrac{1}{\sqrt{2}}|10\rangle.$$

No sequential approach (that is, measuring the qubits one after the other) will be of any help here, regardless of the basis in which the measurements are performed. By measuring the two qubits, in sequence, in the computational basis, Bob can distinguish the states $\frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle)$ from $\frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$. He does this by checking if the outcomes of the two measurements are the same or not. But this kind of measurement makes it impossible to differentiate between $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and $\frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$, or between $\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$ and $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$.

Alternatively, Bob can decide to perform his measurements in a different basis, like $(|+\rangle, |-\rangle)$, where the basis vectors are

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle,$$

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle.$$

Due to the fact that

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{|++\rangle + |--\rangle}{\sqrt{2}}$$

and

$$\frac{|00\rangle - |11\rangle}{\sqrt{2}} = \frac{|+-\rangle + |-+\rangle}{\sqrt{2}},$$

Bob can now reliably distinguish the quantum state $\frac{1}{\sqrt{2}}(|00\rangle+|11\rangle)$ from $\frac{1}{\sqrt{2}}(|00\rangle-|11\rangle)$. Indeed, if the two qubits yield identical outcomes when measured in this new basis, then we can assert with certainty that the state was not $\frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$. Similarly, if the measurement outcomes for the qubits are different, the original state could not have been $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Unfortunately, in this new setup, the quantum states $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and $\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$ become indistinguishable and the same is true about $\frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$ and $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$.

The computational bases $(|0\rangle, |1\rangle)$ and $(|+\rangle, |-\rangle)$ are, respectively, the two extremities of an (theoretically) infinite number of choices for the basis relative to which the quantum measurements are to be performed. But even though the separation line between the four Bell states will drift with the choice of the basis vectors, the two extreme cases discussed above offer the best possible distinguishability.

Intuitively, this is due to the entanglement exhibited between the two qubits in all four states. As soon as the first qubit is measured (regardless of the basis), the superposition describing the entangled state collapses to the specific state consistent with the measurement result. In this process, some of the information originally encapsulated in the entangled state is irremediably lost. Consequently, measuring the second qubit cannot give a complete separation of the four EPR states. But the Bell states do form an orthonormal basis, which means that (at least theoretically) they can be distinguished by an appropriate quantum measurement. However, this measurement must be a *joint* measurement of both qubits simultaneously, in order to achieve the desired distinguishability. Not surprisingly, this is very difficult to accomplish in practice.

The distinguishability of the four Bell (or EPR) states is the key feature in achieving superdense coding [9]. However, in the experimental demonstration of this protocol [12] two of the possibilities cannot be distinguished from one another, precisely because of the difficulties associated with implementing a joint measurement.

4 Generalization

A more compact representation of the Bell basis is through a square matrix where each column is a vector describing one of the Bell states:

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix}$$

The elements of each column are the amplitudes or proportions in which the computational basis states $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ are present in the respective EPR state.

This scenario can be extended to ensembles of more than two qubits. The following matrix describes eight different entangled states that cannot be reliably distinguished unless a joint measurement of all three qubits involved is performed:

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1&0&0&0&0&0&0&1 \\ 0&1&0&0&0&0&1&0 \\ 0&0&1&0&0&1&0&0 \\ 0&0&0&1&1&0&0&0 \\ 0&0&0&1&-1&0&0&0 \\ 0&0&1&0&0&-1&0&0 \\ 0&1&0&0&0&0&-1&0 \\ 1&0&0&0&0&0&0&-1 \end{pmatrix}$$

In general, for a quantum system composed of n qubits, one can define the following 2^n entangled states of the system:

$$\begin{aligned} &\frac{1}{\sqrt{2}}(|000\cdots 0\rangle \pm |111\cdots 1\rangle) \\ &\frac{1}{\sqrt{2}}(|000\cdots 1\rangle \pm |111\cdots 0\rangle) \\ &\qquad\qquad\vdots \\ &\frac{1}{\sqrt{2}}(|011\cdots 1\rangle \pm |100\cdots 0\rangle) \end{aligned} \tag{8}$$

These vectors form an orthonormal basis for the state space corresponding to the n-qubit system. The only chance to differentiate among these 2^n states using quantum measurement(s) is to observe the n qubits simultaneously, that is, perform a single joint measurement of the entire system. In the given context, *joint* is really just a synonym for *parallel*. Indeed, the device in charge of performing the joint measurement must posses the ability to "read" the information stored in each qubit, in parallel, in a perfectly synchronized manner. In this sense, at an abstract level, the measuring apparatus can be viewed as having n probes. With all probes operating in parallel, each probe can "peek" inside the state of one qubit, in a perfectly synchronous operation. The information gathered by the n probes is seen by the measuring device as a single, indivisible chunk of data, which is then interpreted to give one the 2^n entangled states as the measurement outcome.

From a mathematical (theoretical) point of view, such a measurement operator can be easily constructed by defining each of the 2^n states that are to be distinguished to be a projector associated with the measurement operation. We are well aware though, that a physical realization of this mathematical construction is extremely difficult, if not impossible to achieve in practice, with today's technology. The experimental demonstration of the superdense coding protocol mentioned at the end of previous section clearly shows this difficulty (for just two qubits!). Yet, if there is any hope to see a joint measurement performed in the future, then only a device operating in a parallel synchronous fashion on all n qubits (as explained above) would succeed.

It is perhaps worth emphasizing that if such a measurement cannot be applied then the desired distinguishability can no longer be achieved regardless of how many other measuring operations we are allowed to perform. In other words, even an infinite sequence of measurements touching at most $n-1$ qubits at the same time cannot equal a single joint measurement involving all n qubits.

Furthermore, with respect to the particular distinguishability problem that we have to solve, a single joint measurement capable of observing $n-1$ qubits simultaneously offers no advantage whatsoever over a sequence of $n-1$ consecutive *single* qubit measurements. This is due to the fact that an entangled state like

$$\frac{1}{\sqrt{2}}(|000\cdots 0\rangle + |111\cdots 1\rangle)$$

cannot be decomposed neither as a product of $n-1$ individual states nor as a product of two states (one describing a single qubit and the other describing the subsystem composed of the remaining $n-1$ qubits). Any other intermediate decomposition is also impossible.

Overall, our distinguishability problem can only be tackled successfully within a parallel approach, where we can measure all qubits simultaneously. In this sense, distinguishing among entangled quantum states can be viewed as a quantum variant of the measure-compute-set problem formulated in [1], which also admits only a parallel solution.

5 Universal Computation

Finally, we relate the example presented in this paper with the hypothetical notion of a Universal Computer, introduced in [5]. Such a machine must be able to follow (execute) the steps of any program made up of basic input, output and internal processing operations. The Universal Computer is intended to be the most general possible model of computation, encompassing all existing or imagined computational paradigms. Specifically, its internal processing capabilities include (but are not limited to) basic arithmetic and logical operations, unitary quantum gates, operations specific to DNA and natural computing, etc. It must also have a means of communicating with the outside world at any time during a computation, either for receiving input or producing output (results). The machine is endowed with the ability to acquire input data through measurements on outside-world systems, performed by a set of probes (or sensors). The program, the input data (either received or acquired), the output and all intermediate results are stored in (and can be retrieved from) a memory which is generously allowed to be unlimited.

To make this Universal Computer a “realistic” model of computation, it is subjected to the *finiteness condition*: In one step, requiring one time unit, the Universal Computer can execute a finite and fixed number of basic operations (including measurements). It is precisely this limitation (quite natural and reasonable) that makes the Universal Computer a utopian concept. Specifically, three classes of computable functions $\mathcal{F}$ are described in [5], which cannot be computed by any machine obeying the finiteness condition.

One of these classes of problems involves measuring a set of interacting variables. Formally, suppose there are n variables x_0, x_1, $\cdots$, x_{n-1}. Although these variables may represent the parameters of a physical or biological system, the following formalism is abstracted away from any particular realization and does not necessarily describe the dynamics of a quantum system. The dependence of each variable on all others induces the system to continually evolve until a state of equilibrium may eventually be reached. In the absence of any external perturbations, the system can remain in a stable state indefinitely. We can model the interdependence between the n variables through a set of functions, as follows:

$$\begin{aligned} x_0(t+1) &= f_0(x_0(t), x_1(t), \ldots, x_{n-1}(t)) \\ x_1(t+1) &= f_1(x_0(t), x_1(t), \ldots, x_{n-1}(t)) \\ &\vdots \\ x_{n-1}(t+1) &= f_{n-1}(x_0(t), x_1(t), \ldots, x_{n-1}(t)) \end{aligned} \tag{9}$$

This system of equations describes the evolution of the system from state $(x_0(t), x_1(t), \ldots, x_{n-1}(t))$ to state $(x_0(t+1), x_1(t+1), \ldots, x_{n-1}(t+1))$, one time unit later. In the case where the system has reached equilibrium, its parameters will not change over time. It is important to emphasize that, in most cases, the dynamics of the system are very complex, so the mathematical description of functions $f_0, f_1, \ldots, f_{n-1}$ is either not known to us or we only have rough approximations for them.

Assuming the system is in an equilibrium state, our task is to measure its parameters in order to compute a function $\mathcal{F}$, possibly a global property of the system at equilibrium. In other words, we need the values of $x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)$ at moment τ, when the system is in a stable state, in order to compute

$$\mathcal{F}(x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)).$$

We can try to estimate the value of $x_0(\tau)$, for instance[2], by measuring the respective parameter at time τ. Although, for some systems, we can acquire

[2] The choice of x_0 here is arbitrary. The argument remains the same regardless of which of the n parameters we choose to measure first.

the value of $x_0(\tau)$ easily in this way, the consequences for the entire system can be dramatic. Unfortunately, any measurement is an external perturbation for the system, and in the process, the parameter subjected to measurement may be affected unpredictably.

Thus, the measurement operation will change the state of the system from $(x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$ to $(x_0'(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$, where $x_0'(\tau)$ denotes the value of variable x_0 after measurement. In those cases where the measurement process has a non-deterministic effect upon the variable being measured, we cannot estimate $x_0'(\tau)$ in any way. But, regardless of the particular instance of the model, the transition from $(x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$ (that is, the state before measurement) to $(x_0'(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$ (that is, the state after measurement) does not correspond to the normal evolution of the system according to its dynamics described by functions f_i, $0 \leq i < n$.

However, because the equilibrium state was perturbed by the measurement operation, the system will react with a series of state transformations, governed by equations (9). Thus, at each time step after τ, the parameters of the system will evolve either towards a new equilibrium state or maybe fall into a chaotic behavior. In any case, at time $\tau + 1$, all n variables have acquired new values, according to the expressions of functions f_i:

$$\begin{aligned} x_0(\tau+1) &= f_0(x_0'(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)) \\ x_1(\tau+1) &= f_1(x_0'(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)) \\ &\vdots \\ x_{n-1}(\tau+1) &= f_{n-1}(x_0'(\tau), x_1(\tau), \ldots, x_{n-1}(\tau)) \end{aligned} \tag{10}$$

Consequently, unless we are able to measure all n variables, in parallel, at time τ, some of the values composing the equilibrium state

$$(x_0(\tau), x_1(\tau), \ldots, x_{n-1}(\tau))$$

will be lost without any possibility of recovery.

The finiteness condition restricts in this case the number of variables that can be measured in parallel. So, if the Universal Computer is able to measure n variables in parallel (that is, during one step), where n can be arbitrarily large, but finite, then the Universal Computer will fail to solve the same problem for a system involving $n+1$ variables. In other words, the Universal Computer cannot simulate a computation that is perfectly possible for another machine. However, it is exactly the principle of *simulation* that lies at the heart of *universality*.

Choosing a machine endowed with $n + 1$ probes (and therefore capable of measuring $n+1$ variables in parallel) as the Universal Computer is not a solution. By an adversary argument, we can construct an instance of the above problem, only this time involving $n + 2$ parameters to be measured, and the Universal

Computer will fail once again to compute the required function $\mathcal{F}$, although it can be trivially computed by a machine with $n + 2$ probes. This argument is valid for *any* given Universal Computer, having a fixed (and finite) number of probes and therefore a limited degree of parallelism to tackle such inherently parallel tasks. It is important to emphasize that the computational paradigm to which the above setting belongs is not a conventional one. The input data necessary to compute $\mathcal{F}$ is not available at the outset and have to be acquired through measurement operations.

Coming back to the example presented in this paper, it is easy to see that a device capable of measuring at most n qubits simultaneously (where n is a fixed, finite number) will fail to solve the distinguishability problem for $n + 1$ qubits. Our example, taken from the quantum information area is similar in nature with the interacting variables example formalized above and supports the idea advanced in [5] about the impossibility of realizing the concept of a Universal Computer. In the case that we have described, interdependence between variables takes the form of entanglement between qubits, the phenomenon ultimately responsible for making a parallel approach imperative.

6 Conclusions

We have exhibited an example of a task which cannot be successfully completed unless a parallel approach is employed. The task is to distinguish among the elements of a set of quantum states, using any quantum measurements that can be theoretically applied. There are no restrictions concerning the number of measurements allowed or the time when the task has to be completed. We have shown that there exists a set of entangled states, forming an orthonormal basis in the state space spanned by n qubits, for which only a *joint* measurement (in that respective basis) of all the qubits composing the system can achieve perfect distinguishability. An important characteristic of the task is that if the degree of parallelism necessary to successfully solve the problem is not available, then the solution is no better than a purely sequential approach. Such inherently parallel tasks have been shown to exist in a variety of environments, namely, real-time systems [4], dynamical systems [1,7,6] and geometric problems [3].

In this paper, we have shown that parallelism is equally important for yet another computational paradigm, essentially different from the classical theory of computation, namely quantum computation and quantum information. It is important to note that we refer here to the common understanding of the term *parallelism* and not to *quantum parallelism.* The latter syntagm is used to denote the ability to perform a certain computation simultaneously on all terms of a quantum superposition, regardless of the number of qubits composing the quantum register whose state is described by that superposition. As opposed to this interpretation, we refer to parallelism as the ability to act simultaneously on a certain number of qubits. Thus, we can rightfully assert that parallelism transcends the laws of physics and represents a fundamental aspect of computation, regardless of the particular physical way chosen to embody information.

The second contribution of the paper addresses the notion of a Universal Computer obeying the finiteness condition [5]. Distinguishing among entangled quantum states is, conceptually, a quantum example of measuring interdependent variables. This problem, arising in quantum information theory, strengthens the conclusion that there is no *finite*[3] computing device (conventional or unconventional) upon which the attribute *universal* can be bestowed.

This result holds as long as the candidate Universal Computer cannot apply its (internal) set of basic processing operations ("gates") onto systems from the outside world. In other words, its processing capabilities can only be exercised on data already stored in its (internal) memory. For the problem we have analyzed in this paper, all the input data on which the machine can work must be acquired through measurement(s).

An interesting research hypothesis is to allow the computing device to execute its program (set of operations) on systems belonging to the outside world. Then, the machine could first apply a series of quantum gates that changes the entangled basis (8) into the usual computational basis for n qubits, that is $\{|i\rangle,\ 0 \leq i \leq 2^n - 1\}$. Then, measuring each qubit in sequence is enough to distinguish among the 2^n states of the new basis. Such a scenario may reveal an example of a problem that can only be solved by a quantum computer and never by a classical one.

References

1. Selim G. Akl. Coping with uncertainty and stress: A parallel computation approach. to appear in International Journal of High Performance Computing and Networking.
2. Selim G. Akl. Discrete steepest descent in real time. *Parallel and Distributed Computing Practices*, 4(3):301–317, 2001.
3. Selim G. Akl. Inherently parallel geometric problems. Technical Report 2004–480, School of Computing, Queen's University, Kingston, Ontario, April 2004. 19 pages.
4. Selim G. Akl. Superlinear performance in real-time parallel computation. *The Journal of Supercomputing*, 29(1):89–111, 2004.
5. Selim G. Akl. The myth of universal computation. In R. Trobec, P. Zinterhof, M. Vajteršic, and A. Uhl, editors, *Parallel Numerics, Part 2, Systems and Simulation*, pages 211–236. University of Salzburg, Austria and Jožef Stefan Institute, Ljubljana, Slovenia, 2005.
6. Selim G. Akl, Brendan Cordy, and W. Yao. An analysis of the effect of parallelism in the control of dynamical systems. to appear in the International Journal of Parallel, Emergent and Distributed Systems.
7. Selim G. Akl and W. Yao. Parallel computation and measurement uncertainty in nonlinear dynamical systems. *Journal of Mathematical Modelling and Algorithms, Special Issue on Parallel and Scientific Computations with Applications*, 4:5–15, 2005.

[3] In the sense introduced in [5] and briefly described in the previous section.

8. John Bell. On the Einstein-Podolsky-Rosen paradox. *Physics*, 1:195–200, 1964.
9. Charles H. Bennett and S. J. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Physical Review Letters*, 69(20):2881–2884, 1992.
10. P. Dirac. *The Principles of Quantum Mechanics.* Oxford University Press, 4th edition, 1958.
11. Albert Einstein, Boris Podolsky, and Nathan Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical Review*, 47:777–780, May 1935.
12. Klaus Mattle, Harald Weinfurter, Paul G. Kwiat, and Anton Zeilinger. Dense coding in experimental quantum communication. *Physical Review Letters*, 76(25):4656–4659, 1996.
13. Marius Nagy and Selim G. Akl. Real-time minimum vertex cover for two-terminal series-parallel graphs. In *Proceedings of the Thirteenth IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 526–534, Anaheim, California, August 2001.
14. Marius Nagy and Selim G. Akl. Locating the median of a tree in real time. In *Proceedings of the Fourteenth IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 108–113, Cambridge, Massachusetts, November 2002.
15. Marius Nagy and Selim G. Akl. Computing nearest neighbors in real time. In *Proceedings of the Fifteenth IASTED International Conference on Parallel and Distributed Computing and Systems*, volume II, pages 518–524, Marina Del Rey, California, November 2003.
16. Naya Nagy and Selim G. Akl. The maximum flow problem: A real-time approach. *Parallel Computing*, 29:767–794, 2003.
17. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.
18. Erwin Schrödinger. Discussion of probability relations between separated systems. *Proceedings of the Cambridge Philosophical Society*, 31:555–563, 1935.
19. Eric W. Weisstein et al. *Bloch sphere.* From *MathWorld*–A Wolfram Web Resource. http://mathworld.wolfram.com/BlochSphere.html.

On Computational Complexity of Counting Fixed Points in Symmetric Boolean Graph Automata

Predrag T. Tošić and Gul A. Agha

Open Systems Laboratory, Department of Computer Science,
University of Illinois at Urbana-Champaign,
Thomas Siebel Center for Computer Science,
201 N. Goodwin Avenue, Urbana, IL 61801, USA
{p-tosic, agha}@cs.uiuc.edu

Abstract. We study computational complexity of *counting* the *fixed point configurations (FPs)* in certain classes of graph automata viewed as discrete dynamical systems. We prove that both exact and approximate counting of FPs in *Sequential* and *Synchronous Dynamical Systems* (SDSs and SyDSs, respectively) are computationally intractable, even when each node is required to update according to a symmetric Boolean function. We also show that the problems of counting exactly the *garden of Eden configurations (GEs),* as well as all *transient configurations,* are in general intractable, as well. Moreover, exactly enumerating FPs or GEs remains hard even in some severely restricted cases, such as when the nodes of an SDS or SyDS use only two different symmetric Boolean update rules, and every node has a neighborhood size bounded by a small constant.

Keywords: Cellular and graph automata, sequential and synchronous dynamical systems, configuration space properties, computational complexity, #**P**-completeness

1 Introduction and Motivation

We study certain classes of *graph automata* that can be used as an abstraction of the classical networked distributed systems, as well as of various multi-agent systems and *ad hoc* networks, and as a theoretical model for the computer simulation of a broad variety of computational, physical, social, and socio-technical distributed infrastructures. In this and several related papers (see, e.g., [2,3,4,5,6,7,8,9,22,31,32]), the general approach has been to study mathematical and computational *configuration space properties* of such graph automata: what are the possible *global behaviors* of the entire system, given the simple local behaviors of its components, and the interaction pattern among these components.

We specifically focus in this paper on determining *how many* fixed point configurations such graph automata have, and *how hard* is the computational problem of *counting* (or *enumerating*) these configurations. In a nutshell, the contributions of this paper are as follows. We prove that both exact and approximate counting of the number of *fixed point* configurations in *Sequential* and *Synchronous Dynamical Systems* is computationally intractable, even when each node is required to update according to a symmetric Boolean function. We also show that the exact counting of the *"garden of Eden"* configurations, as well as of all *transient configurations,* is intractable, as well.

The rest of the paper is organized as follows. Section 2 is devoted to the necessary preliminaries about the models studied in this paper, namely, the sequential and synchronous dynamical systems. Section 3 summarizes our technical results, and reviews some literature closely related to our work. The original results are presented in Section 4. Finally, we conclude and outline some possible extensions in Section 5.

2 Preliminaries

In this section, we define and briefly discuss the discrete dynamical system models studied in this paper, and their configuration space properties. *Sequential Dynamical Systems* (henceforth referred to as SDSs) are proposed in [8,9,10] as an abstract model for computer simulations. This model has been successfully applied in the development of large-scale socio-economic simulation systems such as the *TRANSIMS* project at the Los Alamos National Laboratory [11].

A *Sequential Dynamical System (SDS)* $\mathcal{S}$ is a triple (G, F, Π), whose components are as follows:

1. $G(V, E)$ is an undirected graph without multi-edges or self-loops. G is referred to as the *underlying graph* of $\mathcal{S}$. We often use n to denote $|V|$ and m to denote $|E|$. The nodes of G are enumerated 1, 2, ..., n.
2. Each node is characterized by its *state.* The state of node i, denoted by s_i, takes on a value from some finite domain, $\mathcal{D}$. In this paper, we shall restrict $\mathcal{D}$ to $\{0, 1\}$. We use d_i to denote the degree of node i. Each node i is associated with a *node update rule* $f_i : \mathcal{D}^{d_i+1} \to \mathcal{D}$, for $1 \leq i \leq n$. We also refer to f_i as the *local transition function.* The inputs to f_i are the state of node i itself and the states of the neighbors of i. We use $F_{\mathcal{S}}$ to denote the *global map* of $\mathcal{S}$, obtained by appropriately composing together all the local update rules f_i, $i = 1, ..., n$.
3. Finally, Π is a permutation of $V = \{1, 2, \ldots, n\}$ specifying the sequential ordering in which the nodes update their states using their local transition functions. Alternatively, Π can be envisioned as a total order on the set of nodes. In particular, $F_{\mathcal{S}} = (f_{\Pi^{-1}(1)}, f_{\Pi^{-1}(2)}, \ldots, f_{\Pi^{-1}(n)})$.

The nodes are processed in the *sequential* order specified by the permutation Π. The processing associated with a node consists of computing the new value of its state according to the node's update function, and changing its state to this new value.

Most of the early work on sequential dynamical systems has focused primarily on the SDSs with *symmetric Boolean functions* as the node update rules

[2,3,4,5,7,8,9]. By *symmetric* is meant that the future state of a node does not depend on the order in which the input values of this node's neighbors are specified. Instead, the future state depends only on $\Sigma_{j \in N(i)} \, x_i$ (where $N(i)$ stands for the *extended neighborhood* of a given node, i, that includes the node i itself), i.e., on how many of the node's neighbors are currently in the state 1.

The assumption about *symmetric* Boolean functions can be easily relaxed to yield more general SDSs. We give special attention to the symmetry condition for two reasons. First, our computational complexity theoretic lower bounds for such SDSs imply stronger lower bounds for determining the corresponding configuration space properties[1] of the more general classes of graph automata and communicating finite state machines (CFSMs). Second, symmetry provides one possible way to model the "mean field effects" used in statistical physics and studies of other large-scale systems. A similar assumption is made in [12].

A *Synchronous Dynamical System* (SyDS) is an SDS *without* the node permutation. In an SyDS, at each discrete time step, all the nodes perfectly synchronously in parallel compute and update their state values. Thus, SyDSs are similar to the finite classical cellular automata (CA), except that in an SyDS the nodes may be interconnected in an arbitrary fashion, whereas in a classical cellular automaton the nodes are interconnected in a regular fashion (such as, e.g., a line, a rectangular grid, or a hypercube). Another difference is that, while in the classical CA all nodes update according to the same rule, in an SyDS different nodes, in general, use different local update rules.

2.1 SDS and SyDS Configuration Space Properties

A configuration of an SDS or SyDS $\mathcal{S} = (G, F, \Pi)$ is a vector $(b_1, b_2, \ldots, b_n) \in \mathcal{D}^n$. A configuration $\mathcal{C}$ can also be thought of as a function $\mathcal{C} : V \rightarrow \mathcal{D}$.

The function computed by an S(y)DS $\mathcal{S}$, denoted by $F_{\mathcal{S}}$, specifies for each configuration $\mathcal{C}$ the next configuration $\mathcal{C}'$ reached by $\mathcal{S}$ after carrying out the updates of the node states in the order given by Π. Thus, the function $F_{\mathcal{S}} : \mathcal{D}^n \rightarrow \mathcal{D}^n$ is a total function on the set of global configurations. This function therefore defines the dynamics of $\mathcal{S}$. We say that $\mathcal{S}$ moves from a configuration $\mathcal{C}$ to a configuration $F_{\mathcal{S}}(\mathcal{C})$ in a single transition step. Alternatively, we say that S(y)DS $\mathcal{S}$ moves from a configuration $\mathcal{C}$ at time t to a configuration $F_{\mathcal{S}}(\mathcal{C})$ at time $t+1$. Assuming that each node update function f_i is computable in time polynomial in the size of the description of $\mathcal{S}$, each transition step will also take polynomial time in the size of the S(y)DS's description.

The *configuration space* (also called *phase space*) $\mathcal{P}_{\mathcal{S}}$ of an SDS or SyDS $\mathcal{S}$ is a directed graph defined as follows. There is a vertex in $\mathcal{P}_{\mathcal{S}}$ for each global configuration of $\mathcal{S}$. There is a directed edge from a vertex representing configuration $\mathcal{C}$ to that representing configuration $\mathcal{C}'$ if $F_{\mathcal{S}}(\mathcal{C}) = \mathcal{C}'$. Since an SDS or SyDS is deterministic, each vertex in its phase space has the out-degree of 1. Since the domain $\mathcal{D}$ of state values is assumed finite, and the number of nodes in the S(y)DS is finite, the number of configurations in the phase space

[1] Configuration spaces of sequential and synchronous dynamical systems will be defined in subsection 2.1.

is also finite. If the size of the domain (that is, the number of possible states of each node) is $|\mathcal{D}|$, then the number of global configurations in $\mathcal{P}_{\mathcal{S}}$ is $|\mathcal{D}|^n$.

Definition 1. *Given two configurations* $\mathcal{C}'$ *and* $\mathcal{C}$ *of an SDS or SyDS* $\mathcal{S}$*, configuration* $\mathcal{C}'$ *is a* **predecessor** *of* $\mathcal{C}$ *if* $F_{\mathcal{S}}(\mathcal{C}') = \mathcal{C}$*, that is, if* $\mathcal{S}$ *moves from* $\mathcal{C}'$ *to* $\mathcal{C}$ *in one global transition step. Similarly,* $\mathcal{C}'$ *is an* **ancestor** *of* $\mathcal{C}$ *if there is a positive integer* t *such that* ${F_{\mathcal{S}}}^t(\mathcal{C}') = \mathcal{C}$*, that is, if* $\mathcal{S}$ *evolves from* $\mathcal{C}'$ *to* $\mathcal{C}$ *in one or more transitions.*

In particular, a predecessor of a given configuration is a special case of an ancestor.

Definition 2. *A configuration* $\mathcal{C}$ *of an S(y)DS* $\mathcal{S}$ *is a* **garden of Eden (GE)** *configuration if* $\mathcal{C}$ *has no predecessor.*

Definition 3. *A configuration* $\mathcal{C}$ *of an S(y)DS* $\mathcal{S}$ *is a* **fixed point (FP)** *configuration if* $F_{\mathcal{S}}(\mathcal{C}) = \mathcal{C}$*, that is, if the transition out of* $\mathcal{C}$ *is to* $\mathcal{C}$ *itself.*

Definition 4. *A configuration* $\mathcal{C}$ *of an S(y)DS is a* **cycle configuration (CC)** *if there exists an integer* $t \geq 2$ *such that*
(i) ${F_{\mathcal{S}}}^t(\mathcal{C}) = \mathcal{C}$*; and*
(ii) ${F_{\mathcal{S}}}^q(\mathcal{C}) \neq \mathcal{C}$*, for any integer* q*,* $0 < q < t$*.*
Integer t *above is called the* **period** *or* **length** *of the temporal cycle.*

Definition 5. *A configuration* $\mathcal{C}$ *of an S(y)DS is a* **transient configuration (TC)** *if* $\mathcal{C}$ *is neither a fixed point nor a cycle configuration.*

As the name suggests, transient configurations, unlike fixed points or cycle configurations, are never revisited. We note that a GE configuration is a special case of a transient configuration; a GE configuration is not reachable from *any* configuration including itself. We also remark that a node in the phase space may have multiple predecessors. This means that the time evolution map F of an SDS or SyDS is in general *not invertible* but is *contractive.* The existence of configurations with multiple predecessors also implies that certain configurations have no predecessors.

3 Summary of Results and Related Work

Given an SDS or SyDS $\mathcal{S}$, let $|\mathcal{S}|$ denote the size of the representation of $\mathcal{S}$. In general, this includes the number of nodes and edges, and the description of the local transition functions. When $\mathcal{D} = \{0, 1\}$ and the local transition functions are given as the truth tables, $|\mathcal{S}| = O(m+|T|n)$, where $|T|$ denotes the maximum size of a table, n is the number of nodes and m is the number of edges in the underlying graph. By *the size of a truth table* we shall throughout the paper mean, for simplicity, just the number of rows in this table. Thus, for a node v_i of degree d_i, the size of a truth table specifying an arbitrary Boolean function is $O(2^{d_i})$, and actually, for any (sufficiently big) positive integer d_i, most Boolean functions on $d_i + 1$ inputs cannot be encoded substantially more succinctly than via a truth table of size $\Theta(2^{d_i})$. In contrast, the size of an optimally succinct truth table fully specifying an arbitrary *symmetric* Boolean function is only $O(d_i)$.

Another, more common way of specifying the local transition functions is via *Boolean formulae.* We shall assume that f_i of *non-symmetric* SDSs and SyDSs considered in the sequel are indeed given as (reasonably succinct[2]) Boolean formulae of appropriately restricted kinds. It follows from the discussion above that, for symmetric Boolean update rules, the exact way these update rules are encoded in an S(y)DS is inconsequential, as long as this encoding is reasonably succinct. We shall also assume throughout that evaluating any local transition function f_i, given its input values, can be done in polynomial time.

We study herewith the problem of *counting* the fixed point (FP) configurations of Boolean SDSs and SyDSs. In particular, we prove the following results:

– counting FPs in the general Boolean (and, consequently, also any other finite domain) SDSs and SyDSs is **#P**-complete;
– this hardness result still holds when the node update rules of these SDSs and SyDSs are restricted to *symmetric Boolean functions;*
– moreover, the result remains valid even when only two different symmetric update rules are used, and when the maximum degree of each node in the underlying graph is a small constant.

3.1 Related Work

SDSs and SyDSs investigated in this paper are closely related to the *graph automata (GA)* models studied in [21,23] and the one-way cellular automata studied by Roka in [25]. In fact, the general finite-domain SyDSs exactly correspond to the graph automata of Nichitiu and Remila as defined in [23].

Barrett, Mortveit and Reidys [8,9,22] and Laubenbacher and Pareigis [20] investigate the mathematical properties of sequential dynamical systems. Barrett et al. study the computational complexity of several phase space questions for SDSs. These include the Reachability, Predecessor existence and Permutation existence problems [5,6]. Problems related to the existence of *garden of Eden* and *fixed point* configurations are studied in [7]. In particular, the basic **NP**-completeness results for the problems of FP, GE and non-unique predecessor existence in various restricted classes of Boolean S(y)DSs are proven in that paper. Algorithms for efficiently finding an FP in certain other restricted classes of S(y)DSs can be also found in [7]. Our results in Section 4 of this paper can be viewed as a natural partial extension of the work in [7]: instead of the appropriate *decision problems* about the fixed points and gardens of Eden in SDSs and SyDSs, we focus herein on studying the related *counting problems.*

Among various restricted classes of Boolean SDSs and SyDSs, those with the local update rules restricted to *symmetric functions* have received particular

[2] By *reasonably succinct* we mean, formulae whose sizes are not artificially blown up by, e.g., repeating the same clause(s) over and over again.

attention (e.g., [9,20,22]). Computational complexity of the reachability-related problems in the context of, among other restricted types, symmetric Boolean SDSs is investigated in [6]. We show in this paper that, in contrast to the computational feasibility of the problem of their reachability [6], the problem of *counting* stable configurations (FPs) in symmetric Boolean SDSs and SyDSs, under the usual assumptions in computational complexity theory, is intractable.

4 Counting Fixed Points of Boolean SDSs and SyDSs

The results in this section constitute an extension of the work presented in [7] and [6]. In [7], the computational complexity of *decision problems* related to the fixed point and the garden of Eden configurations in SDSs and SyDSs is studied. Once **NP**-completeness of these decision problems has been established [7], a natural further course of inquiry about the fundamental SDS phase space properties is to determine how hard it is to count *how many* FPs, GEs, and/or other configurations of interest an SDS of a given type may have.

Intuitively, one would expect, for example, that counting the fixed points of an arbitrary Boolean SDS or SyDS is no easier than counting the satisfying truth assignments of an arbitrary instance of the Satisfiability problem [13,24]. The intuitive notion of computational hardness of counting problems is formalized via the definition of the class **#P** (read: "sharp-P" or "number-P"). A counting problem Ψ belongs to the class **#P** if there exists a nondeterministic algorithm such that for each instance I of Ψ, the number of *nondeterministic guesses* that this algorithm makes that lead to acceptance equals the number of solutions of I_Ψ, and, in addition, it is required that the longest of these nondeterministic computations of the algorithm be polynomially bounded in the size of the description of I_Ψ. For an alternative but equivalent definition of class **#P** in terms of *polynomially balanced relations,* we refer the reader to [24].

The hardest problems in class **#P** are the **#P**-complete problems. A counting problem Ψ is **#P**-complete if and only if (i) it is in class **#P**, and (ii) *every* other problem in **#P** is efficiently reducible to Ψ. Thus, if we could solve *any* particular **#P**-complete problem in deterministic polynomial time, then *all* problems in class **#P** would be solvable in deterministic polynomial time. For more on the class **#P** we refer the interested reader to Chapter 18 of [24], and references therein.

In order to prove the intractability of counting FPs of Boolean SDSs and SyDSs, not any *polynomial time* reduction from a known **#P**-complete problem suffices. What is required is a kind of efficient reduction that *preserves the number of solutions.* We define this special kind of efficient reductions next:

Definition 6. *Given two decision problems Π and Π', a* PARSIMONIOUS REDUCTION *from Π to Π' is a polynomial-time transformation* g *that preserves the number of solutions; that is, if an instance* I *of Π has n_I solutions, then the corresponding instance* g(I) *of Π' also has $n_{g(I)} = n_I$ solutions.*

In practice, one often resorts to reductions that are "almost parsimonious", in a sense that, while they do not exactly preserve the number of solutions, n_I in the previous definition can be efficiently recovered from $n_{g(I)}$.

Definition 7. *Given two decision problems Π and Π', a* WEAKLY PARSIMONIOUS REDUCTION *from Π to Π' is a polynomial-time transformation* g *such that, if an instance* I *of Π has n_I solutions, and the corresponding instance* g(I) *of Π' has $n_{g(I)}$ solutions, then n_I can be computed from $n_{g(I)}$ in polynomial time.*

Our fundamental result on the hardness of counting the fixed point configurations of an arbitrary Boolean S(y)DS in the next subsection, as well as similar hardness results about *symmetric* Boolean S(y)DSs in the subsequent subsections, will follow from

Proposition 1. [24] *Given two decision problems Π and Π', if the corresponding counting problem $\#\Pi$ is* **#P***-hard and if there exists a weakly parsimonious reduction from Π to Π', then the counting problem $\#\Pi'$ is* **#P***-hard, as well.*

4.1 Counting Fixed Points of General Boolean SDSs and SyDSs

We shall use reductions from the known **#P**-complete problems, such as the counting version of *POSITIVE-EXACTLY-1-IN-3SAT (PE3SAT),* to the problems of counting FPs in certain classes of the SDS and SyDS automata. These reductions will formally establish the **#P**-completeness of those counting problems about SDSs and SyDSs. We now define the variants of *Satisfiability* [13,24] that we shall use in the sequel:

Definition 8. Exactly-one-in-three-satisfiability *(or* **E3SAT** *for short), is a version of* 3CNF-SAT *[13] such that, first, each clause in a given* 3CNF *formula contains exactly three literals, and, second, where a truth assignment is considered to satisfy the given* 3CNF *formula if and only if* exactly one of the three literals *is* true *in each clause.* Positive-exactly-one-in-three-satisfiability *(***PE3SAT***) is further restricted: no clause in the* 3CNF *formula is allowed to contain a negated literal.*

Consider the following reduction from the counting problem #PE3SAT to #FP-SDS, where #FP-SDS denotes the problem of counting the fixed point configurations of an arbitrary Boolean SDS.

Let an arbitrary instance I of PE3SAT be given. We construct the corresponding instance of an SDS $\mathcal{S} = \mathcal{S}(I)$ as follows. We remark that $\mathcal{S}$ in this subsection will be "nearly symmetric"; we will modify our construction to a fully symmetric Boolean SDS and SyDS in the next subsection.

Assume that I has n variables and m clauses. The underlying graph of $\mathcal{S}$ has a distinct node for each variable x_i, $1 \leq i \leq n$, and for each clause C_j, $1 \leq j \leq m$. The node labeled x_i is connected to the node labeled C_j if and only if, in the Boolean formula I, variable x_i appears in clause C_j. In addition, our graph has one additional node, labeled y, that is adjacent to nodes C_j for all indices $j = 1, ..., m$. Hence, each C_j has exactly four neighbors, and node y has m neighbors.

The node update functions of our SDS $\mathcal{S}$ are as follows:

- Each node C_j evaluates the logical AND of the current value of node y, the value evaluated by the PE3SAT function of the three variables $\{x_{j_1}, x_{j_2}, x_{j_3}\}$ that appear in the corresponding clause C_j of I, and the current value of itself; that is, the node update function C_j evaluates to 1 if and only if:

(i) *exactly* one out of the three neighboring nodes $x_{j_1}, x_{j_2}, x_{j_3}$ currently holds the value 1; and

(ii) the node y currently holds the value 1; and

(iii) the current value of C_j itself is 1.

- The "special" node y evaluates the AND of its own current value and the entire set of current values held in the clause nodes C_j, $1 \leq j \leq m$. This will enable us to argue that the node y, in effect, evaluates the Boolean formula for the specified truth assignment $\{x_1, ..., x_n\}$, provided that the initial value stored in node y is $y^{t=0} = 1$, and, likewise, that $C_j^{t=0} = 1$, for all j, $1 \leq j \leq m$.

- Each node x_i evaluates the logical AND of itself and the current values stored in the clause nodes $C_{j(i)}$ such that, in the original formula I , variable x_i appears in clause $C_{j(i)}$.

The order of the node updates is $(C_1, ..., C_m, y, x_1, ..., x_n)$.

Since $\mathcal{S}$ has $n+m+1$ nodes, the corresponding configuration space will have 2^{n+m+1} configurations.

We now claim that the reduction from #PE3SAT to #FP-SDS based on the above SDS construction from an instance I of PE3SAT is weakly parsimonious; it will then immediately follow that

Theorem 1. *The problem of counting the fixed points of an arbitrary Boolean SDS (and therefore also of any more general finite domain SDS), is #**P**-complete.*

Remark: Similarly, by a straight-forward modification of the given SDS construction, one can establish that the #FP-SyDS problem for the general Boolean (and therefore any finite domain) SyDSs is #**P**-complete, as well.

Detailed proof that establishes that the given reduction from #PE3SAT to #FP-SDS is, indeed, weakly parsimonious can be found in our electronic technical report [30].

4.2 Counting Fixed Points of Symmetric Boolean SDSs and SyDSs

The hardness results for symmetric Boolean SDSs and SyDSs will be based on an appropriate reduction from the PE2-IN-3SAT problem. We define PE2-IN-3SAT similarly to how we defined PE3SAT, only this time we require each clause to have *exactly two* true variables (rather than *exactly one* as was the case in PE3SAT). We observe that, since PE3SAT is **NP**-complete, so is PE2-IN-3SAT, and moreover the #**P**-completeness of the counting version of the former, denoted #PE3SAT, also implies the **#P**-completeness of the counting version of the latter, #PE2-IN-3SAT.

Let an instance I of PE2-IN-3SAT be given. Assume that there are n Boolean variables, denoted $x_1, ..., x_n$, and m clauses, $C_1, ..., C_m$, in I. We recall that each clause C_j contains *exactly* three unnegated variables, $x_{j_1}, x_{j_2}, x_{j_3}$. An instance I is a *positive* or *satisfying* instance of PE2-IN-3SAT if and only if there exists a truth assignment to $x_1, ..., x_n$ such that *exactly* two variables in each clause are true.

We now prove that counting FPs of a symmetric Boolean SyDS or SDS is **#P**-complete. We recall that fixed points are invariant under the node update ordering; that is, regardless of whether the nodes update synchronously in parallel, or sequentially according to an arbitrary ordering Π, the fixed points of the underlying dynamical system as specified by its graph and the local node update functions remain the same (see [22] for a proof).

Theorem 2. *The problem of counting fixed points of a symmetric Boolean Synchronous Dynamical System, abbreviated as* #FP-SYM-SYDS*, is* **#P***-complete.*

Proof sketch: To show **#P**-hardness, we reduce the problem of counting the satisfying truth assignments of an instance of PE2-IN-3SAT to counting the fixed points of a symmetric Boolean SyDS. We construct an SyDS, $\mathcal{S}$, from an instance of PE2-IN-3SAT as follows. We let the underlying graph of $\mathcal{S}$ have $m+n+1$ vertices: one for each variable, one for each clause, and one additional vertex, denoted by y. The edges of the underlying SyDS graph are as follows: each vertex node x_i is adjacent to those and only those clause nodes $C_{j(i)}$ such that the corresponding variable x_i appears in the corresponding clause $C_{j(i)}$ of formula I. Let each clause node C_j be adjacent to all other clause nodes C_k (for all k, $1 \leq k \leq m$, $k \neq j$), to the special node y, and to the three nodes $x_{j_1}, x_{j_2}, x_{j_3}$ corresponding to the Boolean variables that appear in the clause C_j in the formula; and, finally, by symmetry, let the node y be adjacent to all the clause nodes C_k.

We define the node update functions as follows:

$x_i^{t+1} = x_i^t \wedge (\wedge_{j(i)} C_{j(i)}^t)$;
$C_j^{t+1} = \text{ALL-BUT-ONE} \ \{x_{j_1}^t, x_{j_2}^t, x_{j_3}^t, C_1^t, ..., C_m^t, y^t\}$;
$y^{t+1} = y^t \wedge (\wedge_{j=1}^m C_j^t)$,

where the Boolean function ALL-BUT-ONE $(z_1, ..., z_q) = 1$ if and only if *exactly* one of its inputs z_l $(1 \leq l \leq q)$ is 0, and all the rest are 1*s*.

We now claim that the constructed synchronous dynamical system has $|T|+2$ fixed points if and only if the corresponding instance of PE2-IN-3SAT has $|T|$ satisfying truth assignments. That is, the given reduction from #PE2-IN-3SAT to #FP-SYM-SYDS is, indeed, weakly parsimonious. We omit details due to space constraints, and refer the interested reader to [30].

By the aforementioned invariance of fixed points with respect to the node update ordering, the next result on the hardness of counting FPs in symmetric Boolean SDSs is not at all surprising.

Theorem 3. *The problem of counting fixed point configurations of symmetric Boolean SDSs (abbreviated as* #FP-Sym-SDS*) is* #**P***-complete.*

Proof sketch: In order to prove the theorem explicitly, as well as establish several other complexity-theoretic counting results for symmetric Boolean SDSs, we consider the following construction of an SDS $\mathcal{S}$' from the SyDS $\mathcal{S}$ used in the proof of the previous theorem.

– The underlying graph and the local node updating functions are as in the SyDS construction in the previous theorem.
– Let the node ordering be given by $\Pi = (y, C_1, ..., C_m, x_1, ..., x_n)$. Thus, $y^{t+1} = y^t \wedge (\wedge_{j=1}^{m} C_j^t)$, $C_j^{t+1} = \text{All-But-One } \{y^{t+1}, C_1^{t+1}, ..., C_{j-1}^{t+1}, C_j^t, C_{j+1}^t, ..., C_m^t, x_{j_1}^t, x_{j_2}^t, x_{j_3}^t\}$, and, for any i such that $1 \leq i \leq n$, $x_i^{t+1} = x_i^t \wedge (\wedge_{j(i)} C_{j(i)}^{t+1})$, where, as before, $C_{j(i)}$ denotes precisely those clause nodes that correspond to the clauses in the original Boolean formula in which the variable x_i appears.

Due to the space constraints, we will only summarize what the configuration space of SDS $\mathcal{S}$' looks like. Since there are $n+m+1$ nodes, there are 2^{n+m+1} global configurations in total. Among these, by virtue of Theorem 2 there are precisely $|T|+2$ fixed points, where $|T|$ is the number of solutions of the corresponding PE2-in-3SAT formula I. The number of these solutions is in the range $\{0, 1, ..., 2^n\}$. All of the $|T|$ fixed points corresponding to the solutions of I, as well as the fixed point $(y = 0, C = 1^m, x = 1^n)$, are *isolated fixed points,* in a sense that they do not have any in-coming transients. The configuration 0^{n+m+1} is *the sink* for $\mathcal{S}$', since all transient chains eventually converge to 0^{n+m+1}. All the remaining configurations are transient states, and, in particular, $\mathcal{S}$' does not have any temporal cycles. Furthermore, it can be readily shown that all transient chains are very short, since every transient configuration is either a garden of Eden, or its predecessor is a garden of Eden; this is immediate from the fact that every convergence to the sink 0^{n+m+1} takes at most two steps[3].

In summary, enumerating the fixed points of Symmetric Boolean SDSs and SyDSs *exactly* is #**P**-complete; moreover, it follows from the results in [26] that *approximating* the number of FPs to within $2^{|V|^{1-\epsilon}}$ is **NP**-hard, for any $\epsilon > 0$. Similarly, it can be shown from the given construction of $\mathcal{S}$' that counting *exactly* all TCs or all GEs of a Symmetric Boolean S(y)DS is #**P**-complete, as well. The complexity of counting GEs and TCs in symmetric S(y)DSs *approximately,* however, cannot be deduced from our constructions in this subsection and, to the best of our knowledge, is still open.

[3] For a much more detailed argument, and a rigorous justification of all other assertions that we make in this proof sketch, we refer the reader to the online publication [30].

4.3 Counting in Symmetric Boolean S(y)DSs with Bounded Node Degrees

The constructions of symmetric Boolean SDSs and SyDSs in the previous subsection include a "central control" node, y, that has an unbounded degree. Also, the clause nodes C_j in Theorems 2 and 3 are forming a clique, thus also being of unbounded degrees. We now transform the SyDS and SDS constructions from the previous subsection so that the node y is eliminated altogether, and so that each clause node C_j has only $O(1)$ neighbors. This reduction in the maximum node degree allowed will be done at the expense of doubling the number of the clause nodes, so that the resulting symmetric Boolean S(y)DS has $n+2m$ nodes in total, where, as before, n is the number of variables and m is the number of clauses in the original 3CNF Boolean formula.

Indeed, let's eliminate the node y in the constructions in Theorems 2 and 3, and, instead, for each clause node C_j, introduce its *clone,* C^c_j. Let's now connect each node C_j to its clone C^c_j and also to the clone of the successor clause node, $C^c_{j+1\ (mod\ m)}$. We also delete all the edges among the original clause nodes C_j. Thus, each original clause node C_j will now have *exactly five* neighbors: the three variable nodes, x_{j_1}, x_{j_2} and x_{j_3}, and the two "cloned" clause nodes, C^c_j and $C^c_{j+1\ (mod\ m)}$.

We will also assume that the 3CNF SAT instance is from a restricted class of *monotone* 3CNF formulae where each variable x_i appears in at most five clauses. This restriction does not affect the **#P**-completeness of the underlying counting problem. In fact, counting satisfying truth assignments of the *monotone* 2CNF formulae, abbreviated as MON-2CNF-SAT, is **#P**-complete even when each variable appears in at most five clauses [33]. Each of these MON-2CNF formulae can be readily converted into a special case of the MAJORITY-MON-3CNF formulae, in which a clause is satisfied if and only if *at least two out of three* unnegated variables (that is, their *majority*) appearing in this clause are true.

Since this, restricted type of the counting problem #MAJORITY-MON-3CNF is **#P**-complete, even when no variable occurs in more than five different clauses, and since the general #MAJORITY-MON-3CNF is clearly in the class **#P**, we conclude that the general problem of counting the satisfying assignments to a monotone 3CNF formula according to the MAJORITY rule is **#P**-complete *even when no variable appears in more than five different clauses,* as well.

We now turn to the construction of a *bounded-degree symmetric Boolean SDS or SyDS* from an instance of the MAJORITY-MON-3CNF formula.

Let the variable nodes in the S(y)DS constructed from such a 3CNF formula with restricted number of appearances of each variable update according to the Boolean *AND* rule on (at most) six inputs. Each variable node, as before, is connected to those, and only those, clause nodes such that the corresponding variable in the MAJORITY-MON-3CNF formula appears in the corresponding clause. Hence, each of these variable nodes will have at most five neighbors.

Recall that each clause node C_j is connected to the two *cloned clause nodes* C^c_j and $C^c_{j+1\ (mod\ m)}$. Since each of the original clause nodes has exactly five neighbors in total, the local update rule at each such node needs to be a symmetric Boolean function of six inputs. So, we let each node C_j update its state according to the "AT LEAST FIVE OUT OF SIX" rule.

Furthermore, let's also connect all the cloned clause nodes C_j^c into a ring, so that the only neighbors of C_j^c (beside C_j and $C_{j-1\ (mod\ m)}$) are $C^c_{j-1\ (mod\ m)}$ and $C^c_{j+1\ (mod\ m)}$. Finally, let each of the cloned clause nodes C_j^c update according to the Boolean *AND* function of its five inputs (the states of its four neighbors plus the current state of itself).

If a single cloned node $C^c_{j_\star}$ at any time step updates to 0, this node will eventually force all the remaining cloned clause nodes C_j^c, and consequently also all the original clause nodes C_j, to become 0s, as well. Similarly, if any of the original clause nodes $C_{j_\star}$ ever evaluates to 0, this will first cause its clone, $C^c_{j_\star}$, to evaluate to 0 (and stay at 0 thereafter), and that will, in turn, subsequently force all the other cloned clause nodes to become 0s. Since each of the original clause nodes C_j will then have at least two neighbors stuck in the state 0, that will also ensure that eventually $C_j = 0$ for all $j = 1, ..., m$. Therefore, if any of the clauses in the original formula is not satisfied, the corresponding S(y)DS will quickly converge to the sink fixed point 0^{n+2m}.

In contrast, if initially all $C_j^c = C_j = 1$, and the original Boolean formula is satisfied, then all the cloned clause nodes will remain at 1, and the corresponding global S(y)DS configuration is a fixed point corresponding to a satisfying truth assignment of the original Boolean formula.

To summarize, the following strengthening of the results in the previous subsection holds:

Theorem 4. *The problem of counting the fixed points of a Symmetric Boolean SDS or SyDS is* #**P**-*complete, even when each node in the underlying graph of such an S(y)DS is of a degree* $d_i \leq 5$, *and the nodes of that S(y)DS use only two different symmetric update rules.*

In fact, the upper bound on the maximum degree of any node in a symmetric Boolean S(y)DS can be further reduced: the problem of exactly counting FPs in such SDSs and SyDSs remains #**P**-complete even when each node's degree is required not to exceed 4 (instead of 5 as in the theorem above). A weakly parsimonious reduction directly from MON-2CNF SAT can be used to establish that result. We leave out the details due to space constraints. Insofar as the symmetric S(y)DSs with the maximum node degree not greater than 3 are concerned, counting FPs in their configuration spaces remains an open problem; our attempts to obtain the #**P**-completeness by an appropriate modification of the constructions used in this paper have turned out to be futile. We still suspect that this hardness nonetheless holds, but perhaps a different approach is needed. We leave resolving this conjecture about the symmetric SDSs and SyDSs whose underlying graphs have node degrees not exceeding 3 for the future work.

5 Conclusions and Future Directions

We study in this paper certain types of graph automata viewed as abstract discrete-time, discrete-state dynamical systems. We specifically focus on the problem of counting how many "fixed point" configurations such dynamical systems have in their configuration spaces, when each of their nodes has only two distinct states, and updates according to some simple Boolean function of the states of its neighboring nodes. Concretely, we establish that counting these fixed points in Sequential and Synchronous Dynamical Systems is **#P**-complete, even when the following constraints on the structure of an SDS or SyDS *simultaneously* hold:

– each local update rule is required to be a *symmetric* Boolean function; and
– the underlying graph of this SDS or SyDS is *sparse* in a very strong sense: all the node degrees are *uniformly bounded* by a small constant; and
– the nodes of this SDS or SyDS use only two different symmetric Boolean update rules.

As for our ongoing and future work, there are several directions along which we can strengthen the results presented in this paper, and extend them to similar results about counting other types of configurations and other emerging structures in discrete dynamical systems such as SDSs, Hopfield Networks or classical Cellular Automata. One concrete open problem is the complexity of counting FPs in symmetric Boolean SDSs and SyDSs when no node degree exceeds 3. We have been also studying the complexity of counting in various restricted types of Boolean S(y)DSs when it comes to the *backward dynamics* problems, such as those related to the number of predecessors or the number of all ancestors of an arbitrary configuration. We will report new results in that context elsewhere.

In summary, the formal discrete dynamical systems concepts, paradigms and methodology provide a rich arsenal with which to tackle, in an abstract yet mathematically elegant setting, many fundamental problems about large-scale distributed computational and communication infrastructures and multi-agent systems. Our results in this paper are an example of how the paradigms from nonlinear complex dynamics, coupled with the computational complexity tools, can provide insights into which aspects of the large-scale distributed systems' global behaviors can be reasonably expected to be feasible to predict in practice, and which ones cannot. In particular, it then follows that, in case of the latter, and under the usual assumptions in computational complexity theory, there is no "short-cut" to a step-by-step computer simulation.

Acknowledgments. The first author expresses his sincere gratitude to Harry Hunt (SUNY-Albany), Michael Loui (University of Illinois) and Madhav Marathe (Los Alamos National Laboratory) for many useful discussions and suggestions on various matters related to the problems studied in this paper.

References

1. S. Arora, Y. Rabani and U. Vazirani. "Simulating quadratic dynamical systems is **PSPACE**-complete," *Proc 26th. Annual ACM Symposium on the Theory of Computing* (STOC), pp. 459-467, Montreal, Canada, May 1994

2. C. Barrett, B. Bush, S. Kopp, H. Mortveit and C. Reidys. "Sequential Dynamical Systems and Applications to Simulations", Technical Report, Los Alamos National Laboratory, September 1999
3. C. Barrett, M. Marathe, H. Mortveit, C. Reidys, J. Smith, S.S. Ravi. "Adhop-NET: Advanced Simulation-based Analysis of Ad-Hoc Networks", Los Alamos Unclassified Internal Report, 2000
4. C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Dichotomy Results for Sequential Dynamical Systems", Los Alamos National Laboratory Report, LA-UR-00-5984, 2000
5. C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Predecessor and Permutation Existence Problems for Sequential Dynamical Systems", Los Alamos National Laboratory Report, LA-UR-01-668, 2001
6. C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Reachability problems for sequential dynamical systems with threshold functions", *Theoretical Comp. Sci.,* vol. 295, issues 1-3, pp. 41-64, February 2003
7. C. L. Barrett, H. B. Hunt, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns, P. T. Tosic. "Gardens of Eden and Fixed Points in Sequential Dynamical Systems", Proc. AA (DM-CCG), in *Discrete Mathematics & Theoretical Computer Science* (spec. ed.), July 2001
8. C. Barrett, H. Mortveit, and C. Reidys. "Elements of a theory of simulation II: sequential dynamical systems" *Applied Math. and Comput.*, vol 107/2-3, pp. 121-136, 2000
9. C. Barrett, H. Mortveit and C. Reidys. "Elements of a theory of computer simulation III: equivalence of SDS", *Applied Math. and Comput.,* vol. 122, pp. 325-340, 2001
10. C. Barrett and C. Reidys. "Elements of a theory of computer simulation I: sequential CA over random graphs" *Applied Mathematics and Computation*, vol. 98, pp. 241-259, 1999
11. R.J. Beckman, et. al. "TRANSIMS: Case Study", Dallas Ft-Worth. Los Alamos National Laboratory, LA UR 97-4502, 1999
12. S. Buss, C. Papadimitriou and J. Tsitsiklis. "On the predictability of coupled automata: An allegory about Chaos" *Complex Systems,* vol. 1(5), pp. 525-539, 1991
13. M. R. Garey and D. S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP-completeness",* W. H. Freeman and Co., San Francisco, California, 1979
14. M. Garzon. *"Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks",* Springer, 1995
15. E. Goles, S. Martinez. "Neural and Automata Networks: Dynamical behavior and Applications", Math. and Its Applications series (vol. 58), Kluwer, 1990
16. E. Goles, S. Martinez (eds.) "Cellular Automata and Complex Systems", Nonlinear Phenomena and Complex Systems series, Kluwer, 1999
17. F. Green. "NP-Complete Problems in Cellular Automata", *Complex Systems,* vol. 1, No. 3, pp. 453-474, 1987
18. B. Huberman, N. Glance. "Evolutionary games and computer simulations", *Proc. Nat'l Academy of Sciences,* 1999
19. H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Designing Approximation Schemes using L-reductions", *Proc. 14th Conference on Foundations of Software Technology and Theoretical Computer Science* (FST&TCS'94), Madras, India, Dec. 1994, pp. 342-353

20. R. Laubenbacher and B. Pareigis. "Finite Dynamical Systems", Technical report, Department of Mathematical Sciences, New Mexico State University, Las Cruces, 2000
21. B. Martin. "A Geometrical Hierarchy of Graphs via Cellular Automata", Proc. MFCS'98 Satellite Workshop on Cellular Automata, Brno, Czech Republic, August 1998
22. H. Mortveit, C. Reidys. "Discrete, sequential dynamical systems", *Discrete Mathematics*, vol. 226, pp. 281-295, 2001
23. C. Nichitiu and E. Remila. "Simulations of Graph Automata", Proc. MFCS'98 Satellite Workshop on Cellular Automata, Brno, Czech Republic, August 1998
24. C. Papadimitriou. *"Computational Complexity" ,* Addison-Wesley, Reading, Massachusetts, 1994
25. Z. Roka. "One-way cellular automata on Cayley graphs", *Theoretical Computer Science,* 132(1-2), pp. 259-290, September 1994
26. D. Roth. "On the Hardness of Approximate Reasoning", *Artificial Intelligence,* vol. 82, pp. 273-302, 1996
27. K. Sutner. "On the computational complexity of finite cellular automata", *Journal of Computer and System Sciences,* vol. 50(1), pp. 87-97, February 1995
28. K. Sutner. "Computation theory of cellular automata", *MFCS'98 Satellite Workshop on Cellular Automata* Bruno, Czech Republic, August 1998
29. C. Schittenkopf, G. Deco and W. Brauer. "Finite automata-models for the investigation of dynamical systems" *Information Processing Letters,* vol. 63(3), pp. 137-141, August 1997
30. P. Tosic. "On Complexity of Counting Fixed Point Configurations in Certain Classes of Graph Automata", *Electronic Colloquium on Computational Complexity,* ECCC- TR05-051 (revision 1), April 2005
31. P. Tosic, G. Agha. "Concurrency vs. Sequential Interleavings in 1-D Threshold Cellular Automata", APDCM Workshop within IPDPS'04, Santa Fe, New Mexico, USA, April 2004 (in Proc. IEEE-IPDPS '04)
32. P. Tosic, G. Agha. "Characterizing Configuration Spaces of Simple Threshold Cellular Automata", *Proc. 6th 6th Int'l Conference on Cellular Automata for Research and Industry (ACRI'04),* Amsterdam, The Netherlands, Oct. 25-28, 2004, Springer-Verlag LNCS series, vol. 3305
33. S. Vadhan. "The Complexity of Counting in Sparse, Regular and Planar Graphs", *SIAM J. Computing,* vol. 31 (2), pp. 398-427, 2001
34. L. Valiant. "The Complexity of Computing the Permanent", *Theoretical Comp. Sci.,* vol. 8, pp. 189-201, 1979
35. L. Valiant. "The Complexity of Enumeration and Reliability Problems", *SIAM J. Computing,* vol. 8 (3), pp. 410 - 421, 1979

A New Sibling of BQP

Tereza Tušarová

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
`tusarova@qubit.cz`

Abstract. We describe a quantum complexity class which is contained in AWPP. This class has a compact and simple mathematical definition, involving only polynomial-time computable functions and a unitarity condition. It contains both Deutsch-Jozsa's and Shor's algorithm, while it's relation to BQP is unknown. This shows that in the complexity class hierarchy, BQP is not an extraordinary isolated island, but has "siblings" which as well can solve prime-factorization.

1 Introduction

Quantum computing used to be a very popular discipline ranging from pure theoretical questions, concerning the complexity of the quantum polynomial-time class BQP, to practical concerns such as how to build a quantum computer. Nowadays, it seems a little bit that scientists are loosing their interest. Is it because all "easy" questions have been answered and what remains is too hard or uninteresting? We can find many of unanswered questions in the family of complexity classes. For example, we know that BPP$\subseteq$BQP$\subseteq$AWPP. But questions whether is BQP equal to its "father" AWPP or its "son" BPP have not been answered up to now. Here, we do not answer them either. Instead, we introduce a nontrivial "brother", which we call MQ^2. Surprisingly, this brother can also factorize long integers in polynomial time. Moreover, it has a very compact mathematical definition, which does not explicitly involve any physics.

The paper is organized as follows. In section 2, we introduce the necessary notation and definitions. In section 3, we briefly review classical polynomial-time classes definition. In the following section, we define the class MQ^2 itself. Fifth sections demonstrates two quantum algorithms, Shor's and Deutsch-Jozsa's, in class MQ^2.

2 Definitions and Notation

In this text, we frequently encounter matrices. All matrices here will have square shape. We will denote the element of a matrix M in $i-$th row and $j-$th column as $\langle i|M|j\rangle$, in accordance with the usual notation used in quantum computing, because we think it makes the text more readable then writing $M_{i,j}$. The range of the indices will be $0..n-1$ where n is the number of columns or rows.

We will define our new complexity class with use of matrices. To make the class uniform, we will want that all the matrices are constructed by the same algorithm. We formalize this in the following definition.

Definition 1 (Poly-computable matrix family). *A sequence of matrices* T_1, $T_2, \ldots$ *is called a poly-computable matrix family if there exists a function* $f : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ *computable in time polynomial in the length of all the arguments such that*

$$\forall i, j, n : f(i, j, n) = \langle j|T_n|i\rangle.$$

Sometimes, we will drop the index n when it will be clear from the context.

To show how the classical polynomial-time classes definitions correspond to our definition, we will start with a Turing machine and express its transition function as a transition matrix. A transition matrix is in fact a linear operator defined in a vector space spanned by configurations playing role of base vectors.

Definition 2 (Configuration of a Turing machine). *A configuration of a Turing machine is an ordered triple consisting of:*

-the contents of the tape
-the current state
-the position of the head [1]

We emphasize here that the configuration as defined above contains also the content of the tape, which is not true for configurations as defined elsewhere. Without loss of generality, we will further assume that configurations are indexed and denoted by their indices. A special position among the configurations has the family of initial configuration $I(x)_n$, which we allow to be dependent on x, but require to be computable in polynomial time. Again, we will drop the index n when it will be clear which member of the family we mean. We will also need to be able to recognize accepting configurations. For this purpose, we will have a function $a(x, c)$, computable in polynomial time, where the first argument will be the input for the algorithm and c is a configuration. The function $a(x, c)$ will return 1 iff the configuration c is accepting (possibly depending on x) and 0 otherwise.

Now we are ready to jump to the notion of transition matrix. If a configuration c_1 leads to another configuration c_2 in the next step with probability p, there is p on the position $\langle c_2|T|c_1\rangle$, otherwise there is zero. Because the tapes are of unbounded size, so is the matrix. However, if we know that the time complexity of a Turing machine is $T(n)$, we may for fixed n have a finite matrix cutting the tapes at the distance $T(n)$ from the initial position on both sides. The size of the matrix for inputs of length n is then $2^{O(T(n))} \times 2^{O(T(n))}$. For a probabilistic Turing machine, the transition matrix is stochastic, e.g. every row sums up to 1. Transition matrices naturally form a poly-computable matrix family, since for each pair c_1, c_2, the probability of going from one to another can be read

[1] We assume without loss of generality that the machine has only one tape.

from the description of the underlying probabilistic Turing machine, which is a finite object[2].

3 Traditional Complexity Classes

We will now briefly review classical complexity classes definitions. The common definitions of P, BPP, NP and PP involve a probabilistic Turing machine and look at the accepting probability for each input[3].

One step of a probabilistic Turing machine corresponds to multiplying the transition matrix with a vector representing the current configuration. Thus, instead of saying "the probability of accepting on a configuration $I(x)$ after S steps is p", we may equivalently say "$\sum_{c:a(c,x)=1} \langle c|T^S|I(x)\rangle = p$". We will use this observation in the following definitions.

Definition 3 (Polynomial time classes in matrix notation). *A language L is in class C if there exists a polynomial $p(n)$ and a probabilistic Turing machine M with transition matrix family T_i and functions $I(x)$, $a(x,c)$ computable in polynomial time, such that for all n and for all x of length n:*

complexity class C	*P*	*NP*	*PP*	*BPP*
For $x \in L$, $\sum_{c:a(c,x)=1} \langle c|T_n^{p(n)}|I(x)\rangle$	$= 1$	> 0	$> \frac{1}{2}$	$\geq \frac{2}{3}$
For $x \notin L$, $\sum_{c:a(c,x)=1} \langle c|T_n^{p(n)}|I(x)\rangle$	$= 0$	$= 0$	$\leq \frac{1}{2}$	$\leq \frac{1}{3}$

where exactly one of the columns applies.

s In the same manner, the quantum class BQP is commonly defined:

Definition 4 (BQP). *A language L is in class BQP if there exists a polynomial $p(n)$ and a quantum Turing machine M with transition matrix family T_i of unitary matrices and functions $I(x)$, $a(x,c)$ computable in polynomial time, such that*

$$\text{For } x \in L\text{: } \left|\sum_{c:a(c,x)=1} \langle c|T^{p(n)}|I(x)\rangle\right|^2 \geq \frac{2}{3}$$
$$\text{For } x \notin L\text{: } \left|\sum_{c:a(c,x)=1} \langle c|T^{p(n)}|I(x)\rangle\right|^2 \leq \frac{1}{3}$$

We emphasize here that there are exactly two points in which Definition 4 and the Definition of BPP in Definition 3 differ: First, in Definition 3 we have stochastic matrices while in Definition 4 we have unitary matrices. Second, in

[2] It should be pointed out that for most poly-computable stochastic matrix families, no corresponding probabilistic Turing machine exists. The reason is that each probabilistic Turing machine has finite description of a bounded size, independent on the length of input, while in our definition 1, we allowed function f to arbitrarily depend on n.

[3] Obviously, the class P can be viewed as a special case of a probabilistic class, with probabilities either one or zero.

Definition 3 we are looking at the value of $\sum_{c:a(c,x)=1} \langle c|T^{p(n)}|I(x)\rangle$, while in Definition 4 we look at the square norm of this value. However, the latter can be avoided, since we can equivalently use the square norm in the Definition 3 of BPP. Thus, the only remaining difference between the two classes is the type of matrices used.

Now we will define the class MQ^2 itself.

4 The Sibling

Now we will somehow alter the Definition 4 of class BQP to get a new class MQ^2. In practice, there is often the situation that we do not know what particular output we are looking for, but have some property in mind we would like the outputs to have. For that purpose, we will use a function $a(x, y)$ which decides for a given x at the input, whether y is one of the accepting outputs we are looking for or not.[4]

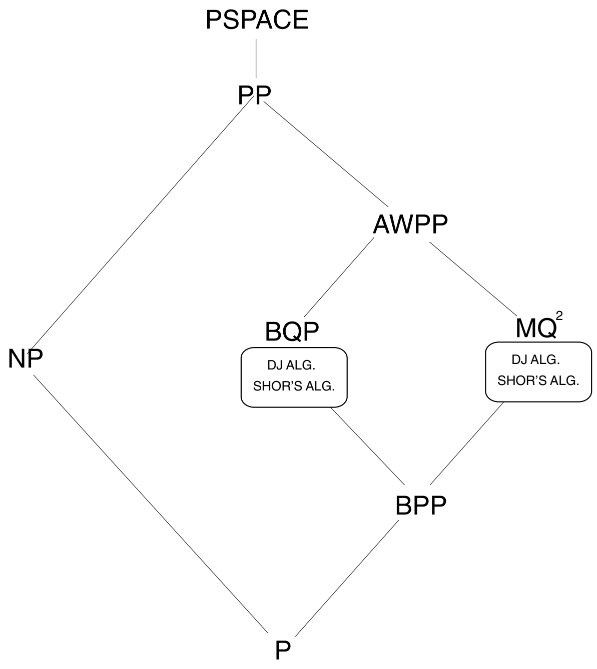


Fig. 1. Hierarchy of classes including MQ^2. For each pair connected by a line, the class that stays upper contains the lower one.

[4] In class BQP, it does not matter whether we have one or more accepting configuration [1]. Nevertheless here, in class MQ^2 we can not use the same trick, since generally we do not have a function computing the entries of the inverse of the matrix used.

Definition 5 (MQ^2). *A language L is in class MQ^2 iff there exists a unitary, poly-computable matrix family T_i, a poly-computable vector family $I(x)$, and a function $a(x,c)$ computable in polynomial time such that*

For $x \in L$, $\sum_{c:a(c,x)=1} \left|\langle c|T^2|I(x)\rangle\right|^2 \geq \frac{2}{3}$

For $x \notin L$, $\sum_{c:a(c,x)=1} \left|\langle c|T^2|I(x)\rangle\right|^2 \leq \frac{1}{3}$

It was shown in [4] that MQ^2 is contained in AWPP.
The resulting hierarchy is visualized in Figure 1.

5 Expressing Quantum Algorithms

In this section, we will suggest how can the class MQ^2 couple with the Deutsch-Jozsa's and Shor's algorithm. For complete proofs, see [4].

At first, we show that class MQ^2 captures Deutsch-Jozsa's problem. For this problem, see [2]. In the Deutsch-Jozsa problem, a quantum oracle is used. That is a diagonal quantum gate, or in other words a diagonal unitary matrix, realizing the transformation $x \to (-1)^{f(x)}$. Here, our matrix will be a product of a poly-computable matrix and this oracle. The result if then poly-computable too.

Theorem 1. *The class MQ^2 solves the Deutsch-Jozsa problem.*

Proof sketch. We will mimic the circuit from Deutsch-Jozsa's algorithm (see Figure 2 a)) by two copies of a poly-computable matrix family T (see Figure 2 b)). For a fixed n, the matrix T will be a product of the oracle and H^n. We may add another matrix for the f function to the front, since it will only add the number $(-1)^{f(0)}$ to the global phase and thus will not change the result. Formally, we define a matrix T as

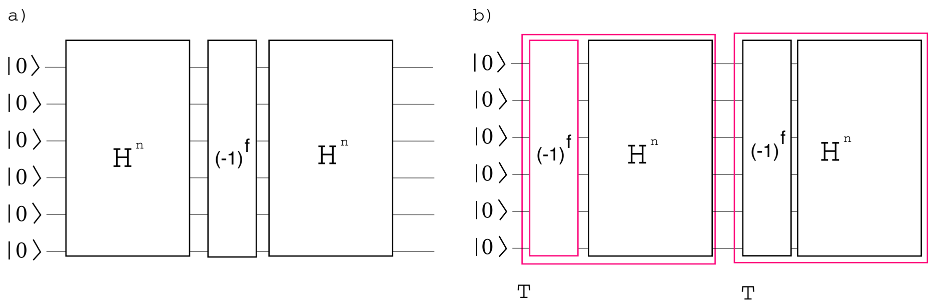


Fig. 2. The trick used to fit Deutsch-Jozsa's algorithm into the MQ^2 class. In a), there is the original setup, in b) there is the one we use, which gives the same result up to a global phase.

$$\langle y|T|x\rangle \equiv (-1)^{f(x)}\langle y|H^n|x\rangle = \frac{1}{\sqrt{2^n}}(-1)^{f(x)}(-1)^{\sum_i x_i y_i \bmod 2} \tag{1}$$

which is obviously computable in poly-time and unitary. We define $c_i(x) = 0^n$ and $c_A(x) = 0^n$ for x of length n. Then we have

$$\left|\langle 0^n|T^2|0^n\rangle\right|^2 = \left|\sum_k \langle 0^n|T|k\rangle\langle k|T|0^n\rangle\right|^2 =$$

$$= \left|\sum_k (-1)^{f(k)}\frac{1}{\sqrt{2^n}}(-1)^{\sum_i k_i 0^n{}_i \bmod 2}\frac{1}{\sqrt{2^n}}(-1)^{f(0^n)}(-1)^{\sum_i 0^n{}_i k_i \bmod 2}\right|^2 =$$

$$= \left|\sum_k \frac{1}{2^n}(-1)^{f(k)}(-1)^{2\sum_i k_i 0^n{}_i \bmod 2}\right|^2 = \frac{1}{2^{2n}}\left|\sum_k (-1)^{f(k)}\right|^2$$

If the function is constant, then the sum $\sum_k (-1)^{f(k)}$ equals $\pm 2^n$ and the probability $|\langle 0^n|T^2|0^n\rangle|^2$ equals 1. If the function is balanced, both the sum and the probability is 0. □

Perhaps not surprisingly, to realize the famous Shor's algorithm [3], we also need only to square a matrix, however this time we need to have multiple accepting configurations.

Theorem 2. *The class MQ^2 solves the factoring problem. More precisely, there exists a constant k such that given numbers x and $N \geq k$ as in the Shor's algorithm, the language*

$$L = \{\langle N, i\rangle | x^a mod N \text{ has a period r whose i} - \text{th bit is 1 }\} \tag{2}$$

is in MQ^2.

Proof sketch. We will use the same notation as in the original paper by Shor [3]. N is the number to factorize and $N = p_1 p_2$ where both p_1 p_2 are primes and are different from each other. We then arbitrarily choose an x co-prime to N. The pair (x, N) is the input of the algorithm. The goal is to find the smallest $r \neq 0$ such that $x^r \bmod N = 1$. This r is called a *period* of x. We choose a number q such that q is a power of 2 and $q \geq 2^{2\lceil log_2 N\rceil}$. This number will, together with the length of x and N, determine the size of the matrix.

In the original setup, see Figure 3 a), we have three different transformations, namely: Hadamard transformation, a transformation computing power modulo N, and the DFT. One can notice that the effect on 0^n of the DFT and of the Hadamard transform is the same, so we suffice with only two different transformations. Furthermore, the matrices realizing them are poly-computable:

$$\langle x', N', a', i'|DFT|x, N, a, i\rangle \equiv \frac{1}{\sqrt{q}}\delta_{x,x'}\delta_{N,N'}\delta_{i,i'}e^{\frac{2i\pi}{q}aa'} \tag{3}$$

$$\langle x', N', a', i'|MOD|x, N, a, i\rangle \equiv \delta_{x,x'}\delta_{N,N'}\delta_{a,a'}\delta_{i'+i,x^a \bmod N} \tag{4}$$

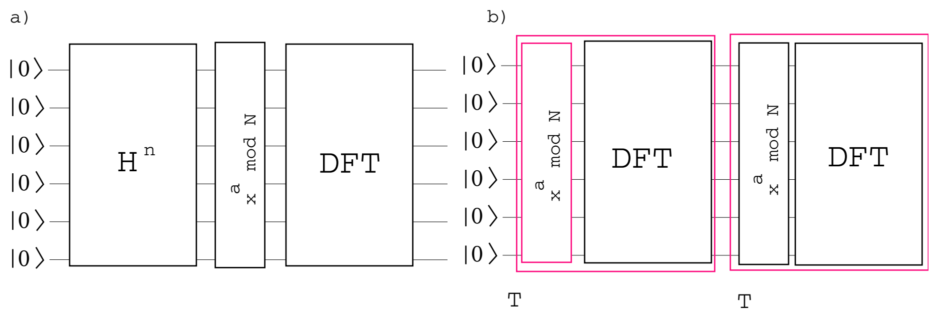


Fig. 3. The trick used to fit the Shor's algorithm into the MQ^2 class. In a), there is the original setup, in b) there is the one we use, which gives the same result up to a global phase.

One may also simply check that both the matrices are unitary. Their product $T \equiv DFT \cdot MOD$ is thus unitary too. Its elements read

$$\langle x', N', a', i'|T|x, N, a, i\rangle = \frac{1}{\sqrt{q}} \delta_{x,x'} \delta_{N,N'} e^{\frac{2i\pi}{q} a' a} \delta_{i+i', x^a \bmod N} \tag{5}$$

and are again clearly functions computable in poly-time. Applying two times the matrix T, we apply an extra MOD transformation comparing to the original setup. Nevertheless, on 0^n, this transform acts as identity. Any classical post-processing, as in the original algorithm, can be incorporated into the function $a(x, c)$. □

6 Conclusion

We saw a complexity class which has a compact purely mathematical definition. In order to describe an algorithm, we suffice with three functions taking bit-strings as arguments: $f(i, j, n)$, $I_n(x)$, and $a(x, c)$. Even quite a complex algorithm, as the Shor's certainly is, can be described on three lines. Further, the class MQ^2 shows that BQP is not the only possible class, lying in between BPP and AWPP, and not being trivially equal to either of them, which can do factorization and exponential speedup with oracles as in Deutsch-Jozsa's algorithm.

Acknowledgment

This material is partially based on the author's master thesis [4], which was supervised by Harry Buhrman. I would like to use this opportunity to thank him for motivating discussions.

References

1. E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing* 26(5):1411–1473, 1997, `citeseer.nj.nec.com/bernstein97quantum.html`.
2. D.Deutsch and R.Jozsa. *Proceedings of the Royal Society of London Ser. A* 439(553), 1992.
3. P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. *IEEE Symposium on Foundations of Computer Science*, pp. 124-134, 1994, `citeseer.nj.nec.com/14533.html`.
4. T. Tušarová. Quantum complexity classes. Master's thesis, Vrije Universiteit, Amsterdam, 2003, `http://arxiv.org/cs.CC/0409051`.

A Twelve-State Optimum-Time Synchronization Algorithm for Two-Dimensional Rectangular Cellular Arrays

Hiroshi Umeo, Masaya Hisaoka, and Shunsuke Akiguchi

University of Osaka Electro-Communication,
Neyagawa-shi, Hatsu-cho 18-8, Osaka, 572-8530, Japan
umeo@umeolab.osakac.ac.jp

Abstract. The firing squad synchronization problem has been studied extensively for more than 40 years [1-18]. The present authors are involved in research on firing squad synchronization algorithms on two-dimensional (2-D) rectangular cellular arrays. Several synchronization algorithms on 2-D arrays have been proposed, including Beyer [2], Grasselli [3], Kobayashi [4], Shinahr [10], Szwerinski [12] and Umeo et al. [13, 15]. To date, the smallest number of cell states for which an optimum-time synchronization algorithm has been developed is 14 for rectangular array, achieved by Umeo et al. [15]. In the present paper, we propose a new optimum-time synchronization algorithm that can synchronize any 2-D $m \times n$ rectangular arrays in $m+n+max(m,n)-3$ steps. We progressively reduce the number of internal states of each cellular automaton on rectangular arrays, achieving twelve states. This is the smallest number of states reported to date for synchronizing rectangular arrays in optimum-step.

1 Introduction

We study a synchronization problem which gives a finite-state protocol for synchronizing a large scale of cellular automata. The synchronization in cellular automata has been known as firing squad synchronization problem since its development, in which it was originally proposed by J. Myhill to synchronize all parts of self-reproducing cellular automata [7]. The firing squad synchronization problem has been studied extensively for more than 40 years [1-18]. The present authors are involved in research on firing squad synchronization algorithms on two-dimensional (2-D) cellular arrays. Several synchronization algorithms on 2-D arrays have been proposed, including Beyer [2], Grasselli [3], Kobayashi [4], Shinahr [10], Szwerinski [12] and Umeo et al. [15]. To date, the smallest number of cell states for which an optimum-time synchronization algorithm has been developed is 14 for rectangular array, achieved by Umeo et al. [13, 15].

In this paper, we propose a new optimum-time algorithm that can synchronize any 2-D $m \times n$ rectangular arrays in $m + n + max(m, n) - 3$ steps. The algorithm is based on a new efficient mapping scheme for embedding a special class of generalized one-dimensional optimum-time synchronization algorithms onto 2-D rectangular arrays. We progressively reduce the number of internal states of each cellular automaton on rectangular arrays, achieving twelve states. This is the smallest number of states reported to date for synchronizing rectangular arrays in optimum-step. Due to the limited space available, we omit the detailed proofs of the theorems given below.

2 Firing Squad Synchronization Problem

Figure 1 shows a finite two-dimensional (2-D) cellular array consisting of $m \times n$ cells. Each cell is an identical (except the border cells) finite-state automaton. The array operates in lock-step mode in such a way that the next state of each cell (except border cells) is determined by both its own present state and the present states of its north, south, east and west neighbors. All cells (*soldiers*), except the north-west corner cell (*general*), are initially in the quiescent state at time $t = 0$ with the property that the next state of a quiescent cell with quiescent neighbors is the quiescent state again. At time $t = 0$, the north-west corner cell $C_{1,1}$ is in the *fire-when-ready* state, which is the initiation signal for synchronizing the array. The firing squad synchronization problem is to determine a description (state set and next-state function) for cells that ensures all cells enter the *fire* state at exactly the same time and for the first time. The set of states must be independent of m and n. Several synchronization algorithms on 2-D arrays have been proposed, including Beyer [2], Grasselli [3], Kobayashi [4], Shinahr [10], Szwerinski [12] and Umeo et al. [13, 15]. Umeo, Maeda and Fujiwara [13] presented a 6-state non-optimum-time two-dimensional synchronization algorithm that fires any $m \times n$ arrays in $2(m + n) - 4$ steps. The algorithm is slightly slower than the optimum ones, but the number of internal states is considerably smaller. Shinahr [10] presented an optimum-time synchronization scheme

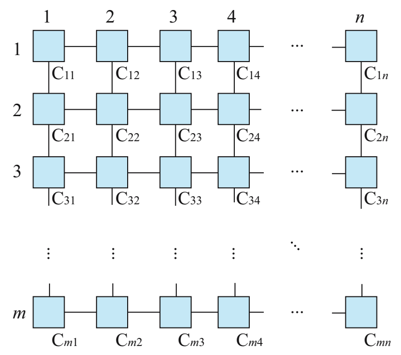


Fig. 1. A two-dimensional cellular automaton

in order to synchronize any $m \times n$ arrays in $m + n + max(m, n) - 3$ steps and gave a 28-state implementation. Recently, Umeo, Hisaoka, Teraoka and Maeda [15] proposed a 14-state implementation of 2-D optimum-time synchronization algorithm. To date, the smallest number of cell states for which an optimum-time synchronization algorithm has been developed is 14 for rectangular array, achieved by Umeo et al. [15].

3 A State-Efficient Mapping Scheme for Embedding 1-D Generalized Synchronization Algorithm onto 2-D Arrays

The proposal is a simple and efficient mapping scheme that enables us to embed a special class of one-dimensional generalized synchronization algorithm onto two-dimensional arrays without introducing additional states. We consider a 2-D array of size $m \times n$. At time $t = 0$ the general is located on $\mathrm{C}_{1,1}$. Without loss of generality we assume that $m \leq n$. We divide mn cells on the array of size $m \times n$ into $m + n - 1$ groups g_k, where $-(m-1) \leq k \leq n-1$, that is defined as follows:

$$g_k = \{\mathrm{C}_{i,j} | j - i = k, 1 \leq i \leq m, 1 \leq j \leq n\},\ -(m-1) \leq k \leq n-1 \text{ i.e.,}$$

$g_0 = \{\mathrm{C}_{1,1}, \mathrm{C}_{2,2}, ..., \mathrm{C}_{m,m}\}$,
$g_{-1} = \{\mathrm{C}_{2,1}, \mathrm{C}_{3,2}, ..., \mathrm{C}_{m,m-1}\}$,
$g_1 = \{\mathrm{C}_{1,2}, \mathrm{C}_{2,3}, ..., \mathrm{C}_{m,m+1}\}$,
$g_{-2} = \{\mathrm{C}_{3,1}, \mathrm{C}_{4,2}, ..., \mathrm{C}_{m,m-2}\}$,
$g_2 = \{\mathrm{C}_{1,3}, \mathrm{C}_{2,4}, ..., \mathrm{C}_{m,m+2}\}$,
.
.
$g_{-(m-1)} = \{\mathrm{C}_{m,1}\}$,
.
.
$g_{n-1} = \{\mathrm{C}_{1,n}\}$.

Figure 2 shows the division of the 2-D array of size $m \times n$ into $m + n - 1$ groups. Let $M = (Q, \delta_1, *)$ be any one-dimensional CA that fires ℓ cells with a general on k-th cell from the left end in $T(k, \ell)$ steps, where Q is the finite state set of M, $\delta_1 : Q^3 \rightarrow Q$ is the local transition function, and $* \in Q$ is the state of the right and left ends. We assume that M has $\ell = m + n - 1$ cells, denoted by C_i, $-(m-1) \leq i \leq n-1$. For convenience, we assume that M has a left and right end cells, denoted by C_{-m} and C_n, respectively. Both end cells C_{-m} and C_n always take the end state $*(\in Q)$.

We consider a one-to-one correspondence, illustrated in Fig. 2, between the i-th group g_i and the i-th cell C_i on M such that $g_i \leftrightarrow \mathrm{C}_i$, $-(m-1) \leq i \leq n-1$. We can construct a 2-D CA $N = (Q, \delta_2, *)$ such that all cells in g_i simulate the i-th cell C_i *in real-time* and N can fire any $m \times n$ arrays at time $t = T(m, m+n-1)$ if and only if M fires 1-D arrays of length $m + n - 1$ with a general on the m-th

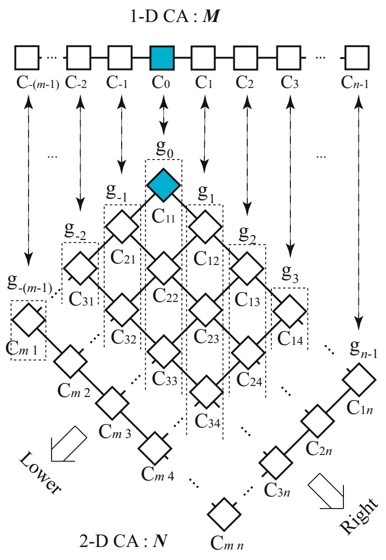
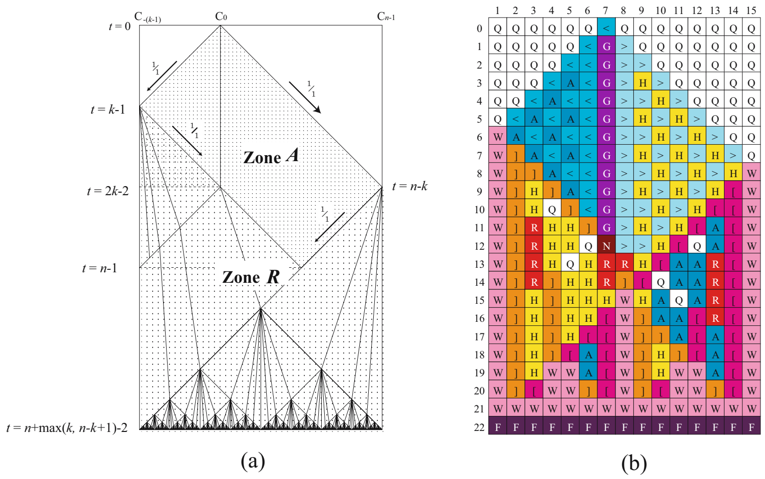


Fig. 2. Correspondence between 1-D and 2-D cellular arrays

cell from the left end at time $t = T(m, m+n-1)$, where $\delta_2 : Q^5 \rightarrow Q$ is the state transition function of N, and $* \in Q$ is the border state of the array. Note that the set of internal states of N is the same as M. A construction of the state transition function δ_2 will be given later.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Q	Q	Q	Q	Q	Q	<	Q	Q	Q	Q	Q	Q	Q	Q
1	Q	Q	Q	Q	Q	<	G	>	Q	Q	Q	Q	Q	Q	Q
2	Q	Q	Q	Q	<	<	G	>	>	Q	Q	Q	Q	Q	Q
3	Q	Q	Q	<	A	<	G	>	H	>	Q	Q	Q	Q	Q
4	Q	Q	<	A	<	<	G	>	>	H	>	Q	Q	Q	Q
5	Q	<	A	<	A	<	G	>	H	>	H	>	Q	Q	Q
6	W	A	<	A	<	<	G	>	>	H	>	H	>	Q	Q
7	W	]	A	<	A	<	G	>	H	>	H	>	H	>	Q
8	W	]	]	A	<	<	G	>	>	H	>	H	>	H	W
9	W	]	H	]	A	<	G	>	H	>	H	>	H	[	W
10	W	]	H	Q	]	<	G	>	>	H	>	H	[	[	W
11	W	]	R	H	H	]	G	>	H	>	H	[	A	[	W
12	W	]	R	H	H	Q	N	>	>	H	[	Q	A	[	W
13	W	]	R	H	Q	H	R	R	H	[	A	A	R	[	W
14	W	]	R	]	H	H	R	]	[	Q	A	A	R	[	W
15	W	]	H	]	H	H	H	W	H	A	Q	A	R	[	W
16	W	]	H	]	H	H	[	W	]	A	A	[	R	[	W
17	W	]	H	]	H	[	[	W	]	]	A	[	A	[	W
18	W	]	H	]	[	A	[	W	]	H	]	[	A	[	W
19	W	]	H	W	W	A	[	W	]	H	W	W	A	[	W
20	W	]	[	W	W	]	[	W	]	[	W	W	]	[	W
21	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
22	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Fig. 3. Time-space diagram for generalized optimum-step firing squad synchronization algorithm

Figure 3 shows a time-space diagram for a generalized optimum-time firing squad synchronization algorithm that fires 1-D $n+k-1$ cells in $T(k, n+k-1) = n+k-3+max(k,n)$ steps, where the general is on the k-th cell from the left end. We consider a special class of transition rule set with a *property* $\mathcal{A}$ for the generalized optimum-time firing squad synchronization algorithms.

Property $\mathcal{A}$: Let s_i^t denote the state of C_i at step t. We say that a generalized firing squad synchronization algorithm has a *property* $\mathcal{A}$, where any state s_i^t appearing *in the zone* $\mathcal{A}$ of the time-space diagram shown in Fig. 3 can be computed from its left and right neighbor states s_{i-1}^{t-1} and s_{i+1}^{t-1} and it never depends on its own previous state s_i^{t-1}.

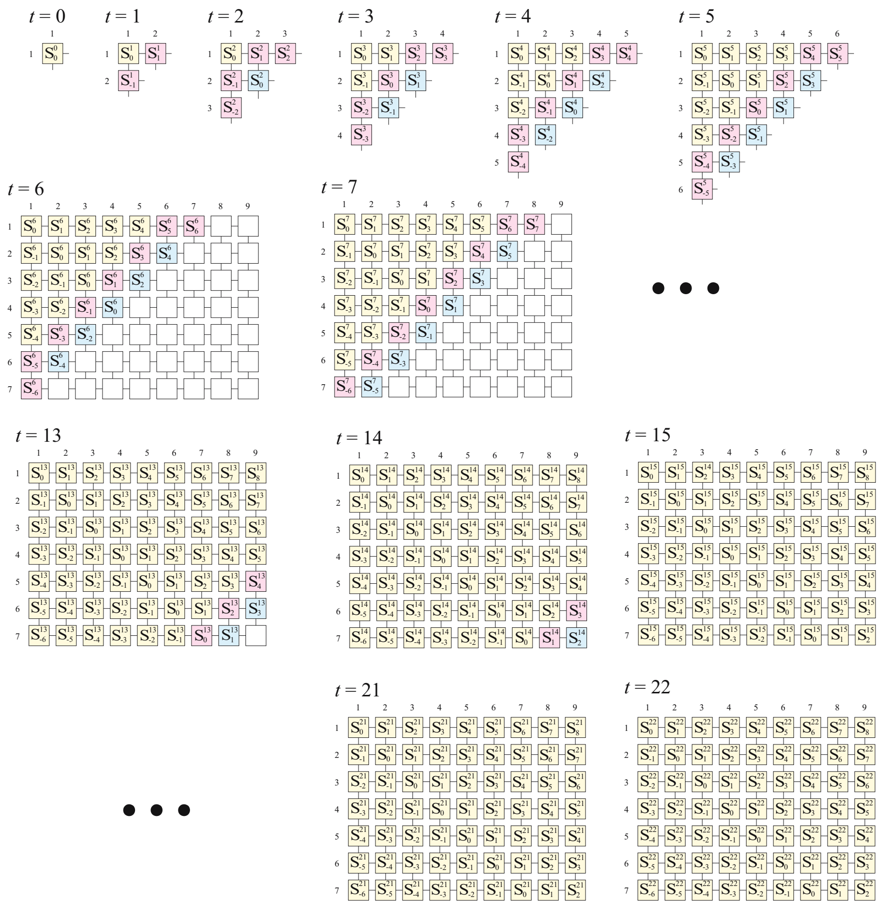


Fig. 4. An embedding of configurations of the 1-D CA performing generalized optimum-time synchronization operations into 2-D arrays

The one-dimensional generalized firing squad synchronization algorithm with the *property* $\mathcal{A}$ can be easily embedded onto two-dimensional arrays without introducing any additional states. It is seen that the construction of 2-D CA N can generate the configuration of M in real-time. Specifically, for any i, $-(m-1) \leq i \leq n-1$, the state of any cell in g_i at any step is either the quiescent state or the state of C_i at each corresponding step. Let s_i^t and $s_{k,\ell}^t$ denote the state of C_i and $\mathrm{C}_{k,\ell}$ at step t, respectively, where $-(m-1) \leq i \leq n-1$, $1 \leq k \leq m$, $1 \leq \ell \leq n$.

Figure 4 shows an embedding of configurations on 1-D cellular array of length $m+n-1$ with the *Property* $\mathcal{A}$ onto 2-D array of size $m \times n$, where $m = 7$, $n = 9$ and $0 \leq t \leq T(7, 15) = 22$. To save a space available, we will explain straightforwardly how to simulate the states of M in real-time on N. At time $t = 1$, $\mathrm{C}_{1,1}$, $\mathrm{C}_{1,2}$, and $\mathrm{C}_{2,1}$ can easily take a state s_0^1, s_1^1, and s_{-1}^1, respectively, since each cell finds necessary states in its neighbors at step $t = 0$. Similarly, at step $t = 2$, $\mathrm{C}_{1,1}$, $\mathrm{C}_{1,2}$, $\mathrm{C}_{1,3}$, $\mathrm{C}_{2,1}$, and $\mathrm{C}_{3,1}$ can take their states s_0^2, s_1^2, s_2^2, s_{-1}^2, and s_{-2}^2, respectively. But the computation of s_0^2 on the cell $\mathrm{C}_{1,1}$ is difficult to be done, since we assume that M has von Neumann neighborhood, thus at step $t = 1$ the cell $\mathrm{C}_{2,2}$ cannot access $\mathrm{C}_{1,1}$ in sate s_0^1 that is normally used to compute s_0^2. To make it possible, we assume the *Property* $\mathcal{A}$. With help of the *property* $\mathcal{A}$, the cell $\mathrm{C}_{2,2}$ can take the state s_0^2 at step $t = 2$. The similar situations occur in the computation of states appearing only at the wave-front of the valid configuration of N (shown in Fig. 4). To be precise, the computation of shaded states in the area A shown in Fig. 4(a) is involved with the *property* $\mathcal{A}$.

The state transition table of N consists of three parts, one is a rule set (Type (A)) that is for the inner cells excluding the wave-front cells and the other set (Type (B) and Type (C)) is for the computation of states of cells in the wave-front area. Let $\delta_1(a, b, c) = d$ be any transition rule of M, where $a, b, c, d \in \{Q - \{w\}\}$. Then, N has nine Type (A) transition rules, as shown in Fig. 5. The first rule (a-1) in Type (A) is used by an inner cell that does not include border cells amongst its four neighbors. Rules (a-2)-(a-5) are used by an inner cell that has a border cell as its upper, lower, left, right, lower left, or upper right neighbor, respectively. Here the terms *upper*, *right* etc. on the rectangular array are interpreted in a usual way, shown in Fig. 2, although the array is rotated by 45° in the clockwise direction. Rules (a-6)-(a-9) are used by an inner cell that has border cells as its right lower, left lower, right upper, and left upper neighbors, respectively. As for the rules related with the computation at the wave-front area, we have four rules (b-1)-(b-4) and seven rules (c-1)-(c-7), which are used by the cells located at the wave-front of each configuration of N.

We define the following set of cells. Let t be any integer such that $0 \leq t \leq T(m, m+n-1)$ and $\mathcal{S}$, $\mathcal{S}_t$, $\overline{\mathcal{S}_t}$ be set of cells such that:

$$\mathcal{S} = \{\mathrm{C}_{k,\ell} | 1 \leq k \leq m, 1 \leq \ell \leq n\},$$
$$\mathcal{S}_t = \{\mathrm{C}_{k,\ell} | 2 \leq k + \ell \leq t + 2, 1 \leq k \leq m, 1 \leq \ell \leq n\},$$
$$\overline{\mathcal{S}_t} = \mathcal{S} - \mathcal{S}_t.$$

Let $S^t_{g_i}$ denote a set of states of the cells in $g_i \cap \mathcal{S}_t$ at step t and $\overline{S}^t_{g_i}$ be a set of states of the cells in $g_i \cap \overline{\mathcal{S}_t}$ at step t, where $0 \leq t \leq T(m, m+n-1)$ and $-(m-1) \leq i \leq n-1$. We can establish the following lemmas for $g_i, -(m-1) \leq i \leq n-m+1$. Throught those lemmas, we assume that $m \leq n$ and q is the quiescent state of N.

Lemma 1. g_0:

1. For any t such that $0 \leq t \leq 2m-3$, $S^t_{g_0} = \{s^t_0\}$ and $\overline{S}^t_{g_0} = \{q\}$.
2. For any t such that $2m-2 \leq t \leq T(m, m+n-1)$, $S^t_{g_0} = \{s^t_0\}, \overline{S}^t_{g_0} = \phi$.

Lemma 2. $g_i (1 \leq i \leq n-m)$:

1. For any integer t such that $0 \leq t \leq i-1$, $S^t_{g_i} = \phi, \overline{S}^t_{g_i} = \{q\}$.
2. For any t such that $i \leq t \leq i+2m-3$, $S^t_{g_i} = \{s^t_i\}$ and $\overline{S}^t_{g_i} = \{q\}$.
3. For any t such that $i+2m-2 \leq t \leq T(m, m+n-1)$, $S^t_{g_i} = \{s^t_i\}, \overline{S}^t_{g_i} = \phi$.

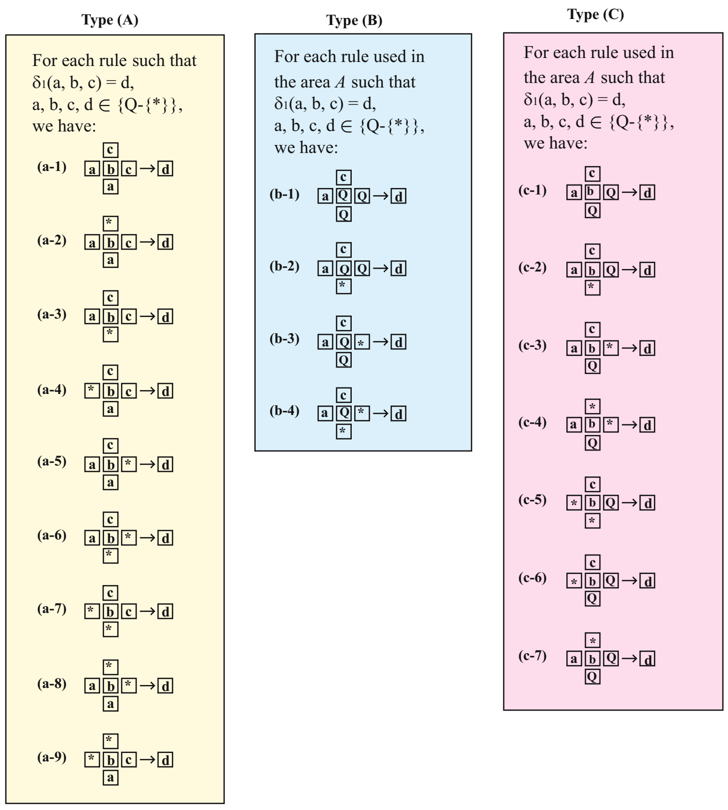

Type (A)
For each rule such that δ1(a, b, c) = d, a, b, c, d ∈ {Q-{*}}, we have:
(a-1)
(a-2)
(a-3)
(a-4)
(a-5)
(a-6)
(a-7)
(a-8)
(a-9)
Type (B)
For each rule used in the area A such that δ1(a, b, c) = d, a, b, c, d ∈ {Q-{*}}, we have:
(b-1)
(b-2)
(b-3)
(b-4)
Type (C)
For each rule used in the area A such that δ1(a, b, c) = d, a, b, c, d ∈ {Q-{*}}, we have:
(c-1)
(c-2)
(c-3)
(c-4)
(c-5)
(c-6)
(c-7)

Fig. 5. Construction of transition rules for 2-D optimum-time firing squad synchronization algorithm

Lemma 3. $g_i(n-m+1 \leq i \leq n-1)$:

1. For any integer t such that $0 \leq t \leq i-1$, $S^t_{g_i} = \phi$, and $\overline{S}^t_{g_i} = \{q\}$.
2. For any t such that $i \leq t \leq 2n-i-3$, $S^t_{g_i} = \{s^t_i\}, \overline{S}^t_{g_i} = \{q\}$.
3. For any t such that $2n-i-2 \leq t \leq T(m, m+n-1)$, $S^t_{g_i} = \{s^t_i\}, \overline{S}^t_{g_i} = \phi$.

Lemma 4. $g_{-i}(1 \leq i \leq m)$:

1. For any t such that $0 \leq t \leq i-1$, $S^t_{g_{-i}} = \phi, \overline{S}^t_{g_{-i}} = \{q\}$.
2. For any t such that $i \leq t \leq 2m-i-2$, $S^t_{g_{-i}} = \{s^t_{-i}\}, \overline{S}^t_{g_{-i}} = \{q\}$.
3. For any t such that $2m-i-1 \leq t \leq T(m, m+n-1)$, $S^t_{g_{-i}} = \{s^t_{-i}\}, \overline{S}^t_{g_{-i}} = \phi$.

Based on the observations above, we can get the following theorem.

Theorem 5. Let M be any s-state generalized synchronization algorithm with the *property* $\mathcal{A}$ operating in $T(k, \ell)$ steps on one-dimensional ℓ cells with a general on the k-th cell from the left end. Then, based on the transformation shown in

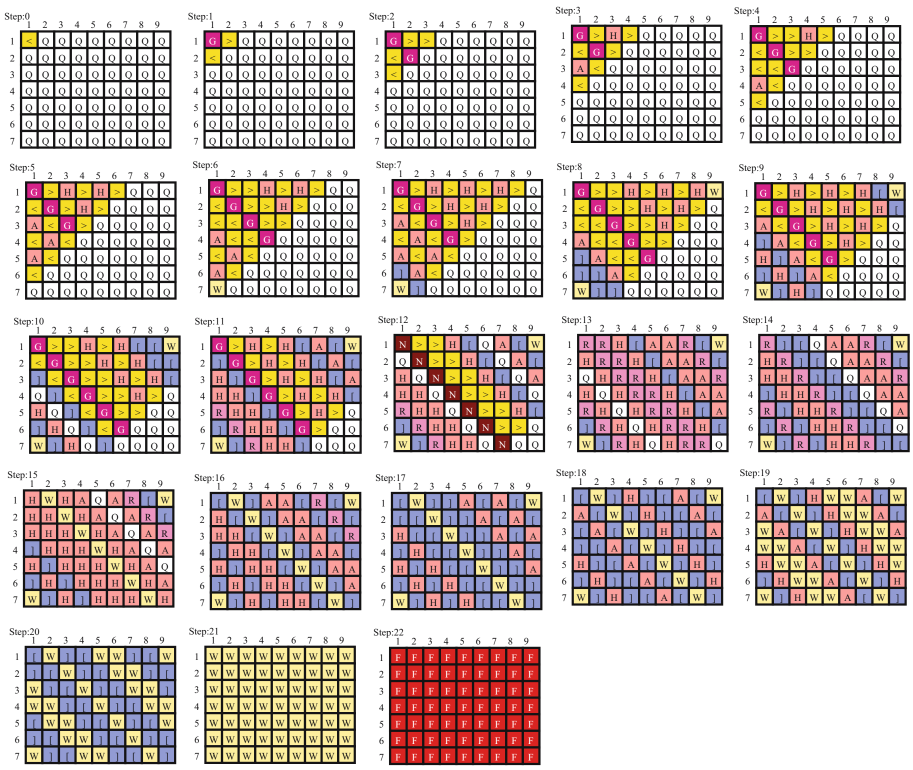


Fig. 6. Snapshots of the proposed 12-state optimum-time firing squad synchronization algorithm on rectangular arrays

Fig. 5, we can construct a two-dimensional s-state cellular automaton N that can synchronize any $m \times n$ rectangular array in $T(m, m+n-1)$ steps.

Moore and Langdon [8], Szwerinski [12] and Varshavsky, Marakhovsky and Peschansky [17] developed a generalized optimum-time firing algorithm with 17, 10 and 10 internal states, respectively, that fires 1-D ℓ cells in $\ell - 2 + max(k, \ell - k + 1)$ steps, where the general is located on C_k. Recently, Settle and Simon [11] and Umeo et al. [14] have proposed a new 9-state generalized synchronization algorithm operating in optimum-step. After revealing that none of those transition rules given in [8, 12, 14, 17] have the *property* $\mathcal{A}$, we newly develop a 12-state transition rule set satisfying the condition. Due to the space available, we omit the whole set of transition rules with the *property* $\mathcal{A}$. In Fig. 3(b) we show some snapshots for the synchronization. The next theorem is our 12-state implementation of the generalized optimum-time synchronization algorithm having the *property* $\mathcal{A}$.

Theorem 6. There exists a 12-state one-dimensional cellular automaton with the *property* $\mathcal{A}$ that can synchronize ℓ cells with a general on the k-th cell from the left end in optimum $\ell - 2 + max(k, \ell - k + 1)$ steps.

Based on [Theorems 5, 6] and by letting $k = m$, $\ell = m + n - 1$ in $T(k, \ell) = \ell - 2 + max(k, \ell - k + 1)$, the time complexity of the 2-D synchronization algorithm is $m + n + \max(m, n) - 3$. Thus, the algorithm is a time-optimum one. Now we can get a 12-state optimum-time synchronization algorithm for rectangular arrays.

Theorem 7. There exists a 12-state firing squad synchronization algorithm that can synchronize any $m \times n$ rectangular array in optimum $m + n + max(m, n) - 3$ steps.

4 Conclusions

We have proposed a new optimum-time algorithm that can synchronize any $m \times n$ two-dimensional rectangular arrays in $m + n + max(m, n) - 3$ steps. The algorithm is based on a state-efficient mapping scheme for embedding a special class of generalized one-dimensional optimum-time firing squad synchronization algorithms with the *property* $\mathcal{A}$ onto 2-D rectangular arrays. In our construction an $m \times n$ array synchronization problem is reduced to one 1-D generalized synchronization problem. We progressively reduce the number of internal states of each cellular automaton operating in optimum-step on rectangular arrays, achieving twelve states. This is the smallest number of states reported to date for synchronizing rectangular arrays in optimum-step.

References

1. R. Balzer: An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, vol. 10(1967), pp. 22-42.
2. W. T. Beyer: Recognition of topological invariants by iterative arrays. Ph.D. Thesis, MIT, (1969), pp. 144.
3. A. Grasselli: Synchronization of cellular arrays: The firing squad problem in two dimensions. *Information and Control*, vol. 28(1975), pp. 113-124.

4. K. Kobayashi: The firing squad synchronization problem for two-dimensional arrays. *Information and Control*, vol. 34(1977), pp. 177-197.
5. J. Mazoyer: A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, vol. 50(1987), pp. 183-238.
6. M. Minsky: *Computation: Finite and infinite machines.* Prentice Hall, (1967), pp. 28-29.
7. E. F. Moore: The firing squad synchronization problem. in *Sequential Machines, Selected Papers* (E. F. Moore ed.), Addison-Wesley, Reading MA., (1964), pp. 213-214.
8. F. R. Moore and G. G. Langdon: A generalized firing squad problem. *Information and Control*, vol. 12(1968), pp. 212-220.
9. H. B. Nguyen and V. C. Hamacher: Pattern synchronization in two-dimensional cellular space. *Information and Control*, vol. 26(1974), pp. 12-23.
10. I. Shinahr: Two- and three-dimensional firing squad synchronization problems. *Information and Control*, vol. 24(1974), pp. 163-180.
11. A. Settle and J. Simon: Smaller solutions for the firing squad. *Theoretical Computer Science*, vol. 276(2002), pp.83-109.
12. H. Szwerinski: Time-optimum solution of the firing-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position. *Theoretical Computer Science*, vol. 19(1982), pp. 305-320.
13. H. Umeo, M. Maeda and N. Fujiwara: An efficient mapping scheme for embedding any one-dimensional firing squad synchronization algorithm onto two-dimensional arrays. *Proc. of the 5th International Conference on Cellular Automata for Research and Industry*, LNCS 2493, Springer-Verlag, pp.69-81(2002).
14. H. Umeo, M. Hisaoka, K. Michisaka, K. Nishioka and M. Maeda: Some generalized synchronization algorithms and their implementations for a large scale cellular automata. *Proc. of the Third International Conference on Unconventional Models of Computation, UMC 2002*, LNCS 2509, Springer-Verlag, pp.276-286(2002).
15. H. Umeo, M. Hisaoka, M. Teraoka and M. Maeda: Several new generalized linear- and optrimum-time synchronization algorithms for two-dimensional rectangular arrays. *Proc. of the 4th Intern. Conf., MCU 2004*, LNCS 3354, Springer-Verlag, pp.223-232(2005).
16. H. Umeo, M. Hisaoka and T. Sogabe: A survey on optimum-time firing squad synchronization algorithms for one-dimensional cellular automata. *International Journal of Unconventional Computing*, Vol.1, No.3, pp.1-24(2005).
17. V. I. Varshavsky, V. B. Marakhovsky and V. A. Peschansky: Synchronization of interacting automata. *Mathematical Systems Theory*, Vol. 4, No. 3(1970), pp. 212-230.
18. A. Waksman: An optimum solution to the firing squad synchronization problem. *Information and Control*, vol. 9(1966), pp. 66-78.

Computing by Self-reproduction: Autopoietic Automata[⋆]

Jiří Wiedermann

Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. We introduce a new formal computational model designed for studying the information transfer among the generations of offspring–producing machines — so–called autopoietic automata. These can be seen as finite state transducers whose "program" can become a subject of their own processing. An autopoietic automaton can algorithmically generate an offspring controlled by a program which is a modification of its parent's program. We show that the computational power of lineages of autopoietic automata is equal to that of an interactive nondeterministic Turing machine. We also prove that there exists an autopoietic automaton giving rise to an unlimited evolution, providing suitable inputs are delivered to individual automata. However, the problem of a sustainable evolution, asking for an arbitrary autopoietic automaton and arbitrary inputs whether there is an infinite lineage of its offspring is undecidable.

1 Introduction

The notion of autopoiesis was coined by Chilean biologists Maturana and Varela since the nineteen seventies. Literally, autopoiesis means self–production and denotes a process whereby a system (or an "organization", as Maturana and Varela call it) produces itself (for more details concerning computational autopoiesis, cf. [2]). Autopoiesis, as its proponents understand it, is not a precisely, mathematically or otherwise formally defined notion and in fact we will use this notion in its literal meaning to denote self–producing or self–creating units. The reason for calling our model autopoietic automata has been the aspiration to distinguish such automata by name from the notoriously known self–reproducing automata which are a kind of cellular automata. The autopoietic automata are definitely not meant to model autopoiesis in the sense of Maturana and Varela. These automata are not based on the formalism of cellular automata and in fact they build upon the classical models of the finite state automata. The autopoietic automata are designed as a mathematical model of self-evolving units capturing the computational (or information processing) essence of self–production, i.e., the use of

[⋆] This research was carried out within the institutional research plan AV0Z10300504 and partially supported by grant No. 1ET100300517 within the National Research Program "Information Society".

a "program" both to drive the (computational) behavior of a unit and to serve as a "template" for the evolutionary process. The lastly mentioned idea originates, of course, from von Neumann [5] whose stress in designing his self–reproducing automata had been just on the design of mechanisms of self–reproduction alone. In our modelling, however, we will not be concerned in these mechanisms: rather, we take their existence as granted and we concentrate instead on algorithms controlling the variations in the offspring–production process and (hence) the ("genetic") information transfer from the parental machine to its offspring. That is, we will not be interested in producing exact copies of the parental machine: in our modelling we will focus on the evolution in which offsprings possess qualities different from their parents. Thus, instead of self–reproduction we should rather speak more precisely about self–like, or offspring–production.

In von Neumann's seminal paper on self–reproducing automata, the problem of the variation of genetic information was not the main issue. Nevertheless, a related question concerning the "evolutionary growth of complexity" of self–reproducing automata has become the issue in the field of artificial life (for an overview, see [1]). In the absence of suitable computational models neither this question nor the related problem of the computational power of automata exhibiting the evolutionary growth of their complexity could have been answered convincingly.

In this paper we present a computational model answering the previous questions. Our model is inspired by contemporary cellular biology. In its design it abstracts the information processing, reproducing and evolutional abilities of the living cells. An autopoietic automaton is a specific kind of a nondeterministic finite state transducer which has access to the representation of its own transition relation. Controlled by this transition relation and making use of the possibility to read the representation of this relation, an autopoietic automaton computes and outputs the transition relation of its offspring. In this way the changes in the new transition relation are controlled by the parental machine. Our main result shows that a series of lineal descents of a single autopoietic automaton (a lineage of autopoietic automata) has a notable computational power — the same as an interactive nondeterministic Turing machine. We also construct an autopoietic automaton which generates a lineage containing all autopoietic automata, i.e., the members of this lineage exhibit unbounded growth of complexity in the computational sense. Within our model, this result answers positively the related question asked by McMullin and its predecessors in the field of artificial life (cf. [1]). Finally, we define the problem of the so–called sustainable evolution which asks after any autopoietic automaton and any infinite sequence of inputs whether there is an infinite lineage generated by that automaton on that input. We show that this is an undecidable problem.

The content of the paper is as follows. Section 2 contains the formal definition of an autopoietic automaton and of its computations, too. In Section 3 the computational power of lineage of autopoietic automata is characterized via interactive nondeterministic Turing machines. The computational aspects of the evolution of autopoietic automata, especially the unboundedness of evolutionary complexity growth and an evolution's sustainability, are studied in Section 4. Section 5 recapitulates the main contributions of the paper.

It is very tempting to interpret the previous results in the framework of original scientific disciplines which served as inspiration for our modelling. As to the extent to which our model captures the information processing and evolutionary abilities of living cells, our results seem to be among the first formal results shedding light on the computational nature and power of the respective mechanisms.

2 Autopoietic Automata

Autopoietic automata are nondeterministic finite state machines capturing the information processing, reproducing and evolving abilities of living cells. Technically, an autopoietic automaton is a nondeterministic transducer (a Mealy automaton) computing and outputting the transition relation of its offspring. The design of an autopoietic automaton supports working in two modes. The first of them is a standard transducer mode controlled by a transition relation and processing external input information read through an input port. In this phase the results of a computation (if any) are sent to the output port. The second mode is a reproducing mode which is controlled by the same transition relation as before. This time, however, no external information is taken into account and, instead, the representation of the transition relation itself is used as a kind of internal input. For this purpose the representation of the automaton's own transition relation is available to an autopoietic automaton on a special, so–called program tape. It is a two–way read–only tape. The result of the reproducing mode is written on a special one–way write–only output tape. Of course, both tapes mentioned before are finite. After finishing the reproduction, the information written on this tape is interpreted as a transition relation of a new autopoietic automaton and the tape itself becomes a new automaton's program tape. The new automaton starts its activity with the empty output tape. The new automaton is seen as an offspring of the original automaton. Depending on the transition relation of the parental automaton, the transition function of the new automaton can differ from the original transition relation. Schematically, the architecture of an autopoietic automaton is depicted in Fig. 1.

Now we are ready to proceed to a formal definition of an autopoietic automaton. One of our final aims is to study the sequences of such automata with an

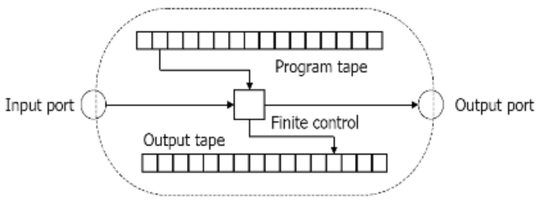


Fig. 1. The schema of an autopoietic automaton

increasing number of internal states and working over alphabets of increasing size. Therefore, in general we will consider an infinite set Q of states whose members will be numbered, i.e., $Q = \{q_1, q_2, \ldots\}$ and similarly an infinite, so–called external working alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$. An equivalent, so–called *internal representation* of the members of these sets will be via sequences of zeros: the i–th member will be encoded as a sequence of i zeros, abbreviated as 0^i.

Definition 1. *An* autopoietic automaton *is a six–tuple* $A = \{\Sigma, Q, R, q_1, q_2, \delta\}$, *where*

- Σ, *with* $\varepsilon \in \Sigma$, *is the finite or infinite* external alphabet *whose symbols are read on the input port or are written to the output port, one symbol at a time;* ε *is the empty symbol;*
- Q *is the finite or infinite* set of states*;*
- $R \subset Q$, $R \neq \emptyset$ *is the distinguished* set of reproducing states*;*
- $q_1 \in Q - R$ *is the* initial state *in which the computation of* A *starts, with either head at the left end on the respective tape;*
- $q_2 \in R$ *is the* final reproducing state*; entering it finishes the reproduction mode of* A *and starts a computation of the* A*'s offspring which is an autopoietic automaton whose transition relation had been generated by* A *on its output tape. Simultaneously,* A *empties its output tape and restarts its computation from its initial state. Since this time on the computation of both* A *and that of its offspring have been independent, each of them receiving its own input;*
- δ, *the* transition relation*, comes in two forms depending on whether* $q \in Q - R$ *or* $q \in R$ *:*
  - Transducer mode*: if* A *is in state* $q \in Q - R$, *then we say that* A *is in a transducer mode. Then* δ *is a finite subset of* $\Sigma \times Q \times \Sigma \times Q \times D$, *where* $D = \{d_1, d_2, d_3, d_4\}$ *is the* alphabet of directions *corresponding to the moves of the program tape head (*d_1 *denotes the left shift by one position,* d_2 *the right shift by one position,* d_3 *means no shift,* d_4 *means that the shift direction is undefined, which is the case in the transducer mode). The elements of* δ *are formed by five–tuples of form* $(\sigma_i, q_j, \sigma_k, q_\ell, d_m) \in \Sigma \times Q \times \Sigma \times Q \times D$ *from which a sequence (in arbitrary order) is formed and encoded in binary on the program tape; the respective encoding for the above mentioned tuple is* $10^i10^j10^k10^\ell10^m1$; *the encoding of all tuples representing* δ *is also embraced by 1s (i.e., the entire encoding starts and ends by two consecutive 1s). A tuple* $(\sigma_i, q_j, \sigma_k, q_\ell, d_4) \in \Sigma \times Q \times \Sigma \times Q \times D$ *corresponds to one computational step (transition) of* A *in the following way: if* A *is in state* $q_j \in Q - R$ *and* $\sigma_i \in \Sigma$ *is a symbol at the input port,* A *changes its state to* q_ℓ *and writes* $\sigma_k \in \Sigma$ *to its output port; the position of the head on the program and output tape remains unchanged.*
  - Reproducing mode: *if* A *is in a state* $q \in R$, *then* A *is in a reproducing mode. In such a case* δ *is a finite subset of* $\{0,1\} \times Q \times \{0,1\} \times Q \times D$. *A tuple* $(\sigma_i, q_j, \sigma_k, q_\ell, d_m) \in \{0,1\} \times Q \times \{0,1\} \times Q \times D$ *is represented on the* A*'s program tape similarly as any tuple of a transducer mode,*

with 0 *representing the element* 0 *and* 00 *representing* 1. *Such a tuple corresponds to one computational step of* A *in the following way: if* A *is in state* $q_j \in R$, *and* $\sigma_i \in \{0, 1\}$, *the symbol scanned by the read head on the program tape,* A *changes its state to* q_ℓ *and writes* $\sigma_k \in \{0, 1\}$ *to its output tape; doing so the head on the program tape shifts in direction* $d_m (d_m \neq d_4)$ *and the head on the output tape advances by one position to the right.*

Note please that we have admitted that both sets Σ and Q can be infinite sets. However, the transition relation δ must always be finite (or more precisely: a finite subset of $\Sigma \times Q \times \Sigma \times Q \times D$). This unusual arrangement allows an autopoietic automaton to generate offsprings working over larger (or different) sets of states or symbols than it was possible for the original automaton. In Section 4 we will see that this is what enables a kind of evolutionary growth of complexity of the underlying automata. Also note that in order to distinguish the types of individual states also syntactically we made use of the last component (corresponding to the program tape head move direction) in the five–tuple representing a transition.

An autopoietic automaton A starts its computation in state q_1, with the head on either tape in the leftmost position. The automaton reads the symbol appearing on its input port and realizes the respective transition as described in the previous definition. In general, thanks to nondeterminism, the transition relation allows several choices for the next step. As is customary with nondeterministic computations we take the viewpoint that any choice that will eventually lead the automaton to enter the final reproducing state q_2 is a legal move.

In automaton's further activities, the general rule is that while being in non–reproducing states the automaton reads the symbols from the input port and writes the symbols to the output port, possibly moving its head along the program tape. No symbols are written to the output tape. When entering a reproducing state, instead of the external symbols the automaton reads the symbols scanned by its head on the program tape and writes the binary symbols to the output tape. Entering the final reproducing state q_2, A terminates its current activities with (the binary representation of a) new transition relation δ_{new} written on its output tape, and *reproduces by fission*, so to say. It splits into two automata (see Fig. 2): the first one is driven by the original transition relation δ (denoted as Program 1 in Fig. 2) while the other one by relation δ_{new} (denoted as Program 2). The new automata start with the empty output tapes.

Thanks to the fact that on the same inputs the final reproducing state can be achieved via several computational paths, a single nondeterministic autopoietic automaton can produce several different offsprings, not just one. In our further considerations we will assume that all such offsprings are produced, indeed. Due to the nature of the fission mechanism one of the offsprings will be identical to its parent. We can imagine that after the fission all automata continue processing their own input symbols. In principle, we see that by iterating this scenario a potentially infinite tree of offsprings can be generated from a single autopoietic automaton. In this tree, each offspring is related to its parent. Now, if we con-

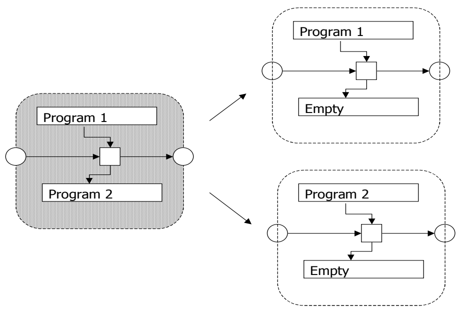


Fig. 2. The fission of an autopoietic automaton

centrate on a single path starting in the root of such a tree, we get a so–called *lineage of autopoietic automata.* In its entirety each lineage realizes a translation of a potentially infinite stream of the input symbols into a similar stream of output symbols. Note that along such a computational path the automata enter the reproducing states infinitely often. In the sequel we will study the computational power of lineages.

3 The Power of Lineages of Autopoietic Automata

The first question concerning any computational model is the one of its computational power. The computational power of an autopoietic automaton is not different from that of a finite state transducer: this is because it is driven by a finite state mechanism, and its ability to read its own "program" does not add any power, since in principle the same information could be stored in the automaton's states. Nevertheless, when considering a lineage of autopoietic automata things get more interesting. We show the equivalence of lineages of autopoietic automata with so–called interactive Turing machines. This type of machines has been introduced by van Leeuwen and Wiedermann (cf. [3], [4]) when studying the so–called interactive evolutionary algorithms. Interactive Turing machines are variants of standard Turing machines adapted for processing infinite input streams. That is, instead of the input tape with a priori given input data these machines read the input data through an input port much like the autopoietic automata; the output symbols are also treated in a similar way. A nondeterministic interactive Turing machine M is said to realize a *translation* from an infinite input stream S_1 to an infinite output stream S_2 if there exists a computation of M on S_1 passing through an infinite number of accepting states of M and producing S_2 as its output (we assume that after entering an accepting state, M can prolong its computations). Two translations are considered to be *equivalent* if they are equivalent after deleting all empty symbols from them.

The equivalence between a lineage of autopoietic automata and an interactive Turing machine will be shown by mutual simulations of these devices. We start with a simpler case — namely simulating a lineage of autopoietic automata by an interactive Turing machine. Prior to proceeding to the respective simulation we must solve one fine detail. This is the problem of the unbounded input alphabet and that of translating the symbols of the external alphabet to their internal representation being used on the program tape. While the offsprings of autopoietic automata can, by their very definition, work with increasingly complex symbols, for a Turing machine with a fixed transition relation this is not possible: more complex symbols require a longer encoding. Therefore we will assume that the elements of Σ which can be directly read by the members of a lineage of autopoietic automata will be presented to a Turing machine in their unary notation, as stated in Definition 1. That is, for a Turing machine a symbol $\sigma_i \in \Sigma$ will be presented as a string of form 0^i and the strings in a sequence will be separated by ones. Thus, a Turing machine needs $O(i)$ steps to read (or write down) σ_i. However, we are not interested in the exact complexity of our simulations — we are merely concerned with the principal possibility of such simulations and in this respect a less efficient coding does not make any difference.

Having said so we are ready to present our first simulation theorem:

Theorem 1. *Any lineage of autopoietic automata can be simulated by a nondeterministic interactive Turing machine.*

Sketch of the proof. Let $\mathcal{A} = A_1, A_2, \ldots$ be a lineage of the autopoietic automata. We design a nondeterministic interactive Turing machine M simulating $\mathcal{A}$ and working as follows. M is a universal Turing machine which reads via its input port the inputs encoded in the unary notation and is able to simulate any A_i given by the description (encoding) of its transition relation δ_i. For such a purpose, M maintains the representation of both tapes of A_i on its tapes: the program tape, on which a representation of δ_i is written in the form as stated in Definition 1, and the output tape on which the transition relation δ_{i+1} of A_{i+1} is generated, for $i = 1, 2, \ldots$. Starting from $i = 1$, M simulates the actions of A_i as dictated by δ_i written on the program tape until A_i reproduces and the processing is taken over by A_{i+1}. In such a case, M enters its accepting state and exchanges the roles of its two tapes: the output tape becomes the program tape with δ_{i+1} already being written on it, and the original program tape after being "cleaned" becomes the new output tape. Clearly, in this way M realizes the same translation as $\mathcal{A}$ does. $\square$

The reverse simulation is more complicated and requires more preliminaries. First of all, we have to specify, in more detail, the Turing machine to be simulated. We will consider a nondeterministic interactive Turing machine M with one input and one output port and with only one working tape unbounded to the right. The input and output symbols will be from a finite alphabet $\Sigma_M \subset \Sigma$, with Σ being the external alphabet of the autopoietic automata at hand. The working (or tape) alphabet of M will be $\Omega = \{0, 1, \flat\}$, with $\flat$ denoting the blank symbol. The set of states of M will be $Q_M = \{q_1, q_2, \ldots, q_z\}$ for some $z > 1$. The transition relation of M will be $\delta_M \subseteq \Sigma_M \times \Omega \times Q_M \times \Sigma_M \times \Omega \times Q_M \times D$, where $D = \{d_1, d_2, d_3, d_4\}$ is the alphabet of directions of the moves of M's head on its tape and the meanings of d_i's are the same as in Definition 1. An element of δ_M of form $(\sigma_i, x_j, q_k, \sigma_\ell, x_m, q_n, d_p)$ is read as *“machine M, reading σ_i at its input port and scanning x_j in state q_k sends σ_ℓ to its output port, writes x_m into the scanned cell, enters state q_n and moves its head in direction d_p”.*

For the purpose of simulation, we split the processing time of M into time intervals during which the space complexity of M remains unchanged. Thanks to our assumption on the left–boundedness of M's tape, the amount of the space consumed by M's computation increases by 1 when the head of M reads $\flat$, going past the rightmost rewritten cell on its tape. Within the intervals of unchanged space complexity M can clearly be simulated by a finite transducer. Let C_i be the finite transducer which is equivalent to M computing within space of size i. The idea of the simulation is then to encode the (tape) configurations of M into states of an autopoietic automaton A_i which, in its non–reproducing states, behaves as C_i. When M is to increase its space complexity, A_i switches to a reproduction mode and generates a new, “bigger” automaton A_{i+1} which in its non–reproducing states simulates C_{i+1}, etc. Thus, in fact A_i is a merger of two automata: one corresponds to C_i while the other — let us call it R — takes care of reproduction. The transition relations of both automata are written on A_i's program tape. In the reproduction mode, A_i reads the transition relation of C_i and, being controlled by R, A_i generates the code for C_{i+1} and appends to it the code of R again. Thus, the code of R remains unchanged in all A_is.

We will assume that the *tape configuration* of M is of form $\$w_1\$q\$w_2\$$, where $w_1, w_2 \in \Omega^* \cup \{\varepsilon\}$, $q \in Q_M$ and w_1 is the contents of M's tape to the left of the position of $M's$ head on M's tape, and w_2 is the contents of M's tape to the right of w_1. That is, M's head points to the first symbol of w_2 (which might be a blank symbol, $\flat$ in the case when $w_2 = \varepsilon$). The *length of the tape configuration* $\$w_1\$q\$w_2\$$ is $|w_1| + |w_2|$, i.e., the sum of lengths of w_1 and w_2, respectively.

A tape configuration of M in state q_j will be represented as a sequence $\$\{0,1\}^* \cup \{\varepsilon\}\$0^j\$\{0,1\}^* \cup \{\flat\}\$$, i.e., the states of M are represented in unary, the tape contents in binary. As mentioned above the tape configurations of M in the previous form will straightforwardly correspond to the states of an autopoietic automaton. This idea requires a slight change in the definition of an autopoietic automaton — so far the states of an autopoietic automaton have been expressed in unary on automaton's program tape. The newly proposed representation of states calls for introduction of a further separator symbol (\$) among the symbols of the automaton's tape alphabet. Also note that neither the symbol read from the input port nor the one written to the output port by M is included in the above defined notion of M's configuration — these two symbols will be represented explicitly in A_i's configuration.

Now we are ready to formulate and prove the next theorem.

Theorem 2. *Any nondeterministic interactive Turing machine can be simulated by a lineage of autopoietic automata.*

Sketch of the proof. Let M be a given nondeterministic interactive Turing machine. By induction on i we will construct a lineage $\{A_i\}$ of autopoietic automata such that each A_i will simulate M with tape configurations of length i

(or of space complexity i) and each A_{i+1} will be an offspring of A_i, for $i \geq 2$. As mentioned above the "program" of each A_i will consist of two parts. The first part describes the transition relation of C_i while the second one that of R. For technical reasons — viz. the necessity to begin with an automaton which is able to simulate both right and left moves of the M's head on a tape of length 2 we start our induction with $i = 2$. This case is captured by C_2 and corresponds to the beginning of M's computation and to its subsequent computations until the moment when the M's head is about to enter the 3-rd cell on its tape.

Consider a generic "instruction" of δ_M of form $(\sigma_i, x_j, q_k, \sigma_\ell, x_m, q_n, d_p)$ performed over a tape configuration t_1 and resulting into tape configuration t_2, with both configurations corresponding to a tape of length 2 and the head positioned either on cell 1 or 2. This is reflected in C_2's instructions of form $(\sigma_i, t_1, \sigma_\ell, t_2, d_p)$. All these instructions, i.e., the instructions for all $\sigma_i, \sigma_\ell \in \Sigma_M$, all t_1 and t_2, all $q_k, q_n \in Q_M$ and all $d_p \in D$ conformed with δ_M are written on the A_2's program tape encoded as shown in Definition 1.

Any instruction of M attempting to move the M's head to the right of the 2-nd cell (or in general: increasing the current space complexity) will lead to the reproduction of A_2. Let $(\sigma_i, x_j, q_k, \sigma_\ell, x_m, q_n, d_2)$ be the move of M performed over a tape configuration t_1 of length 2 and resulting in tape configuration t_2 of length 3, with the head in t_1 positioned on the second cell and the head in t_2 positioned on the 3-rd cell.

Then, when A_2 reads σ_i in state t_1, it recognizes that this is the case when the head will move to the right. Under this circumstance A_2 will write σ_ℓ to its output port and will enter a reproducing phase in which the program tape of A_3 will be generated using the code for R. First, the program for C_3 will be constructed by reading the program for C_2. Starting from the state which corresponds to tape configuration t_2, automaton C_3 must be able to simulate all moves of M in space of size 3. In order to do so C_3 must have basically the same instructions as C_2 had. However, these instructions must be adopted for the case of the longer Turing machine tape (which is now longer by 1 cell that could contain $\flat$, 0 or 1). The "new" instructions are generated from the "old" ones by making appropriate local changes to the latter. The new instructions are generated to A_2's output tape. The respective changes must be made for all one–symbol prolongments (i.e., for 0, 1, and $\flat$) of the (current) tape configurations of M. To generate all new instructions, several (but a fixed finite number, depending on δ_M) scans over the simulation code of C_2 are needed. No doubt that this is an algorithmic procedure which can be carried by a finite automaton R thanks to the fact that the "templates" for producing new instructions are available on the program tape of A_2. An extra provision is needed for capturing also transitions which so far have not been represented in C_2 since they could not come into action due to the small size of the M' tape at that time. The details of an actual design of R are left to the reader.

After the entire program for C_3 is generated, the program for R is appended to it. This finishes the generation of the entire output tape of A_2 and therefore A_2 undergoes a fission and finishes its activity. The output tape of A_2 becomes the program tape of A_3 and the simulation of M is taken over by A_3. As mentioned before, A_3 starts from the state corresponding to configuration t_2.

Now, assuming that we have any A_i for $i \geq 3$ simulating M on tape configurations of size i it is a straightforward matter to see that upon entering its reproducing phase A_i will produce A_{i+1} simulating M on tape configurations of size $i+1$, for any $i \geq 3$. In fact, the "induction step" itself is performed by the automata themselves, by their very design as described at the beginning of this proof. The size (measured in the number of states, or length of the program tape) of our automata grows exponentially in i, i.e., in the length of the tape configuration of M at the time interval in which M is simulated by the automaton at hand.

Note that during their reproducing phases the automata in $\mathcal{A}$ are neither reading nor producing any external symbols. Therefore it is clear that on any inputs the lineage of A_is realizes the same translation as M does. □

Putting the claims of both previous theorems together we get the following consequence:

Corollary 1. *The computational power of lineages of autopoietic automata is equal to the computational power of nondeterministic interactive Turing machines.*

4 The Evolution of Autopoietic Automata

At the close of Section 2 we have already mentioned that in principle a single nondeterministic autopoietic automaton can produce an infinite tree (or at least a lineage) of its descendants. In order that this can happen, it must hold that at least one of the offsprings of a given parent must "survive", i.e., it must reach a reproducing state on a given input. Thus, in addition to the automaton's functionality the existence of infinite lineages depends on the "availability" of the "right" inputs. A trivial solution would be supplying the same inputs as before to all offsprings of an automaton which has just reproduced. Since among its offsprings there is one identical to its parent, this strategy will lead to an infinite lineage of identical automata. Obviously, this is not a very interesting case since no evolution is involved.

Under the assumption that the "right" inputs are supplied to the respective automata at each level of the descendant tree we construct an autopoietic automaton among whose descendants all possible autopoietic automata will appear. For the brevity we will call the mode of purposefully supplying inputs which will cause the given automaton to reproduce (if such inputs exist at all) the *nondeterministic input mode*. This mode assumes that the data read by the automaton exist and are supplied to the input port in such an order that will eventually lead to the fission of the automaton at hand (note that this does not correspond to the case of darwinian evolution).

Theorem 3. *There exists an autopoietic automaton which, when working in a nondeterministic input mode, generates a descendant tree containing all autopoietic automata.*

Sketch of the proof. The idea is to construct a single automaton which generates a descendant tree in which all autopoietic automata are enumerated. That is, this automaton will generate offsprings (direct descendants) with syntactically correct transition relations of a bounded length which will increase with the depth of the tree on which the offsprings are located. If a generated transition relation happens to be a transition relation of an autopoietic automaton that reproduces on some input, then the nondeterministic input mode will guarantee that the automaton at hand will reproduce.

Let B be the automaton we are after. Its transition relation δ will consist of k five–tuples, for some $k > 0$ (see Definition 1) and will be written on the B's program tape in form of a binary code. This code will read the B's program tape tuple by tuple and copy the tape either faithfully or with some modifications. Call any sequence of zeros representing a state or a symbol of Σ in the representation of the transition relation of B as written on the B's program tape, a *segment.* A segment from the program tape will be transformed via δ into a segment on the output tape according to the following rules:

– when reading a symbol in a segment, B nondeterministically decides whether to copy or skip it; in the former case, it writes 0 on the output tape and in both cases it proceeds to the next symbol in the program tape;
– after reaching the end of a segment, B will nondeterministically decide whether to prolong the segment by one additional zero;
– after processing the last segment, B will nondeterministically decide whether to add one tuple more to the generated transition relation, and if so, B will generate it nondeterministically, respecting the syntax of the encoding stated in Definition 1;
– in any tuple, the direction d of the move of the head on the program tape is also a subject to a nondeterministic choice.

A separator between the segments (i.e., symbol 1) will be copied without any change.

Having done so, B enters the final reproducing state. In this way, a bounded number of offsprings of B is generated, one on each branch of the respective nondeterministic computation. Each offspring possesses a syntactically correct (encoding of a) transition relation, differing in all but one offspring in certain segments from δ and containing at most one tuple more. The number of offsprings is related to the length of the program tape of B (it is exponential in k). Not all offsprings are functionally different since any transition relation admits a number of equivalent representations. On the other hand, among the offsprings there are all automata with the transition relation consisting of k or $k + 1$ tuples.

Now, all these offsprings start to act on their own. Thanks to the nondeterministic input mode those automata that can in principle reach the final reproducing state will get such an input which will eventually lead to their reproduction, indeed. The automata which cannot reproduce on any input will either stuck in some state or fall into endless loops. In any case, some automata will reproduce and give birth to still larger automata. In the worst case this will be automata functionally identical to their parents, having an equivalent transition relation but larger by one tuple. These automata will reproduce on the same input as their parents did. Eventually, a sufficiently large automaton not appearing in the lineage of its predecessors will be generated having a functionally different transition function and reproducing on different inputs than its parent.

In such a way the evolutionary process will continue, generating among other automata also different automata of increasing size, covering increasingly larger part of the space of all autopoietic automata. □

In the previous theorem we answered positively the question whether there is an autopoietic automaton which under suitable inputs (supplied in a nondeterministic mode) leads to an unbounded evolution producing automata with increasingly complex computational behavior. Now we will pose in a sense a reverse question. Namely, we will ask whether we can decide, for any autopoietic automaton and any infinite input sequence, whether there is an infinite lineage of automata whose members are all descendants of a given automaton on a given input. This is the problem of *sustainable evolution*. We will show a negative answer:

Theorem 4. *The problem of sustainable evolution is undecidable.*

Sketch of the proof. Let A be an autopoietic automaton and let $\mathcal{S}$ be a potentially infinite sequence of inputs. In accordance with the results from Section 3 for each lineage starting by A there is a nondeterministic interactive Turing machine simulating that lineage on $\mathcal{S}$. Now it is obvious that the sustainability problem can be transformed to the halting problem which is undecidable. □

It is interesting to compare the two previous results. While the first one assures that there is an autopoietic automaton which, when “fed” by proper inputs, will give rise to an unbounded evolution, the second result points to the fact that in general we cannot decide whether an autopoietic automaton will give birth to an infinite lineage of offsprings, under the given input. Thus, sustainability seems to require either adaptation of machines to their environment, or changes in the environment enabling the machines to survive, or both.

5 Conclusions

We have presented a novel model of offspring–producing automata based on the notion of finite state transducers. This model allows studies of the algorithmic variability of information controlling the computational behavior and replication of automata. This is achieved by enhancing the functional abilities of a standard transducer by allowing it to read its own transition relation and based on it, to generate a transition relation of its offsprings. The transition relation of an autopoietic automaton contains programs both for automaton's information processing tasks and for controlling its own evolution (via its offsprings). This idea allows a fresh look at the mechanisms of variability and inheritance of the "genetic information" passed from the parent to its offspring. Namely, in our model the driving force behind the evolution is an algorithmic procedure which itself can become a subject of an evolution driven by itself, so to say. In contrast to many previous approaches and speculations the evolution in our model is not based primarily on random mutations of randomly chosen parts of the controlling code, but on mutations which are algorithmically directed to those parts of the code which can bring only syntactically correct changes in programs controlling both the computational and evolutionary activities of the automaton at hand. The results showing the equivalency with the interactive nondeterministic Turing machines (Corollary 1) point to the great computing power of the lineages of autopoietic automata. There exist autopoietic automata which under suitable input conditions could give rise to unbounded complexity growth along the lineages of offsprings of such automata (Theorem 3). This offers a positive answer to the related open problem in the domain of artificial life. On the other hand, Theorem 4 shows the fragility of such phenomena — in general one cannot decide whether a lineage will evolve infinitely under given input conditions. The potential of our model in artificial life modelling is the subject of author's ongoing research.

References

1. McMullin, B.: John von Neumann and the Evolutionary Growth of Complexity: Looking Backwards, Looking Forwards... Artificial Life, Vol 6. Issue 4, Fall 2000, pp. 347-361
2. McMullin, B.: Thirty Years of Computational Autopoiesis: A Review. Artificial Life, Vol. 10, Issue 3, Summer 2004, pp. 277 - 296
3. van Leeuwen, J., Wiedermann, J: The Turing machine paradigm in contemporary computing, in: B. Enquist and W. Schmidt (Eds.), *Mathematics unlimited - 2001 and beyond*, Springer-Verlag, 2001, pp. 1139-1155.
4. van Leeuwen, J., Wiedermann, J.: Beyond the Turing limit: evolving interactive systems, in: L. Pacholski, P. Ružička (Eds.), *SOFSEM'01: Theory and Practice of Informatics*, 28th Conference on Current Trends in Theory and Practice of Informatics, Lecture Notes in Computer Science Vol. 2234, Springer-Verlag, Berlin, 2001, pp. 90–109.
5. von Neumann, J.: Theory of Selfreproducing Automata. A. Burks (Ed.), University of Illinois Press, Urbana and London, 1966

Lower Bounds on the Computational Power of an Optical Model of Computation

Damien Woods[1] and J. Paul Gibson[2]

[1] Boole Centre for Research in Informatics, School of Mathematics, University College Cork, Ireland
d.woods@bcri.ucc.ie
[2] Theoretical Aspects of Software Systems Research Group, Department of Computer Science, National University of Ireland, Maynooth, Ireland

Abstract. We present lower bounds on the computational power of an optical model of computation called the $\mathcal{C}_2$-CSM. We show that $\mathcal{C}_2$-CSM time is at least as powerful as sequential space, thus giving one of the two inclusions that are necessary to show that the model verifies the parallel computation thesis. Furthermore we show that $\mathcal{C}_2$-CSMs that simultaneously use polynomial space and polylogarithmic time decide at least the class NC.

1 Introduction

The computational model we study is relatively new and is called the continuous space machine (CSM) [6,7,8,14,15,16]. The CSM is inspired by classical Fourier optical computing architectures and uses complex-valued images, arranged in a grid structure, for data storage. The program also resides in images. The CSM has the ability to perform Fourier transformation, complex conjugation, multiplication, addition, thresholding and resizing of images. It has simple control flow operations and is deterministic. We analyse the model in terms of seven complexity measures inspired by real-world resources.

A rather general variant of the model was previously shown [14,16] to decide the membership problem for all recursively enumerable languages, and as such is unreasonable in terms of implementation. Also, the growth in resource usage was shown for each CSM operation, which in some cases was unreasonably large [14,15]. These results motivated the definition of the $\mathcal{C}_2$-CSM, a restricted CSM.

Recently [14] we have given upper and lower bounds on the computational power of the $\mathcal{C}_2$-CSM by showing that it verifies the parallel computation thesis. This thesis [2,3,4,5,9,12] states that parallel time corresponds, within a polynomial, to sequential space for reasonable parallel models. Furthermore we have characterised the class NC in terms of the $\mathcal{C}_2$-CSM [14].

Here we present one of the two inclusions that are necessary in order to verify the parallel computation thesis; we show that the languages accepted by nondeterministic Turing machines in $S(n)$ space are accepted by $\mathcal{C}_2$-CSMs computing in TIME $O(S(n) + \log n)^4$.

$$\text{NSPACE}(S(n)) \subseteq \mathcal{C}_2\text{-CSM–TIME}(O(S(n) + \log n)^4)$$

For example polynomial TIME $\mathcal{C}_2$-CSMs accept the PSPACE languages. Also we show that $\mathcal{C}_2$-CSMs that simultaneously use polynomial SPACE and polylogarithmic TIME accept the class NC.

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

These inclusions are established via $\mathcal{C}_2$-CSM simulation of index-vector machines.

2 The CSM

We begin by informally describing the model, this brief overview is not intended to be complete: Detailed definitions and discussions are to be found in [14,16].

Definition 1 (Image). *A complex-valued image, or simply an image, is a function $f : [0,1) \times [0,1) \to \mathbb{C}$, where $[0,1)$ is the half-open real unit interval.*

We let $\mathcal{I}$ denote the set of all complex-valued images. Let $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$ and $\mathbb{N} = \mathbb{N}^+ \cup \{0\}$. An address is an element of $\mathbb{N} \times \mathbb{N}$. For a given CSM M we let $\mathcal{N}$ be a countable set of images that encode M's addresses. Also for a given M there is an address encoding function $\mathfrak{E} : \mathbb{N} \to \mathcal{N}$ such that $\mathfrak{E}$ is Turing machine decidable, under some *reasonable* representation of images as words [14].

Definition 2 (CSM). *A CSM is a quintuple $M = (\mathfrak{E}, L, I, P, O)$, where*

$\mathfrak{E} : \mathbb{N} \to \mathcal{N}$ *is the address encoding function*
$L = ((s_\xi, s_\eta), (a_\xi, a_\eta), (b_\xi, b_\eta))$ *are the addresses: sta, a, and b,*
$I = ((\iota_{1_\xi}, \iota_{1_\eta}), \ldots, (\iota_{k_\xi}, \iota_{k_\eta}))$ *are the addresses of the k input images,*
$P = \{(\zeta_1, p_{1_\xi}, p_{1_\eta}), \ldots, (\zeta_r, p_{r_\xi}, p_{r_\eta})\}$ *are the r programming symbols and their addresses where* $\zeta_j \in (\{h, v, *, \cdot, +, \rho, st, ld, br, hlt\} \cup \mathcal{N}) \subset \mathcal{I}$,
$O = ((o_{1_\xi}, o_{1_\eta}), \ldots, (o_{l_\xi}, o_{l_\eta}))$ *are the addresses of the l output images.*

Each address is an element from $\{0, 1, \ldots, \Xi - 1\} \times \{0, 1, \ldots, \mathcal{Y} - 1\}$ where $\Xi, \mathcal{Y} \in \mathbb{N}^+$. Addresses a and b are distinct.

Addresses whose contents are not specified by P in a CSM definition are assumed to contain the constant image $f(x, y) = 0$. We interpret the above definition to mean that M is (initially) defined on a grid of images bounded by the constants Ξ and $\mathcal{Y}$, in the horizontal and vertical directions respectively.

In our grid notation the first and second elements of an address tuple refer to the horizontal and vertical axes of the grid respectively, and image $(0, 0)$ is at the lower left-hand corner of the grid. The images have the same orientation as the grid. Configurations are defined in a straightforward way as a tuple $\langle c, e \rangle$ where c is an address called the control and e represents the grid contents.

In Definition 2 the tuple P specifies the grid locations of programming symbols ζ_j that are from the (low-level) CSM programing language [14,16]. Here

$\mathrm{h}(i_1;i_2)$	: replace image at i_2 with horizontal 1D Fourier transform (FT) of i_1.
$\mathrm{v}(i_1;i_2)$	: replace image at i_2 with vertical 1D FT of image at i_1.
$*(i_1;i_2)$	: replace image at i_2 with the complex conjugate of image at i_1.
$\cdot\,(i_1,i_2;i_3)$	: pointwise multiply the two images at i_1 and i_2. Store result at i_3.
$+(i_1,i_2;i_3)$	: pointwise addition of the two images at i_1 and i_2. Store result at i_3.
$\rho(i_1,z_{\mathrm{l}},z_{\mathrm{u}};i_2)$	: filter the image at i_1 by amplitude using z_{l} and z_{u} as lower and upper amplitude threshold images, respectively. Place result at i_2.
$[\xi_1', \xi_2', \eta_1', \eta_2'] \leftarrow [\xi_1, \xi_2, \eta_1, \eta_2]$	: copy the rectangle of images whose bottom left-hand address is (ξ_1, η_1) and whose top right-hand address is (ξ_2, η_2) to the rectangle of images whose bottom left-hand address is (ξ_1', η_1') and whose top right-hand address is (ξ_2', η_2'). See illustration in Figure 2(c).

Fig. 1. CSM high-level programming language instructions. In these instructions $i, z_{\mathrm{l}}, z_{\mathrm{u}} \in \mathbb{N} \times \mathbb{N}$ and $\xi, \eta \in \mathbb{N}$. Control flow instructions are described in the text.

we introduce a less cumbersome, high-level, language. Figure 1 gives the basic instructions of this high-level language. There are also **if/else** and **while** control flow instructions with conditions of the form $(f_\psi == f_\phi)$ where f_ψ, f_ϕ are binary symbol images (see Figure 2(a)). Finally there are user defined functions. For convenience we write function input addresses before a ';' and function output addresses after the ';'. In Section 4 the reader will find example programs written in this programming language. See [14] for technical details and arguments to show that the low-level and high-level languages are equivalent.

Next we define some CSM complexity measures. All resource bounding functions map from $\mathbb{N}$ into $\mathbb{N}$ and have the usual properties [1].

Definition 3. *The* TIME *complexity of a CSM M is the number of configurations in the computation sequence of M, beginning with the initial configuration and ending with the first final configuration.*

Definition 4. *The* GRID *complexity of a CSM M is the minimum number of images, arranged in a rectangular grid, for M to compute correctly on all inputs.*

For example suppose M accepts language L, then the GRID complexity of M is the minimum number of images accessible by M and arranged in a rectangular grid, such that M accepts exactly L.

Let $S : \mathcal{I} \times (\mathbb{N} \times \mathbb{N}) \rightarrow \mathcal{I}$, where $S(f(x, y), (\Phi, \Psi))$ is an image, with $\Phi\Psi$ constant-valued pixels arranged in Φ columns and Ψ rows, that approximates f. If we choose a reasonable and realistic S then the details of S are unimportant.

Definition 5. *The* SPATIALRES *complexity of a CSM M is the minimum $\Phi\Psi$ such that if each image $f(x, y)$ in the computation of M is replaced with $S(f(x, y), (\Phi, \Psi))$ then M computes correctly on all inputs.*

Definition 6. *The* DYRANGE *complexity of a CSM M is the ceiling of the maximum of all the amplitude values stored in all of M's images during M's computation.*

In previous treatments we also defined the complexity measures AMPLRES, PHASERES and FREQ [14,15,16]. AMPLRES and PHASERES are measures of the cardinality of discrete amplitude and phase values respectively of the complex numbers in the range of CSM images. In the present work AMPLRES and PHASERES both have constant value of 2 (due to Definition 8) which means that all images are of the form $f : [0,1) \times [0,1) \to \{0, \pm\frac{1}{2}, \pm 1, \pm\frac{3}{2}, \ldots\}$. Furthermore we are studying a restricted CSM to which FREQ does not apply. Often we wish to make analogies between space on some well-known model and CSM 'space-like' resources. For this purpose we define the following convenient term.

Definition 7. *The* SPACE *complexity of a CSM M is the product of all of M's complexity measures except* TIME.

We have defined the complexity of a computation (sequence of configurations) for each measure. We extend this definition to the complexity of a (possibly non-final) configuration in the obvious way. Also, we sometimes talk about the complexity of an image, this is simply the complexity of the configuration that the image is in. More details on the complexity measures are be found in [16].

In [14,15] we defined the $\mathcal{C}_2$-CSM, a restricted class of CSM.

Definition 8 ($\mathcal{C}_2$-CSM). *A $\mathcal{C}_2$-CSM is a CSM whose computation* TIME *is defined for $t \in \{1, 2, \ldots, T(n)\}$ and has the following restrictions:*

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

3 Index-Vector Machines and Representations

Here we introduce vector machines, and the variant that we simulate called index-vector machines. We then describe our image representation of vectors.

The vector machine model was originally described by Pratt, Rabin and Stockmeyer [10], here we mostly use the conventions of Pratt and Stockmeyer [11]. A vector V is a binary sequence that is infinite to the left only and is *ultimately constant* (after a finite number of bits every bit to the left is either always 0 or always 1). The length of V, denoted $|V|$, is the length of the non-ultimately constant part of V. An ultimately 0 sequence represents a positive number and an ultimately 1 sequence represents a negative number [11,1]. The non-constant part represents a positive binary integer in the usual way, with the rightmost vector bit representing the least significant integer bit. The negative integer $-n$ is represented by the bitwise complement of the vector representing n. A vector machine (program) is a list of instructions where each is of the form given in the following definition.

Definition 9 (Vector machine instructions and their meanings [1]).

Vector instruction	*Meaning*
$V_i := x$	*Load the positive constant binary number x into vector V_i.*
$V_k := \neg V_i$	*Bitwise parallel negation of vector V_i.*
$V_k := V_i \wedge V_j$	*Bitwise parallel 'and' of two vectors.*
$V_k := V_i \uparrow V_j$	*If V_j is ultimately* 0 *(resp.* 1*) then shift V_i to the left (resp. right) by the distance given by the binary number v_j and store the result in V_k. If $V_j = 0$ then V_i is copied to V_k.*
$V_k := V_i \downarrow V_j$	*If V_j is ultimately* 0 *(resp.* 1*) then shift V_i to the right (resp. left) by the distance given by the binary number v_j and store the result in V_k. If $V_j = 0$ then V_i is copied to V_k.*
goto m *if* $V_i = 0$	*If $V_i = 0$ then branch to the instruction labelled m.*
goto m *if* $V_i \neq 0$	*If $V_i \neq 0$ then branch to the instruction labelled m.*

Instructions are labelled to facilitate the goto instruction. Configurations, (accepting) computations and computation time are all defined in the obvious way. Computation space is the maximum over all configurations, of the sum of the lengths of the vectors in each configuration. A language accepting vector machine on input w has an input vector of the form ...000w where $w \in 1\{0,1\}^*$. In this work we consider only deterministic vector machines. See [1] for details.

Definition 10 (Index-vector machines [11]). *A vector machine is of class $\mathcal{V}_I$ (equivalently, an index-vector machine) if its registers are partitioned into two disjoint sets, one set called index registers and the other called vector registers, such that (i) each Boolean operation in the program involves either only index registers or only vector registers; and (ii) each shift instruction is of the form*

$$V_1 := V_2 \uparrow I, \quad V_1 := V_2 \downarrow I, \quad I := J \uparrow 1, \quad I := J \downarrow 1$$

where V_1 and V_2 are vector registers, and I and J are index registers. For language recognition the input register is a vector register.

It is straightforward to prove the following lemma by induction on t.

Lemma 1 ([11]). *Given index-vector machine $M \in \mathcal{V}_I$ with n as the maximum input length, there is a constant c such that vector length in index (respectively vector) registers is bounded above by $c+t$ (respectively $2^{c+t}+n$) after t timesteps.*

Pratt and Stockmeyer's [11] main result is a characterisation of the power of index-vector machines. The characterisation is described by two inclusions, proved for time bounded index-vector machines and space bounded Turing machines:

$$\text{NSPACE}(S(n)) \subseteq \mathcal{V}_I\text{–TIME}(O(S(n) + \log n)^2) \tag{1}$$

$$\mathcal{V}_I\text{–TIME}(T(n)) \subseteq \text{DSPACE}(O(T(n)(T(n) + \log n))) \tag{2}$$

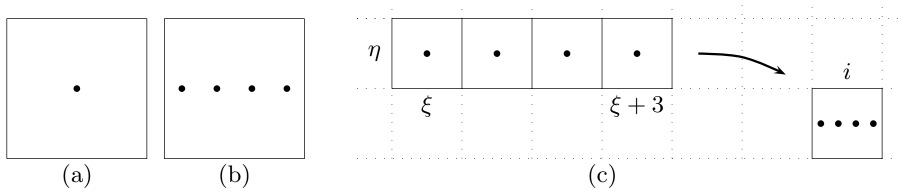


Fig. 2. Representing data by images. To represent the value ψ, the black point has value ψ. The white area denotes value zero. (a) *Binary symbol image* f_ψ where $\psi \in \{0,1\}$, or *number image* where $\psi \in \{0, \pm\frac{1}{2}, \pm 1, \pm\frac{3}{2}, \ldots\}$, (b) *binary list image* where $\psi \in \{0,1\}$. (c) Illustration of the instruction $i \leftarrow [\xi, \xi+3, \eta, \eta]$ that copies four images to a single image that is denoted i.

In other words, index-vector machines verify the parallel computation thesis and are a member of the second machine class [12]. Modulo a polynomial, deterministic and nondeterministic vector machines have equal power [10].

3.1 Image Representation of Vectors

Let $v_i \in \{0,1\}^*$ denote the non-'ultimately constant' part of vector V_i. If the ultimately constant part of V_i is 0^ω (respectively 1^ω) then let $\mathrm{sign}(v_i) = 0$ (respectively let $\mathrm{sign}(v_i) = 1$). In this work we use binary symbol images, number images and binary list images. These represent binary symbols, numbers from $\{0, \pm\frac{1}{2}, \pm 1, \pm\frac{3}{2}, \ldots\}$, and binary words in a straight-forward way that is illustrated in Figure 2. Further details are to be found in [14,16].

The vector V_i is represented by three images: $\overline{v_i}$, $\overline{|v_i|}$ and $\overline{\mathrm{sign}(v_i)}$. The image $\overline{v_i}$ is the binary list image representation of v_i. Image $\overline{|v_i|}$ is the natural number image represention of $|v_i|$ (the length of v_i). Accessing these images respectively incurs SPATIALRES and DYRANGE costs that are linear in $|v_i|$. Image $\overline{\mathrm{sign}(v_i)}$ is f_0 (the binary symbol image representing 0) if $\mathrm{sign}(v_i) = 0$ and f_1 if $\mathrm{sign}(v_i) = 1$. We use the same representation scheme for vector program constants. The simulation uses natural number images as addresses, which are clearly reasonable in the sense of the $\mathcal{C}_2$-CSM definition. Hence addressing incurs a (linear) DYRANGE cost. The three images types are illustrated in Figures 2(a) and 2(b).

Another issue to consider is the layout of the grid of images; where to place input, program constants (f_0, f_1, f_{-1}, $f_{\frac{1}{2}}$, f_2), local variables, etc. There are only a constant number of such images hence there a number of layouts that work, a specific grid layout is given in [14]. Rows 0 and 1 are used to store temporary images. The only images explicitly referred to by numerical addresses are in these two rows (all other addresses have identifier names from the outset).

4 $\mathcal{C}_2$-CSM Simulation of Index-Vector Machines

In this section we prove that $\mathcal{C}_2$-CSMs are at least as powerful as index-vector machines (up to a polynomial in time). More precisely

$$\mathcal{V}_I\text{–TIME}(T(n)) \subseteq \mathcal{C}_2\text{-CSM–TIME}(O(T^2(n))). \qquad (3)$$

To prove this we simulate each index-vector machine instruction in $O(\log |V_{\max}|)$ TIME where $|V_{\max}| \in \mathbb{N}$ is the maximum length of (the non-ultimately constant part of) any of the vectors mentioned in the instruction. Additionally we simulate the index-vector shifts in linear TIME. From Lemma 1 this TIME bound ensures that our overall simulation executes in quadratic TIME, which is sufficient for the inclusion given by Equation (3). The SPACE bound on the simulation is $O(|V_{\max}|^3)$. Some of the simulations are merely sketched, full proofs are to be found in [14].

We begin by giving a straightforward simulation of vector assignment.

Theorem 1 ($V_i := x$)**.** *The vector machine assignment instruction $V_i := x$ is simulated by a $\mathcal{C}_2$-CSM in $O(1)$ TIME, $O(1)$ GRID, $O(|x|)$ DYRANGE and $O(\max(|x|, |v_i|))$ SPATIALRES.*

Proof. The images representing x are simply copied to those representing V_i:

```
assignment($\overline{x}$, $\overline{|x|}$, $\overline{\mathrm{sign}(x)}$; $\overline{v_i}$, $\overline{|v_i|}$, $\overline{\mathrm{sign}(v_i)}$)
  $\overline{v_i} \leftarrow \overline{x}$
  $\overline{|v_i|} \leftarrow \overline{|x|}$
  $\overline{\mathrm{sign}(v_i)} \leftarrow \overline{\mathrm{sign}(x)}$
end // assignment
```

We require $O(\max(|x|, |v_i|))$ SPATIALRES to represent x and v_i as binary list images. DYRANGE of $O(|x|)$ is needed to represent $|x|$ as a natural number image. No address goes beyond the initial grid limits hence we use constant GRID. □

A $\mathcal{C}_2$-CSM can quickly generate a list image g, where each list element is identical. We state the following lemma for the specific case that each list element is a binary symbol image f_ψ. By simply changing the value of one input, the algorithm generalises to arbitrary repeated lists (with a suitable change in resource use, dependent only on the complexity of the new input image element).

Lemma 2 (generate_list$(f_\psi, l; g)$)**.** *A list image g that contains l list elements, each of which is a copy of input binary image f_ψ, is generated in $O(\log l)$ TIME, $O(l)$ GRID, SPATIALRES and DYRANGE.*

Proof (Sketch). The algorithm horizontally juxtaposes two copies of f_ψ and rescales them to a single image. This juxtaposing and rescaling is repeated on the new image; the process is iterated a total of $\lceil \log l \rceil$ times to give a list of length $2^{\lceil \log l \rceil}$, giving the stated TIME bound. In constant TIME, the list image is then stretched to its full length across $2^{\lceil \log l \rceil}$ images, l juxtaposed images are then selected and rescaled to a single output image g. $O(l)$ SPATIALRES is necessary to store the list in a single image. $O(l)$ GRID is used to stretch the list out to its full length. Recall that we are using natural number images for addresses, hence $O(l)$ DYRANGE is used to stretch the list across $2^{\lceil \log l \rceil}$ images. □

Theorem 2 ($V_k := \neg V_i$)**.** *The vector machine negation instruction $V_k := \neg V_i$ is simulated by a $\mathcal{C}_2$-CSM in $O(\log |v_i|)$ TIME, $O(|v_i|)$ GRID and DYRANGE, and $O(\max(|v_k|, |v_i|))$ SPATIALRES.*

```
$\neg(\overline{v_i}, \overline{|v_i|}, \overline{\text{sign}(v_i)}; \overline{v_k}, \overline{|v_k|}, \overline{\text{sign}(v_k)})$
    generate_list($f_{-1}$, $\overline{|v_i|}$; list_neg_ones)                    // generate list of $-1$s
    $\cdot$ ($\overline{v_i}$,list_neg_ones;$\overline{-v_i}$)                     // change each 1 in $\overline{v_i}$ to $-1$
    generate_list($f_1$, $\overline{|v_i|}$; list_ones)                          // generate list of 1s
    $+$($\overline{-v_i}$,list_ones;$\overline{v_k}$)        // change $-1$s to 0s, 0s to 1s, & place in $\overline{v_k}$
    **if** ( $\overline{\text{sign}(v_i)}$ == $f_1$ ) **then** $\overline{\text{sign}(v_k)} \leftarrow f_0$
    **else** $\overline{\text{sign}(v_k)} \leftarrow f_1$ **end if**
end // $\neg$
```

Program 4.1. Simulation of $V_k := \neg V_i$

Proof. Program 4.1 simulates $V_k := \neg V_i$. The program generates a list of -1s of length $|v_i|$. This list image is then multiplied by $\overline{v_i}$; changing each 1 in $\overline{v_i}$ to -1 and leaving each 0 unchanged. Then we add 1 to each element in the resulting list. A simple **if** statement negates $\overline{\text{sign}(v_i)}$. Each call to the function generate_list($\cdot$) requires $O(\log|v_i|)$ TIME, otherwise TIME is constant. The remaining resource usages are for accessing vectors and rescaling them to their full length. □

The proof of the following straightforward lemma gives a program that decides which of two vectors is the longer in constant TIME. It also shows that we can decide the max or min of two integer images in constant TIME.

Lemma 3 (max($\cdot$) **and** min($\cdot$))**.** *The* max *(or* min*) length of the vectors V_i and V_j is decided in* $O(1)$ TIME*,* $O(1)$ GRID*,* $O(\max(|v_i|, |v_j|))$ SPATIALRES*,* $O(\max(|v_i|, |v_j|))$ DYRANGE*.*

Proof (Sketch). The function header for max($\cdot$) is formatted as follows:

max($\overline{v_i}$, $\overline{|v_i|}$, $\overline{\text{sign}(v_i)}$, $\overline{v_j}$, $\overline{|v_j|}$, $\overline{\text{sign}(v_j)}$; longest, |longest|, sign(longest))

The encoding of $-|v_i|$ is created by the instruction $\cdot$ ($\overline{|v_i|}$,f_{-1};$\overline{-|v_i|}$), then the max($\cdot$) algorithm thresholds the value $|v_j| - |v_i|$ to the range $[0, 1]$. If the result is the zero image f_0 then V_i is the longer vector and its representation is copied to the three output addresses, else the representation of V_j is output. In a similar way we decide the min length of two vector images, the function header for min($\cdot$) has the format:

min($\overline{v_i}$, $\overline{|v_i|}$, $\overline{\text{sign}(v_i)}$, $\overline{v_j}$, $\overline{|v_j|}$, $\overline{\text{sign}(v_j)}$; shortest, |shortest|, sign(shortest)) □

Theorem 3 ($V_k := V_i \wedge V_j$)**.** *The vector machine instruction $V_k := V_i \wedge V_j$ is simulated by a $\mathcal{C}_2$-CSM in* $O(\log\max(|v_i|, |v_j|))$ TIME*,* $O(\max(|v_i|, |v_j|, |v_k|))$ SPATIALRES*, and* $O(\max(|v_i|, |v_j|))$ GRID *and* DYRANGE*.*

Proof. Program 4.2 simulates $\wedge$. It uses multiplication of vector images to simulate $V_i \wedge V_j$ in parallel. However if $|v_i| \neq |v_j|$, we first pad the shorter vector image with zeros so that both have equal length. To find the longer and shorter of the two vectors we make use of the max($\cdot$) and min($\cdot$) routines given above.

```
$\wedge$ ($\overline{v_i}$, $\overline{|v_i|}$, $\overline{\text{sign}(v_i)}$, $\overline{v_j}$, $\overline{|v_j|}$, $\overline{\text{sign}(v_j)}$; $\overline{v_k}$, $\overline{|v_k|}$, $\overline{\text{sign}(v_k)}$ )
    max($\overline{v_i}$, $\overline{|v_i|}$, $\overline{\text{sign}(v_i)}$, $\overline{v_j}$, $\overline{|v_j|}$, $\overline{\text{sign}(v_j)}$; longest, |longest|, sign(longest))
    min($\overline{v_i}$, $|v_i|$, sign($v_i$), $\overline{v_j}$, $|v_j|$, sign($v_j$); shortest, |shortest|, sign(shortest))
    **if** ( sign(longest) == $f_1$ ) **then**
        $\cdot$ ($f_{-1}$,|shortest|;−|shortest|)
        +(longest,−|shortest|;difference)
        generate_list($f_1$, difference; pad)
        [1,|shortest|,1,1] ← shortest
        +(|shortest|,$f_1$;|shortest|+1)
        [|shortest|+1,|longest|,1,1] ← pad
        padded_shortest ← [1,|longest|,1,1]
    **else**
        [1,|longest|,1,1] ← $f_0$
        [1,|shortest|,1,1] ← shortest
        padded_shortest ← [1,|longest|,1,1]
    **end if**
    $\cdot$ (longest,padded_shortest;$\overline{v_k}$)          // a single multiplication simulates $v_i \wedge v_j$
    $\cdot$ (sign(longest),sign(shortest);$\overline{\text{sign}(v_k)}$)
    $\overline{|v_k|}$ ← |longest|
end // $\wedge$
```

Program 4.2. Simulation of $V_k := V_i \wedge V_j$

The program requires $O(\log \max(|v_i|, |v_j|))$ TIME for the generate_list(·) call (the worst case is when exactly one of the vectors is of length 0). The remainder of the program runs in $O(1)$ TIME, including determining which vector is longer, padding of the shorter vector and parallel multiplication of vectors. The remaining resource usages on vector images in the theorem statement are for accessing and storing to a single image, and stretching to full length. □

Next we give algorithms to simulate vector left shift and right shift. The main idea is to copy large numbers of images to simulate shifting.

Lemma 4 (left_shift($n, \overline{v_i}, \overline{|v_i|}, \overline{\text{sign}(v_i)}; \overline{v_k}, \overline{|v_k|}, \overline{\text{sign}(v_k)}$))**.** *A left shift of distance $n \geqslant 0$ on a vector V_i, to create vector V_k, is simulated in $O(1)$* TIME*, $O(|v_i + n|)$* GRID *and* DYRANGE*, and $O(\max(|v_i + n|, |v_k|))$* SPATIALRES*.*

Proof (Sketch). The algorithm assumes that n is given as a natural number image. We simulate the shift by stretching $\overline{v_i}$ out to its full length, placing n zero images to the right of the stretched $\overline{v_i}$, and then selecting all of $\overline{v_i}$ along with the n zeros and rescaling back to one image. After the shift (in accordance with the definition of vector shift), 0s are to be placed in the rightmost positions. □

An algorithm for right_shift(·) would work similarly. However this time we select the leftmost $|v_i| - n$ images of the stretched $\overline{v_i}$. If $n \geqslant |v_i|$ the output is the representation of the zero vector.

```
$\uparrow$ ($\overline{v_i}$, $\overline{|v_i|}$, $\overline{\text{sign}(v_i)}$, $\overline{v_j}$, $\overline{|v_j|}$, $\overline{\text{sign}(v_j)}$; $\overline{v_k}$, $\overline{|v_k|}$, $\overline{\text{sign}(v_k)}$ )
  shift_distance $\leftarrow f_0$
  current_bit $\leftarrow \overline{|v_j|}$
  current_power_2 $\leftarrow f_1$
  [1,$\overline{|v_j|}$,0,0] $\leftarrow \overline{v_j}$
  $\rho$($\overline{|v_j|}$,$f_0$,$f_1$;flag)
  **while** ( flag == $f_1$ ) **do**
    **if** ( $\overline{\text{sign}(v_j)}$ == $f_0$ ) **then**
      **if** ( [current_bit,current_bit,0,0] == $f_1$ ) **then**
        +(shift_distance,current_power_2;shift_distance)
      **end if**
    **else**
      **if** ( [current_bit,current_bit,0,0] == $f_0$ ) **then**
        +(shift_distance,current_power_2;shift_distance)
      **end if**
    **end if**
    $\cdot$ (current_power_2,$f_2$;current_power_2)
    +(current_bit,$f_{-1}$;current_bit)
    $\rho$(current_bit,$f_0$,$f_1$;flag)
  **end while**
  **if** ( $\overline{\text{sign}(v_j)}$ == $f_0$ ) **then**
    left_shift(shift_distance, $\overline{v_i}$, $\overline{|v_i|}$, $\overline{\text{sign}(v_i)}$; $\overline{v_k}$, $\overline{|v_k|}$, $\overline{\text{sign}(v_k)}$)
  **else** right_shift(shift_distance, $\overline{v_i}$, $\overline{|v_i|}$, $\overline{\text{sign}(v_i)}$; $\overline{v_k}$, $\overline{|v_k|}$, $\overline{\text{sign}(v_k)}$) **end if**
end // $\uparrow$
```

Program 4.3. Simulation of $V_k := V_i \uparrow V_j$

Theorem 4 **($V_k := V_i \uparrow V_j$).** *The vector machine instruction $V_k := V_i \uparrow V_j$ is simulated by a $\mathcal{C}_2$-CSM in $O(|v_j|)$ TIME, $O(|v_i|+2^{|v_j|})$ GRID and DYRANGE, and $O(\max(|v_k|, |v_i| + 2^{|v_j|}))$ SPATIALRES.*

Proof. Program 4.3 simulates the shift by stretching V_i out to its full length; then selecting either part of V_i, or V_i and some extra zero images; and finally rescaling back to one image. The simulator's addresses are represented by natural number images whereas vectors are represented by binary list images. In order to perform the stretching the program converts the binary number defined by V_j to a natural number image called shift_distance.

The **while** loop efficiently generates a value of $O(2^{|v_j|})$ in $O(|v_j|)$ TIME. At different stages of the algorithm each of $\overline{v_i}$ and $\overline{v_j}$ are rescaled to their full length, across $|v_i|$ and $|v_j|$ images respectively. We get the value $O(|v_i| + 2^{|v_j|})$ for GRID since in the worst case V_i is left shifted by the value $2^{|v_j|}$, and (when stretched) the resulting vector spans $O(|v_i| + 2^{|v_j|})$ images. This upper bound also covers the right shift case (when V_j is negative). Analogously we get the same value for SPATIALRES and DYRANGE (except $|v_k|$ is also in the SPATIALRES expression as it could contain some values before the program executes). □

The converse shift instruction ($V_k := V_i \downarrow V_j$) is simulated by Program 4.3 except that the calls to right_shift($\cdot$) and left_shift($\cdot$) are exchanged. The resource usage remains the same.

The proof of the following lemma gives a log TIME algorithm to decide if a list or vector image represents a word that consists only of zeros. It is possible to give a constant TIME algorithm that makes use of the FT (to 'sum' the entire list in constant TIME). Not using the FT enables us to state Corollary 6.

Lemma 5. *A $\mathcal{C}_2$-CSM that does not use Fourier transformation decides whether or not a list (equivalently vector) image $\overline{v_i}$ represents the word $0^{|v_i|}$ in $O(\log |v_i|)$ TIME, $O(|v_i|)$ GRID, SPATIALRES and DYRANGE.*

Proof (Sketch). The binary list image $\overline{v_i}$ is padded with zeros so that is of length $2^{\lceil \log |v_i| \rceil}$. The algorithm splits $\overline{v_i}$ into a left half and a right half, adds both halves (in a one step parallel pointwise fashion), and repeats until the list is of length 1. A counter image keeps track of list length. The resulting image is thresholded below by f_0 and above by f_1. If the result is the zero image then $\overline{v_i}$ represents a list of zeros, otherwise $\overline{v_i}$ represents a list with at least one 1. □

Theorem 5 (goto m if $V_i = 0$)**.** *The vector machine instruction* goto m if $V_i = 0$ *(or* goto m if $V_i \neq 0$*) is simulated by a $\mathcal{C}_2$-CSM in $O(\log |v_i|)$ TIME, $O(|v_i|)$ GRID, SPATIALRES, and DYRANGE.*

Proof. Due to the vector machine number representation, there are exactly two representations for 0; the constant sequences $\ldots 000$ and $\ldots 111$. Using our $\mathcal{C}_2$-CSM representation of vectors, if $\overline{|v_i|} = 0$ then the vector V_i is constant, and hence represents 0. We can test $\overline{|v_i|} = \overline{0}$ in constant time with an **if** statement.

However, it may be the case that $\overline{|v_i|} = n > 0$ and yet $\overline{V_i}$ represents 0. In this case $\overline{v_i}$ represents a list of 0s (respectively 1s) and $\overline{\text{sign}(v_i)}$ represents 0 (respectively 1). A sequential search through $\overline{v_i}$ will require exponential TIME (worst case) and as such is too slow. Instead we use the log TIME technique given by the previous lemma. In the case that V_i is ultimately 1 we make use of the $\neg(\cdot)$ program defined in Theorem 2. For the goto part of the instruction we merely note that gotos are simulated by **if**s and **while**s. Clearly the related instruction 'goto m if $V_i \neq 0$' is simulated with the same resource usage. □

Given a vector machine M there is a $\mathcal{C}_2$-CSM M' that simulates M. In particular, if vector machine M decides a language L then we can easily modify our simulation of vector machines so that M' decides L.

Theorem 6. *Let M be an index-vector machine that decides $L \in \{0,1\}^*$ in time $T(n)$ for input length n. Then L is decided by a $\mathcal{C}_2$-CSM M' in $O(T^2(n))$ TIME, $O(2^{T(n)})$ GRID, SPATIALRES and DYRANGE.*

Proof. By Lemma 1 M's index-vectors have length $O(T(n))$, while unrestricted vectors have length $O(2^{T(n)})$. From the above simulation theorems, any non-shifting instruction is simulated in TIME that is log of the length of the vectors. The remaining operations, right and left shift, are simulated in TIME that is linear in the length of their index-vector input. From these bounds it is straightforward to work out that M decides L in $O(T^2(n))$ TIME and that each of GRID, SPATIALRES and DYRANGE is $O(2^{T(n)})$. □

From the previous theorem M' uses $O(2^{3T(n)})$ SPACE to decide L, hence our simulation uses SPACE that is cubic in the space of M.

Corollary 1. $\mathcal{V}_I$–TIME$(T(n)) \subseteq \mathcal{C}_2$-CSM–TIME$(O(T^2(n)))$

Let $S(n) = \Omega(\log n)$. From the inclusion in Equation (1) we get:

Corollary 2. NSPACE$(S(n)) \subseteq \mathcal{C}_2$-CSM–TIME$(O(S(n) + \log n)^4)$

Combining this result with the upper bound on TIME bounded $\mathcal{C}_2$-CSM power [14]:

Corollary 3. NSPACE$(S(n)) \subseteq \mathcal{C}_2$-CSM–TIME$(O(S(n) + \log n)^4)$
$\subseteq$ DSPACE$(O(S(n) + \log n)^8))$

To summarise, the $\mathcal{C}_2$-CSM satisfies the parallel computation thesis:

Corollary 4. NSPACE$(S^{O(1)}(n)) = \mathcal{C}_2$-CSM–TIME$(S^{O(1)}(n))$

This links space bounded sequential computation and TIME bounded $\mathcal{C}_2$-CSM computation. For example $\mathcal{C}_2$-CSM–TIME$(n^{O(1)})$ = PSPACE. We strengthen this result by restricting the $\mathcal{C}_2$-CSM. Let a 1D-$\mathcal{C}_2$-CSM be a $\mathcal{C}_2$-CSM with constant GRID and SPATIALRES, in one of the vertical or the horizontal directions.

Corollary 5. *The* 1D-$\mathcal{C}_2$-CSM *verifies the parallel computation thesis.*

Proof. The index-vector machine simulation used only constant GRID and SPATIALRES in the vertical direction. Moreover we can rotate the grid layout and all images by 90°, to obtain a simulation where GRID and SPATIALRES are constant in the horizontal direction only. □

Corollary 6. *The* $\mathcal{C}_2$-CSM *without the DFT operations* h *and* v *verifies the parallel computation thesis.*

Proof. Our $\mathcal{C}_2$-CSM simulation of index-vector machines did not use h nor v. □

The thesis relates parallel time to sequential space, however in our simulations we explicitly gave *all* resource bounds. As a final result we show that the class of $\mathcal{C}_2$-CSMs that simultaneously use polynomial SPACE and polylogarithmic TIME decide at least the languages in NC. Let $\mathcal{C}_2$-CSM–SPACE, TIME$(S(n), T(n))$ be the class of languages decided by $\mathcal{C}_2$-CSMs that use SPACE $S(n)$ and TIME $T(n)$. It is known [5] that $\mathcal{V}_I$–SPACE, TIME$(n^{O(1)}, \log^{O(1)} n)$ = NC. From the resource overheads in our simulations:

$$\begin{aligned}&\mathcal{V}_I\text{–SPACE, TIME}(O(2^{T(n)}), T(n))\\ &\subseteq \mathcal{C}_2\text{-CSM–SPACE, TIME}(2^{O(T(n))}, T^{O(1)}(n))\end{aligned}$$

For the case of $T(n) = \log^{O(1)} n$ we have our final result.

Corollary 7. NC $\subseteq \mathcal{C}_2$-CSM–SPACE, TIME$(n^{O(1)}, \log^{O(1)} n)$

In [14] it was shown that the converse inclusion also holds.

5 Discussion

We have given lower bounds on $\mathcal{C}_2$-CSM power in terms of space and NC complexity classes via simulation of index-vector machines. The quadratic time bound for index-vector simulation is reasonably tight. In a certain sense this is not surprising; both are SIMD models. Possibly the power in Corollary 2 could be reduced from 4 to 2 by direct Turing machine simulation.

The simulation uses the reasonable (we argue) natural number representation of images. This incurs a dyRange cost that is a constant times the longest vector. Since the binary values in vectors are represented by images with constant dyRange, it would be interesting if another addressing scheme could be employed that works (say) on binary values (e.g. binary list image addresses). We believe that dyRange could be reduced without a significant increase in the other measures. Simultaneously, a constant grid simulation might be possible, the main problem is to simulate vector shifts while using at most constant grid. By using these trade-offs we conjecture that space can be reduced to be linear in index-vector machine space, with only a polynomial increase in time.

In addition to fulfilling our needs of giving a lower bound on $\mathcal{C}_2$-CSM power, the results in this paper are useful to the practitioner since we have given a method to directly translate vector machine algorithms to optical algorithms.

Interestingly we did not make use of Fourier transformation in the simulation. Optical computers are often celebrated for having a constant time FT operation. Our results prove that the $\mathcal{C}_2$-CSM has remarkable power without explicitly using the FT. However in a more fine grained analysis, say using the $\mathcal{C}_2$-CSM to design algorithms for NC and AC problems, some advantages of Fourier transformation would be observed.

Acknowledgements

We thank Tom Naughton for interesting discussions. The first author is funded by the Irish Research Council for Science, Engineering and Technology.

References

1. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity, vols I and II.* EATCS Monographs on Theoretical Computer Science. Springer, Berlin, 1988.
2. A. K. Chandra and L. J. Stockmeyer. Alternation. In *In 17th annual symposium on Foundations of Computer Science*, pages 98–108, Houston, Texas, Oct. 1976. IEEE. Preliminary Version.
3. L. M. Goldschlager. *Synchronous parallel computation.* PhD thesis, University of Toronto, Computer Science Department, Dec. 1977.
4. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to parallel computation: P-completeness theory.* Oxford university Press, Oxford, 1995.
5. R. M. Karp and V. Ramachandran. *Parallel algorithms for shared memory machines.* Volume A of van Leeuwen [13], 1990.

6. T. J. Naughton. Continuous-space model of computation is Turing universal. In S. Bains and L. J. Irakliotis, editors, *Critical Technologies for the Future of Computing*, Proceedings of SPIE vol. 4109, San Diego, California, Aug. 2000.
7. T. J. Naughton. A model of computation for Fourier optical processors. In R. A. Lessard and T. Galstian, editors, *Optics in Computing 2000*, Proc. SPIE vol. 4089, pages 24–34, Quebec, Canada, June 2000.
8. T. J. Naughton and D. Woods. On the computational power of a continuous-space optical model of computation. In M. Margenstern and Y. Rogozhin, editors, *Machines, Computations and Universality: Third International Conference*, volume 2055 of *Lecture Notes in Computer Science*, pages 288–299, Chişinău, Moldova, May 2001. Springer.
9. I. Parberry. *Parallel complexity theory.* Wiley, 1987.
10. V. R. Pratt, M. O. Rabin, and L. J. Stockmeyer. A characterisation of the power of vector machines. In *Proc. 6th annual ACM symposium on theory of computing*, pages 122–134. ACM press, 1974.
11. V. R. Pratt and L. J. Stockmeyer. A characterisation of the power of vector machines. *Journal of Computer and Systems Sciences*, 12:198–221, 1976.
12. P. van Emde Boas. *Machine models and simulations.* Volume A of van Leeuwen [13], 1990.
13. J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume A. Elsevier, Amsterdam, 1990.
14. D. Woods. *Computational complexity of an optical model of computation.* PhD thesis, National University of Ireland, Maynooth, 2004. Submitted.
15. D. Woods and J. P. Gibson. Complexity of continuous space machine operations. In S. B. Cooper, B. Löewe, and L. Torenvliet, editors, *New Computational Paradigms, First Conference on Computability in Europe*, volume 3526 of *Lecture Notes in Computer Science*, pages 540–551, Amsterdam, June 2005. Springer.
16. D. Woods and T. J. Naughton. An optical model of computation. *Theoretical Computer Science*, 334(1–3):227–258, Apr. 2005.

On Counterfactual Computation

Paolo Zuliani

Department of Computer Science,
Princeton University,
Princeton, NJ 08544, USA
pzuliani@cs.princeton.edu

Abstract. In this paper we pursue two targets. First, showing that counterfactual computation can be rigorously formalised as a quantum computation. Second, presenting a new counterfactual protocol which improve previous protocols. Counterfactual computation makes use of quantum mechanics' peculiarities to infer the outcome of a quantum computation without running that computation. In this paper, we first cast the definition of counterfactual protocol in the quantum programming language qGCL, thereby showing that counterfactual computation is an example of quantum computation. Next, we formalise in qGCL a probabilistic extension of counterfactual protocol for decision problems (whose result is either 0 or 1). If p_G^r denotes for protocol G the probability of obtaining result r "for free" (*i.e.* without running the quantum computer), then we show that for any probabilistic protocol $p_G^0 + p_G^1 \leq 1$ (as for non-probabilistic protocols). Finally, we present a probabilistic protocol K which satisfies $p_K^0 + p_K^1 = 1$, thus being optimal. Furthermore, the result is attained with a single insertion of the quantum computer, while it has been shown that a non-probabilistic protocol would obtain the result only in the limit (*i.e.* with an infinite number of insertions).

1 Introduction

Counterfactuality is the fact that the sole possibility for an event to occur allows one to gain some information about that event, even though it did not actually occur. Counterfactual computation [4,5] uses peculiar features of quantum mechanics to infer counterfactual statements about the result of a computation. In particular, it is possible to devise methods for probabilistically inferring the outcome of a computation without actually running the computation: the mere fact that the quantum computer implementing that computation might have run is sufficient.

One of the first examples of counterfactuality was given by Elitzur and Vaidman [1] with the so-called *interaction-free* measurements. That technique allows determining the presence of an object by means of a test particle, possibly with no "interaction" occurring between the object and the test particle. A potential application of this technique is the acquisition of the image of an object without any light or other radiation interacting with the object (see [8] for example). If one replaces the object with a quantum computer implementing some computation C and the test particle with the computer's "switch", it is then possible to know the outcome of computation C without the computer ever being turned on. This application of quantum mechanics is known as counterfactual computation and it was firstly introduced by Jozsa [4] and then further formalised by Mitchison and Jozsa [5].

The aim of this paper is two fold:

1. to show how a programming language, qGCL [7], can be used for rigorously describing and reasoning about counterfactual computation, thereby embedding it in a more general framework, which in particular includes classical, probabilistic and quantum computation;
2. to present a new example of counterfactual computation, along with its proof of correctness in qGCL. In particular, we consider a probabilistic extension of counterfactual computation.

We assume the reader has some knowledge of the basics of quantum computing.

2 Quantum Programming

We give here a short presentation of the features of qGCL (a full introduction can be found in [7]). qGCL has been used to describe and reason about all known quantum algorithms and to derive the Deutsch-Jozsa algorithm from its specification. The problem of compiling qGCL code has been studied in [9].

2.1 Quantum Types

We define the type $\mathbb{B} \mathrel{\widehat{=}} \{0,1\}$, which we will treat as booleans or bits, depending on convenience. A classical register of size $n{:}\mathbb{N}$ is a vector of n booleans. The type of all registers of size n is then defined to be the set of boolean-valued functions on $\{0,1,\ldots,n-1\}$:

$$\mathbb{B}^n \mathrel{\widehat{=}} \{0,1,\ldots,n-1\} \longrightarrow \mathbb{B}\,.$$

The quantum analogue of $\mathbb{B}^n$ is the set of complex-valued functions on $\mathbb{B}^n$ whose squared modulus sum to 1:

$$q(\mathbb{B}^n) \mathrel{\widehat{=}} \{\chi{:}\mathbb{B}^n \longrightarrow \mathbb{C} \quad | \sum_{x:\mathbb{B}^n} |\chi(x)|^2 = 1\}\,.$$

An element of $q(\mathbb{B})$ is called a *qubit* and that of $q(\mathbb{B}^n)$ a *qureg*. Classical state is embedded in its quantum analogue by the Dirac delta function:

$$\begin{aligned} &\delta{:}\mathbb{B}^n \longrightarrow q(\mathbb{B}^n) \\ &\delta_x(y) \mathrel{\widehat{=}} (y = x)\,. \end{aligned}$$

The range of δ, $\{\delta_x \mid x{:}\mathbb{B}^n\}$, forms a *basis* for quantum states, that is:

$$\forall \chi{:}q(\mathbb{B}^n) \bullet \chi = \sum_{x:\mathbb{B}^n} \chi(x)\delta_x\,.$$

The Hilbert space $\mathbb{B}^n \longrightarrow \mathbb{C}$ (with the structure making it isomorphic to $\mathbb{C}^{2^n}$) is called the *enveloping space* of $q(\mathbb{B}^n)$. The usual scalar product becomes the application $\langle\cdot,\cdot\rangle{:}q(\mathbb{B}^n)\times q(\mathbb{B}^n)\rightarrow\mathbb{C}$ defined by:

$$\langle\psi,\phi\rangle \mathrel{\widehat{=}} \sum_{x:\mathbb{B}^n}\psi(x)^*\phi(x)$$

where z^* is the complex conjugate of $z{:}\mathbb{C}$. The *norm* of ψ is $\|\psi\| \mathrel{\widehat{=}} \langle\psi,\psi\rangle^{\frac{1}{2}}$.

2.2 Quantum Language qGCL

qGCL is an extension of pGCL [6], which in turn extends Dijkstra's guarded-command language with a probabilistic choice constructor in order to address probabilism. The BNF syntax for qGCL is as follows:

$$\begin{aligned}
\langle \mathit{qprogram}\rangle &::= \langle \mathit{qstatement}\rangle \{\, ; \langle \mathit{qstatement}\rangle \}\\
\langle \mathit{qstatement}\rangle &::= \chi := \langle \mathit{unitary\ op}\rangle(\chi) \mid\\
&\quad\ \mathbf{Fin}(\langle \mathit{identifier}\rangle, \langle \mathit{identifier}\rangle, \langle \mathit{identifier}\rangle) \mid\\
&\quad\ \mathbf{In}(\langle \mathit{identifier}\rangle) \mid\\
&\quad\ \mathbf{skip} \mid x := e \mid \langle \mathit{loop}\rangle \mid \langle \mathit{conditional}\rangle \mid\\
&\quad\ \langle \mathit{nondeterministic\ choice}\rangle \mid\\
&\quad\ \langle \mathit{probabilistic\ choice}\rangle \mid \langle \mathit{local\ block}\rangle\\
\chi &::= \langle \mathit{identifier}\rangle\\
\langle \mathit{loop}\rangle &::= \mathbf{while}\ \langle \mathit{cond}\rangle\ \mathbf{do}\ \langle \mathit{qstatement}\rangle\ \mathbf{od}\\
\langle \mathit{cond}\rangle &::= \langle \mathit{boolean\ expression}\rangle\\
\langle \mathit{conditional}\rangle &::= \langle \mathit{qstatement}\rangle \lhd \langle \mathit{cond}\rangle \rhd \langle \mathit{qstatement}\rangle\\
&\quad\ \text{executes the LHS if predicate } \langle \mathit{cond}\rangle \text{ holds}\\
\langle \mathit{nondeterministic\ choice}\rangle &::= \langle \mathit{qstatement}\rangle \sqcap \langle \mathit{qstatement}\rangle\\
\langle \mathit{probabilistic\ choice}\rangle &::= \langle \mathit{qstatement}\rangle\ {}_p\!\oplus \langle \mathit{qstatement}\rangle\\
&\quad\ \text{executes LHS (RHS) with probability } p\,(1-p)\\
\langle \mathit{local\ block}\rangle &::= \mathbf{var}\ \bullet \langle \mathit{qstatement}\rangle\ \mathbf{rav}
\end{aligned}$$

where without loss of clarity we omitted the formal definitions of $\langle \mathit{identifier}\rangle$ and $\langle \mathit{boolean\ expression}\rangle$; $\langle \mathit{unitary\ op}\rangle(\chi)$ is just some mathematical expression involving qureg χ - such expression should of course denote a unitary operator.

Probabilistic choice may be written using a prefix notation, in case the branches are more than two. Let $[\ (p_j, r_j) \mid 0 \leqslant j < m\ ]$ be a finite indexed family of (program, number) pairs with $\sum_j r_j = 1$, then the probabilistic choice in which p_j is chosen with probability r_j is written in prefix form: $\oplus[\ p_j\ @\ r_j \mid 0 \leqslant j < m\ ]$.

Initialisation is a procedure which simply assigns to its qureg state the uniform square-convex combination of all standard states

$$\forall\chi{:}q(\mathbb{B}^n)\bullet\mathbf{In}(\chi) \mathrel{\widehat{=}} \left(\chi := \frac{1}{\sqrt{2^n}}\sum_{x:\mathbb{B}^n}\delta_x\right).$$

Quantum-mechanical systems evolve over time under the action of *unitary* transformations. *Evolution* thus consists of iteration of unitary transformations on quantum state. Evolution of qureg χ under unitary operator U is described via the assignment $\chi := U(\chi)$. The well-known *no-cloning* theorem forbids any assignment $\chi := U(\psi)$ if (syntactically) $\chi \neq \psi$.

The content of a qureg can be read (measured) through quantum procedure *Finalisation* and suitable *observables.* An observable is defined from a family of pairwise orthogonal subspaces which together span the enveloping space of the qureg being read. Let $\mathcal{O}$ be an observable defined by the family of pairwise orthogonal subspaces $\{S_j \mid 0 \leqslant j < m\}$. In our notation we write $\mathbf{Fin}(\mathcal{O}, i, \chi)$ for the measurement of $\mathcal{O}$ on a quantum system described by state $\chi{:}q(\mathbb{B}^n)$, where i stores the result determining the subspace to which state χ is reduced. Finalisation is entirely defined using the probabilistic combinator of pGCL (see [7] for an unabridged treatment); in our notation we write:

$$\mathbf{Fin}\,(\mathcal{O}, i, \chi) \mathrel{\widehat{=}} \oplus \left[\left(i, \chi := j, \frac{P_{S_j}(\chi)}{\|P_{S_j}(\chi)\|} \right) @ \langle \chi, P_{S_j}(\chi) \rangle \;\mid 0 \leqslant j < m \right]$$

where P_{S_j} is the projector onto subspace S_j. We denote by Δ the observable spanned by the computational basis, also known as *diagonal* measurement.

Semantics for pGCL (and in turn for qGCL) can be given either relationally [3] or in terms of expectation transformers [6]. We shall use the latter, due to its simplicity in calculations. Expectation-transformer semantics is a probabilistic extension of the predicate-transformer one. In predicate-transformer semantics a transformer maps post-conditions to their weakest pre-conditions. Analogously, an expectation transformer represents a computation by mapping post-expectations to their greatest pre-expectations. We shall retain the *wp* prefix notation of predicate-transformer calculus for convenience and we denote the greatest pre-expectation of post-expectation q on program P by $wp.P.q$. For a standard predicate p we denote by $[p]$ its embedding into expectation transformers: the greatest pre-expectation $wp.P.[p]$ is then the *maximum guaranteed probability* that p holds after the execution of P.

In the Appendix we briefly review expectation-transformer semantics and some associated programming laws used in this paper.

3 Counterfactual Computation

3.1 An Example

We begin by giving the simple example of counterfactual computation introduced by Jozsa [4]. Suppose we are given a decision problem (*i.e.* a problem with a binary solution, "yes" or "no") and a quantum computer Q with an "on-off" switch programmed to solve that problem when the switch is set to "on". Therefore we need a qubit to represent the switch and another qubit for the result of the computation. The computer might need an extra qureg to use during its functioning, but we assume that Q works reversibly, so that at the end of the computation the output will be placed in the output register as bit-wise XOR. We map "off/on" and "no/yes" to δ_0/δ_1, so for example δ_{00} means switch at "off" and result "no". The functioning of the computer is thus:

$$Q(\delta_{ij}) \mathrel{\widehat{=}} \delta_{i(j \oplus r)}$$

where $\oplus$ is bit-wise XOR and r is the result of the problem. We suppose that the computation takes at most a finite time T.

From Table 1 we see that for $r = 0$ the computer behaves as the identity transform, *i.e.* "do nothing", while for $r = 1$ it behaves as the well known CNOT transform over the switch and output qubits. By assuming that switch and output are encoded by qureg χ, we can readily model Q by program QC:

$$\begin{aligned} &QC(\chi) \mathrel{\widehat{=}} (QC_0(\chi) \mathbin{\Box} QC_1(\chi)) \\ &QC_0(\chi) \mathrel{\widehat{=}} \mathbf{skip}\ , \quad QC_1(\chi) \mathrel{\widehat{=}} \chi := CNOT(\chi) \end{aligned}$$

thus representing our ignorance about the inner working of the computer and the result of the decision problem. The goal is to start with the switch "off" and, after at least a time T, to determine which operation **skip** or CNOT has been performed, without setting the switch to "on".

Table 1. Functioning of the quantum computer Q

$r = 0$	$r = 1$	
$\delta_{00} \to \delta_{00}$	$\delta_{00} \to \delta_{00}$	switch "off"
$\delta_{01} \to \delta_{01}$	$\delta_{01} \to \delta_{01}$	switch "off"
$\delta_{10} \to \delta_{10}$	$\delta_{10} \to \delta_{11}$	switch "on"
$\delta_{11} \to \delta_{11}$	$\delta_{11} \to \delta_{10}$	switch "on"

Consider the following program N:

$$N \mathrel{\widehat{=}} [\ \chi := \delta_{00} \mathbin{⨟} \chi := H \otimes \mathbb{1}(\chi) \mathbin{⨟} QC(\chi) \mathbin{⨟} \chi := H \otimes \mathbb{1}(\chi)\ ]$$

where $\chi{:}q(\mathbb{B}^2)$, $\mathbb{1}$ is the identity transform over qubits, and H is the single-qubit Hadamard transform defined as:

$$\begin{aligned} &H{:}q(\mathbb{B}) \to q(\mathbb{B}) \\ &H(\delta_x) \mathrel{\widehat{=}} \frac{1}{\sqrt{2}}(\delta_0 + (-1)^x \delta_1)\ . \end{aligned}$$

We show that if the result of the problem is 1, then N can probabilistically infer it with probability $\frac{1}{4}$, without running the computer. We reason on program N:

N

$=$ law A-2, definition of H

$\chi := \frac{1}{\sqrt{2}}(\delta_{00} + \delta_{10}) \mathbin{⨟} QC(\chi) \mathbin{⨟} \chi := H \otimes \mathbb{1}(\chi)$

$$= \quad \text{definition of } QC \text{ and law A-3}$$

$$[\chi := \text{CNOT}(\tfrac{1}{\sqrt{2}}(\delta_{00} + \delta_{10}))] \,\Box\, [\chi := \tfrac{1}{\sqrt{2}}(\delta_{00} + \delta_{10}) \,\fatsemi\, \textbf{skip}] \,\fatsemi\, \chi := H \otimes \mathbb{1}(\chi)$$

$$= \quad \text{definition of CNOT and } \textbf{skip} \text{ identity}$$

$$[\chi := \tfrac{1}{\sqrt{2}}(\delta_{00} + \delta_{11})] \,\Box\, [\chi := \tfrac{1}{\sqrt{2}}(\delta_{00} + \delta_{10})] \,\fatsemi\, \chi := H \otimes \mathbb{1}(\chi)$$

$$= \quad \text{laws S-3, A-3}$$

$$[\chi := H \otimes \mathbb{1}(\tfrac{1}{\sqrt{2}}(\delta_{00} + \delta_{11}))] \,\Box\, [\chi := H \otimes \mathbb{1}(\tfrac{1}{\sqrt{2}}(\delta_{00} + \delta_{10}))]$$

$$= \quad \text{definition of } H$$

$$[\chi := \tfrac{1}{2}(\delta_{00} + \delta_{01} + \delta_{10} - \delta_{11})] \,\Box\, [\chi := \delta_{00}]$$

Suppose we now measure χ in the standard basis: if $r = 0$ (*i.e.* RHS of the nondeterministic choice) we always measure 00 and, because with probability $\frac{1}{4}$ we may measure 00 when $r = 1$, we cannot reliably infer the result of the computation. Suppose now $r = 1$: with probability $\frac{1}{4}$ we measure 10 and we know for sure that 1 is the result of the problem. Furthermore, the computer has not run, because if it had the output register (initially set to 0) should display 1. Therefore, if $r = 1$, with probability $\frac{1}{4}$ we learn the result of the problem without running the computer! The output 10 is thus a *counterfactual outcome.* Finally, with probability $\frac{1}{4}$ each we measure 01 and 11, thereby learning that $r = 1$, but the computer has run (the output register has changed).

3.2 Formal Definition

We now code in qGCL the definition of *protocol* given by Mitchison and Jozsa [5]. For a datatype D we denote by *seq*(D) the datatype of finite sequences of elements of type D.

Definition 1. *A* **protocol** *G is a terminating program of the following type:*

$$G \mathrel{\widehat{=}} \textbf{var}\ \chi{:}q(\mathbb{B}^n), o{:}seq(\mathbb{B}^n), \psi{:}q(\mathbb{B}^p), s{:}\mathbb{B}^p \bullet\ body \,\fatsemi\, \textbf{Fin}(\Delta, s, \psi)\ \textbf{rav}$$

where:

1. *body, according to Mitchison and Jozsa [5], is* "a sequence of steps where each step is one of the following:
   (a) A unitary operation (not involving the computer) on a finite number of specified qubits.
   (b) A measurement on a finite number of specified qubits.
   (c) An 'insertion of the computer' QC, where the state of two selected qubits is swapped into the switch and output registers of the computer."
2. *o returns the list of outcomes of the measurements of steps of type (b).*

In order to formally describe what we mean by saying that the computer has not run, we procede as follows. After each insertion of QC we project the state over two orthogonal subspaces, the "off" and "on" subspaces, by entangling it with a qubit from ψ. That transformation is defined as:

$$E{:}q(\mathbb{B}^n) \to q(\mathbb{B}^{n+1})$$
$$E(v) \mathrel{\widehat{=}} (P_{\delta_0} \otimes \mathbb{1})v \otimes \delta_0 + (P_{\delta_1} \otimes \mathbb{1})v \otimes \delta_1.$$

We note that we need as many qubits as the number of insertions of the computer. The action of E is thus to create two coherent superpositions of the state vector, one 'living' in the off subspace, the other living in the on subspace. By measuring ψ at the end of the computation we can recognise a computation which has always taken place in the (desirable) off subspace: in that case ψ would reduce to δ_{0^p} (equivalently, s is the p-bit string 0, an "all-off" string). Our method is equivalent to the graphical *history* approach of Mitchison and Jozsa [5]: o and s collectively denote a history. The advantage of our approach is that it embeds all the necessary concepts in a single, general-purpose programming formalism.

We are now ready to formalise the definition of counterfactual computation in qGCL.

Definition 2. *Given a protocol G, a sequence $m{:}seq(\mathbb{B}^n)$ is a* counterfactual outcome *of type $r{:}\mathbb{B}$ if the following two conditions hold:*

$$\text{(1)}\quad \forall c{:}\mathbb{B}^m \bullet wp.G_r.[\ o = m, s = c\ ] = 0 \quad \textit{iff} \quad c \neq 0$$
$$\text{(2)}\quad wp.G_{1-r}.[\ \textstyle\sum_{\chi_i:M} \chi_i = 0\ ] = 1$$

where M is the set of state vectors for which $o = m$, and G_r denotes protocol G when $QC = QC_r$, i.e. *only operation QC_r is performed by the computer.*

Condition (1) states that if QC_r is used in the protocol, then m is seen iff the (only) computation leading to it has always stayed in the off subspace. Therefore we can infer the result (r) of the problem for "free", since the switch of the quantum computer was always found at off. Condition (2) states that when QC_{1-r} is used, then all the computations leading to m annihilate themselves, by means of the so-called *destructive interference.* That implies $wp.G_{1-r}.[o = m] = 0$, *i.e.* m never occurs (the converse needs not to hold) and we are sure that the result of the problem is r.

3.3 Limits on Counterfactual Computation

In this section we show how qGCL can be effective in reasoning about counterfactual computation. We begin by noting that a sequence of measurement outcomes cannot be a counterfactual outcome of type 0 and 1 at the same time. That is, if we define $CF_G(t)$ as the set of counterfactual outcomes of type t for protocol G, then the sets $CF_G(0)$ and $CF_G(1)$ are disjoint.

Theorem 1. *For any protocol G we have that $CF_G(0) \cap CF_G(1) = \emptyset$.*

Proof. A contradiction arises between condition (2) for $m{:}CF_G(0)$ and condition (1) for $m{:}CF_G(1)$. □

Let p^0_G and p^1_G denote the probability of learning "for free" outcomes 0 and 1 respectively, in a protocol G; in our notation we write:

$$\forall r{:}\mathbb{B} \bullet p^r_G \mathrel{\widehat{=}} \sum_{m:seq(\mathbb{B}^n)} wp.G_r.[\ o = m, s = 0\ ].$$

The sum is well defined as G always terminates. One may wish to design a protocol for which both p^0_G and p^1_G are greater than 0 and perhaps $p^0_G + p^1_G > 1$. Mitchison and Jozsa [5] stopped further conjectures, showing that for any protocol the two probabilities sum to at most 1. We replay here their result in our formalism.

Theorem 2 (Mitchison and Jozsa). *For any protocol G we have:*

$$p^0_G + p^1_G \leqslant 1\,.$$

Proof. We reason:

$$\begin{aligned}
&p^0_G + p^1_G \\
=\ & \hfill \text{definition of } p^r_G \\
&\textstyle\sum_{t:\mathbb{B}} \sum_{m:seq(\mathbb{B}^n)} wp.G_r.[\ o = m, s = 0\ ] \\
=\ & \hfill \text{logic and definition of } CF_G(\cdot) \\
&\textstyle\sum_{m:CF_G(0)} wp.G_0.[\ o = m, s = 0\ ] + \sum_{l:CF_G(1)} wp.G_1.[\ o = l, s = 0\ ] \\
=\ & \hfill \text{in the off subspace } wp.G_0 = wp.G_1 \\
&\textstyle\sum_{m:CF_G(0)} wp.G_0.[\ o = m, s = 0\ ] + \sum_{l:CF_G(1)} wp.G_0.[\ o = l, s = 0\ ] \\
=\ & \hfill \text{Theorem 1} \\
&\textstyle\sum_{m:CF_G(0)} wp.G_0.[\ o = m, s = 0\ ] \\
\leqslant\ & \hfill \text{logic} \\
&\textstyle\sum_{m:seq(\mathbb{B}^n)} wp.G_0.[\ o = m, s = 0\ ] \\
=\ & \hfill wp\text{-semantics and 1 top element} \\
&wp.G_0.[\ s = 0\ ] \leqslant 1
\end{aligned}$$

□

Mitchison and Jozsa also derived complexity constraints between p^r_G and the number of times the computer is inserted. They showed that for any protocol G such that $p^0_G + p^1_G = 1 - \epsilon$, the number of insertions of the computer must necessarily tend to infinity as ϵ tends to 0.

4 Probabilistic Extension

In this section we formalise in qGCL a probabilistic extension of counterfactual computation proposed by Mitchison and Jozsa [5]. In particular, we consider the case in which we allow a relaxation of condition (1) of definition 2, while (2) is carried over intact.

Definition 3. *Given a protocol G, a sequence $m{:}seq(\mathbb{B}^n)$ is an* approximate counterfactual outcome *of type $r{:}\mathbb{B}$ if the following two conditions hold:*

$$\begin{array}{ll} (1') & wp.G_r.[\ o = m, s \neq 0\ ] < \epsilon \\ (2') & wp.G_{1-r}.[\ \sum_{\chi_i:M} \chi_i = 0\ ] = 1 \end{array}$$

where ϵ is a small real in the $(0,1]$ interval.

Condition $(1')$ implies that, when using QC_r in the protocol, m may arise from a computation which does not lie in the off subspace, *i.e.* the computer has run throughtout the protocol. However, the probability of such an event is bounded by the small number ϵ. It is of course expected that the computation of the off subspace leading to m has probability greater than ϵ. Together, the two conditions ensure that when we see m the answer to the decision problem is r (because of $(2')$), and with high probability the computer has not run.

Probabilistic protocols (*i.e.* protocols which feature approximate counterfactual outcomes) face some of the limitations of non-probabilistic ones.

Theorem 3. *For any probabilistic protocol G we must have $p_G^0 + p_G^1 \leq 1$.*

Proof. The proof of Theorem 2 still applies, as condition $(2')$ is what ensures that an outcome can be only be measured under either G_r or G_{1-r}. □

However, probabilistic protocols do not require an infinite number of insertions of the computer in order to reach the limit 1. The next example shows that a single insertion suffices.

4.1 Probabilistic Protocol

We describe a probabilistic protocol which can infer the answer to the decision problem with certainty, but requires a run of the quantum computer with probability $\frac{1}{2}$. We first draft the functioning of the protocol in words, then we code it in qGCL, and we finally prove its correctness. Again, for simplicity we write the state of the switch and of the output qubits as a single qureg.

We start with the switch and output register in the equally-weighted superposition of standard states, that is $\chi = \frac{1}{2}\sum_{i:\mathbb{B}^2}\delta_i$. Then we perform phase inversion on state δ_{11}, thus giving $\chi = \frac{1}{2}(\delta_{00} + \delta_{01} + \delta_{10} - \delta_{11})$. We apply the quantum computer:

$$\chi = \begin{cases} v_0 \mathrel{\widehat{=}} \frac{1}{2}(\delta_{00} + \delta_{01} + \delta_{10} - \delta_{11}) & \text{if } r = 0 \\ v_1 \mathrel{\widehat{=}} \frac{1}{2}(\delta_{00} + \delta_{01} + \delta_{11} - \delta_{10}) & \text{if } r = 1 \end{cases}$$

We now measure χ using observable $\{\mathbb{C}v_0, \mathbb{C}v_1, (\mathbb{C}v_0 \oplus \mathbb{C}v_1)^\perp\}$ where $\mathbb{C}v_i$ is the unidimensional spaces spanned by v_i and $\perp$ denotes orthogonality. Since $v_0 \perp v_1$, we are thus able to learn r with certainty and we do not perturb state χ. A subsequent measurement of the switch qubit reduces χ to its off subspace (*i.e.* switch set to 0) with probability $\frac{1}{2}$.

In qGCL the protocol is coded as follows:

$$\begin{aligned}
K \mathrel{\widehat{=}} &\ \textbf{var}\ \chi{:}q(\mathbb{B}^2), \psi{:}q(\mathbb{B}), o{:}\{0,1,2\}, s{:}\mathbb{B}\ \bullet\\
&\quad \textbf{In}(\chi)\,;\\
&\quad \chi := T_{\delta_{11}}(\chi)\,;\\
&\quad QC(\chi)\,;\\
&\quad \chi, \psi := E(\chi)\,;\\
&\quad \textbf{Fin}(\mathcal{V}, o, \chi \otimes \psi)\,;\\
&\quad \textbf{Fin}(\mathcal{S}, s, \chi \otimes \psi)\\
&\ \textbf{rav}
\end{aligned}$$

where:

1. for function $f{:}\mathbb{B}^n \to \mathbb{B}$ between registers, unitary transformation T_f between quregs inverts χ (pointwise) about 0 if f holds and otherwise leaves it unchanged
$$\begin{aligned}
&T_f{:}q(\mathbb{B}^n) \to q(\mathbb{B}^n)\\
&(T_f\chi)(x) \mathrel{\widehat{=}} (-1)^{f(x)}\chi(x)
\end{aligned}$$
2. observable $\mathcal{V} \mathrel{\widehat{=}} \{V_0, V_1, V_2\}$ has $V_i \mathrel{\widehat{=}} \mathbb{C}E(v_i)$ for $i{:}\mathbb{B}$, and $V_2 \mathrel{\widehat{=}} (V_0 \oplus V_1)^\perp$
3. observable $\mathcal{S} \mathrel{\widehat{=}} \{S_0, S_1\}$ has $S_i \mathrel{\widehat{=}} \mathbb{C}^2 \otimes \mathbb{C}\delta_i$

Proposition 1. *Outcome $m{:}\mathbb{B}$ is an approximate counterfactual outcome of type m for protocol K. In particular, we have:*

$$\begin{aligned}
&\text{(a)} \quad wp.K_m.[o = m, s \neq 0] = \tfrac{1}{2}\\
&\text{(b)} \quad wp.K_{1-m}.[\ \textstyle\sum_{\chi_i:M} \chi_i = 0\ ] = 1
\end{aligned}$$

Proof. We reason directly on K:

$$\begin{aligned}
&K\\
&= \qquad\qquad \text{definition of } \textbf{In}\\
&\chi := \tfrac{1}{2}(\delta_{00} + \delta_{01} + \delta_{10} + \delta_{11})\,;\\
&\chi := T_{\delta_{11}}(\chi)\,;\\
&QC(\chi)\,;\\
&\chi, \psi := E(\chi)\,;\\
&\textbf{Fin}(\mathcal{V}, o, \chi \otimes \psi)\,;\\
&\textbf{Fin}(\mathcal{S}, s, \chi \otimes \psi)\\
&= \qquad\qquad \text{definition of } T_f \text{ and law A-2}
\end{aligned}$$

$$\begin{array}{l}\chi := v_0 ⨾\\ QC(\chi) ⨾\\ \chi, \psi := E(\chi) ⨾\\ \mathbf{Fin}(\mathcal{V}, o, \chi \otimes \psi) ⨾\\ \mathbf{Fin}(\mathcal{S}, s, \chi \otimes \psi)\end{array}$$

$=$ definition of QC and laws A-3, S-3

$$\begin{pmatrix}\chi := v_0 ⨾\\ \chi := \mathrm{CNOT}(\chi) ⨾ \chi, \psi := E(\chi)\end{pmatrix} \Box \begin{pmatrix}\chi := v_0 ⨾\\ \mathbf{skip} ⨾ \chi, \psi := E(\chi)\end{pmatrix} ⨾$$
$$\begin{array}{l}\mathbf{Fin}(\mathcal{V}, o, \chi \otimes \psi) ⨾\\ \mathbf{Fin}(\mathcal{S}, s, \chi \otimes \psi)\end{array}$$

$=$ law A-2, definition of CNOT and **skip** identity

$$\begin{array}{l}(\chi, \psi := E(v_1)) \Box (\chi, \psi := E(v_0)) ⨾\\ \mathbf{Fin}(\mathcal{V}, o, \chi \otimes \psi) ⨾\\ \mathbf{Fin}(\mathcal{S}, s, \chi \otimes \psi)\end{array}$$

$=$ law S-2

$$\begin{pmatrix}\chi, \psi := E(v_1) ⨾\\ \mathbf{Fin}(\mathcal{V}, o, \chi \otimes \psi) ⨾\end{pmatrix} \Box \begin{pmatrix}\chi, \psi := E(v_0) ⨾\\ \mathbf{Fin}(\mathcal{V}, o, \chi \otimes \psi) ⨾\end{pmatrix} ⨾$$
$$\mathbf{Fin}(\mathcal{S}, s, \chi \otimes \psi)$$

$=$ definition of **Fin**

$$\oplus \left[\begin{pmatrix}\chi, \psi := E(v_1) ⨾\\ o, \chi, \psi := j, \frac{P_{V_j}(\chi \otimes \psi)}{\|P_{V_j}(\chi \otimes \psi)\|}\end{pmatrix} @\langle \chi \otimes \psi, P_{V_j}(\chi \otimes \psi)\rangle \mid j{:}\{0,1,2\} \right] \Box$$
$$\oplus \left[\begin{pmatrix}\chi, \psi := E(v_0) ⨾\\ o, \chi, \psi := k, \frac{P_{V_k}(\chi \otimes \psi)}{\|P_{V_k}(\chi \otimes \psi)\|}\end{pmatrix} @\langle \chi \otimes \psi, P_{V_k}(\chi \otimes \psi)\rangle \mid k{:}\{0,1,2\} \right] ⨾$$
$$\mathbf{Fin}(\mathcal{S}, s, \chi \otimes \psi)$$

$=$ law A-2 and linear algebra

$$\begin{array}{l}(o, \chi, \psi := 1, E(v_1)) \Box (o, \chi, \psi := 0, E(v_0)) ⨾\\ \mathbf{Fin}(\mathcal{S}, s, \chi \otimes \psi)\end{array}$$

$=$ law S-2 and definition of **Fin** and E

$$\oplus \left[\begin{pmatrix}o, \chi, \psi := 1, \frac{1}{2}(\delta_{000} + \delta_{010} + \delta_{111} - \delta_{101}) ⨾\\ s, \chi, \psi := j, \frac{P_{S_j}(\chi \otimes \psi)}{\|P_{S_j}(\chi \otimes \psi)\|}\end{pmatrix} @\langle \chi \otimes \psi, P_{S_j}(\chi \otimes \psi)\rangle \mid j{:}\mathbb{B} \right] \Box$$
$$\oplus \left[\begin{pmatrix}o, \chi, \psi := 0, \frac{1}{2}(\delta_{000} + \delta_{010} + \delta_{101} - \delta_{111}) ⨾\\ s, \chi, \psi := k, \frac{P_{S_k}(\chi \otimes \psi)}{\|P_{S_k}(\chi \otimes \psi)\|}\end{pmatrix} @\langle \chi \otimes \psi, P_{S_k}(\chi \otimes \psi)\rangle \mid k{:}\mathbb{B} \right]$$

$=$ law A-2 and linear algebra

$$[(o,s,\chi,\psi := 1,0,\tfrac{1}{\sqrt{2}}(\delta_{000} + \delta_{010}))\ {}_{\frac{1}{2}}\!\oplus\ (o,s,\chi,\psi := 1,1,\tfrac{1}{\sqrt{2}}(\delta_{111} - \delta_{101}))]\ \square$$
$$[(o,s,\chi,\psi := 0,0,\tfrac{1}{\sqrt{2}}(\delta_{000} + \delta_{010}))\ {}_{\frac{1}{2}}\!\oplus\ (o,s,\chi,\psi := 0,1,\tfrac{1}{\sqrt{2}}(\delta_{101} - \delta_{111}))]$$

By inspection we can easily see that claims (a) and (b) are fully satisfied (K_1 is the LHS of the nondeterministic choice, while K_0 is the RHS). □

We observe that protocol K thus exhibits two counterfactual outcomes: 0 and 1. Mitchison and Jozsa also exhibited a protocol with two counterfactual outcomes which fully satisfies definition 2, and for which $p_0 = p_1 = 0.172$, thereby giving $p_0 + p_1 = 0.344$. However, the main advantage of protocol K is that it reaches the probability bound 1 with a single insertion of the computer, while a standard protocol reaches 1 only in the limit.

Proposition 2. *Protocol K is optimal, that is $p^0_K + p^1_K = 1$.*

Proof. It follows by Proposition 1 and Theorem 3. □

We note that K is also optimal with respect to the number of insertions of the computer, since at least one insertion is required by any protocol. In Table 2 we provide a summary of the features of standard and probabilistic counterfactual computation.

We conclude by providing another example of probabilistic protocol, which stems from protocol K. Of course it cannot improve protocol K, but it provides another example of protocol (and of quantum computation, in the end). We first recall the "inversion about the mean" transform introduced firstly by Grover in his search algorithm [2]. It is the unitary transform $M(\cdot)$ defined as:

$$M{:}q(\mathbb{B}^n) \to q(\mathbb{B}^n)$$
$$(M\chi)(x) \mathrel{\widehat{=}} 2\left(\tfrac{1}{2^n}\textstyle\sum_{y:\mathbb{B}^n}\chi(y)\right) - \chi(x).$$

Together with transform T_f, they form Grover's algorithm core iteration.

Table 2. Standard vs. probabilistic protocols

	G standard protocol	G probabilistic protocol
$p^0_G + p^1_G$	≤ 1 (Mitchison and Jozsa [5])	≤ 1 (Theorem 3; protocol K attains exactly 1)
Number N of insertions of the computer	$N \to \infty$ when $p^0_G + p^1_G \to 1$ (Mitchison and Jozsa [5])	N can be as low as 1 (our protocol K)

Our new protocol works like protocol K, except that before the last (diagonal) finalisation we first unitarily transform the switch and the output register in such a way "to drive" the switch to the off state. In qGCL it is coded as follows, where for simplicity we did not include the qubit ψ for distingushing between the off and on subspaces:

```
$O \mathrel{\widehat=}$ var $\chi{:}q(\mathbb{B}^2),\ o, s{:}\mathbb{B}^2 \bullet$
    In$(\chi)$⨾
    $\chi := T_{\delta_{11}}(\chi)$⨾
    $QC(\chi)$⨾
    Fin$(\mathcal{W}, o, \chi)$⨾
    $(\chi := T_{\delta_{10}}(\chi)) \lhd o \rhd (\chi := T_{\delta_{11}}(\chi))$⨾
    $\chi := T_{\delta_{00}}(\chi)$⨾
    $\chi := M(\chi)$⨾
    Fin$(\Delta, s, \chi)$
    rav
```

where $\mathcal{W}$ is the observable $\{\mathbb{C}v_0, \mathbb{C}v_1, (\mathbb{C}v_0 \oplus \mathbb{C}v_1)^\perp\}$. We argue that outcome $m{:}\mathbb{B}$ is an approximate counterfactual outcome of type m, and that $p_O^0 = p_O^1 = \frac{1}{2}$. We give here an informal proof of our claim.

We know from the functioning of K that, assuming QC_r has been executed, the state after the measurement of $\mathcal{W}$ is:

$$\chi = \frac{1}{2}(\delta_{00} + \delta_{01} + \delta_{1r} - \delta_{1\neg r}).$$

The following conditional thus transforms the state to:

$$\chi = \frac{1}{2}(\delta_{00} + \delta_{01} + \delta_{10} + \delta_{11}).$$

The next instruction flips the sign of the amplitude for basis state δ_{00}:

$$\chi = \frac{1}{2}(-\delta_{00} + \delta_{01} + \delta_{10} + \delta_{11}).$$

The execution of M on χ gives $\chi = \delta_{00}$. This means that we always end up in the initial "off" state δ_{00}. Furthermore, we recorded the outcome of the measurement of $\mathcal{W}$, which identifies with certainty the result of the decision problem. We note that although O always ends in δ_{00}, there are computations which return the outcome m, but do not belong to the off subspace. Those computations have been annihilated by the functioning of the algorithm (by embedding state χ via transformation E one can easily calculate them). Therefore, we cannot claim that the result is always for free, and that motivates condition (1) of definition 1 requiring that only the computation in the off subspace must have non-zero probability.

5 Conclusions

Counterfactual computation allows a seemingly paradoxical effect: to infer the result of a computation without running it. This remarkable fact can be achieved by means of peculiar properties of quantum mechanics. In this paper we showed how it is possible to formalise counterfactual computation in a framework provided by the quantum programming language qGCL, thereby showing that counterfactual computation is just an example of quantum computation. The main benefit of this approach is the possibility of exploiting the well-established body of programming laws which comes with qGCL. Next, we proposed a probabilistic extension of the original definition of counterfactual computation and we casted it into qGCL. We showed that probabilistic counterfactual protocols share some of the limitations of non-probabilistic protocols. In particular, for any probabilistic protocol G we must have $p_G^0 + p_G^1 \leq 1$. We presented a probabilistic protocol K for which $p_K^0 + p_K^1 = 1$, thus being optimal. Furthermore, our protocol K requires a single insertion of the quantum computer, while any non-probabilistic protocol would reach the upper bound 1 only with an infinite number of insertions. Therefore, the probabilistic relaxation of counterfactual computation helps only with respect to the complexity of the protocol.

Acknowledgements

This work was mainly carried out while visiting the Oxford University Computing Laboratory, with the support of *Consiglio Nazionale delle Ricerche* (Italy). The author is currently supported by a *Marie Curie Outgoing International Fellowship* within the 6^{th} Framework Programme of the European Commission. The author would like to thank Jeff Sanders for his support.

References

1. Avshalom C. Elitzur and Lev Vaidman. Quantum mechanical interaction-free measurements. *Foundations of Physics*, 32(7):987–997, 1993.
2. Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual symposium on the theory of Computing*, pages 212–219, 1996.
3. H. Jifeng, A. McIver, and K. Seidel. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28:171–192, 1997.
4. Richard Jozsa. Quantum effects in algorithms. *QCQC '98 Springer-Verlag LNCS*, 1509:103–112, 1999.
5. Graeme Mitchison and Richard Jozsa. Counterfactual computation. *Proceedings of the Royal Society of London* A, 457:1175–1193, 2001.
6. C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.
7. J. W. Sanders and P. Zuliani. Quantum programming. *Mathematics of Program Construction, Springer-Verlag LNCS*, 1837:80–99, 2000.
8. G.A. White, F.R. Mitchell, O. Nairz, and P. Kwiat. Interaction-free imaging. *Physical Review A*, 58:605–613, 1998.
9. Paolo Zuliani. Compiling quantum programs. To appear in *Acta Informatica*, 2005.

A pGCL Semantics

In this section we briefly review expectation-transformer semantics for pGCL [6], qGCL's parent language. Furthermore, we list some associated programming laws used in this work.

Definition 4. *The* state *x of a program P is the array of global variables used during the computation. That is*

$$x \mathrel{\widehat{=}} (v_1, \ldots, v_n) : T_1 \times T_2 \times \ldots \times T_n.$$

The Cartesian product $T_1 \times T_2 \times \ldots \times T_n$ of all the data types used is called the state space *of program P.*

An *expectation* is a $[0,1]$-valued function on a state space X and may be thought of as a "probabilistic predicate". The set $\mathcal{Q}$ of all expectations is defined:

$$\mathcal{Q} \mathrel{\widehat{=}} X \rightarrow [0,1].$$

Expectations can be ordered using the standard pointwise functional ordering for which we shall use the symbol $\Rrightarrow$, and $p \Rrightarrow q$ means "p everywhere no more than q". Standard predicates are easily embedded in $\mathcal{Q}$ by identifying *true* with expectation $\mathbf{1}$ and *false* with $\mathbf{0}$. For a standard predicate p we shall write $[p]$ for its embedding.

The pair $(\mathcal{Q}, \Rrightarrow)$ forms a complete lattice, with greatest element the constant expectation $\mathbf{1}$ and least element the constant expectation $\mathbf{0}$. For $i, j{:}\mathcal{Q}$ we shall write $i \equiv j$ iff $i \Rrightarrow j$ and $j \Rrightarrow i$ (or $i \Lleftarrow j$). The set $\mathcal{J}$ of all expectation transformers is defined:

$$\mathcal{J} \mathrel{\widehat{=}} \mathcal{Q} \rightarrow \mathcal{Q}.$$

Not every expectation transformer corresponds to a computation: only the *sub-linear* ones do [6].

The following table gives the expectation-transformer semantics for some pGCL commands (we shall retain the *wp* prefix of predicate-transformer calculus for convenience):

$$\boxed{\begin{aligned}
wp.\mathbf{abort}.q &\mathrel{\widehat{=}} \mathbf{0} \\
wp.\mathbf{skip}.q &\mathrel{\widehat{=}} q \\
wp.(x := E).q &\mathrel{\widehat{=}} q[x \backslash E] \\
wp.(R \mathbin{;} S).q &\mathrel{\widehat{=}} wp.R.(wp.S.q) \\
wp.(R \lhd cond \rhd S).q &\mathrel{\widehat{=}} [cond] * (wp.R.q) + [\neg cond] * (wp.S.q) \\
wp.(R \mathbin{\Box} S).q &\mathrel{\widehat{=}} (wp.R.q) \sqcap (wp.S.q) \\
wp.(R \mathbin{{}_p\oplus} S).q &\mathrel{\widehat{=}} p * (wp.R.q) + (1-p) * (wp.S.q)
\end{aligned}}$$

where $q{:}\mathcal{Q}$, $x{:}X$, $p{:}[0,1]$ and *cond* is an arbitrary predicate over state space; $q[x\backslash E]$ denotes the expectation obtained after replacing all free occurrences of x in q by expression E; $\sqcap$ denotes the greatest lower bound. Recursion is treated in general using the existence of fixed points in $\mathcal{J}$.

We now list a few algebraic programming laws which we used in the paper; the semantic models adopted and proofs can be found in [3,6]. In the following laws we use the term e to indicate an expression whose type is determined by the context.

Law (Id "*skip *identity"). $(P \mathbin{;} \mathbf{skip}) = (\mathbf{skip} \mathbin{;} P) = P$

Law (P-1). $P \ {}_1{\oplus}\ Q = P$

Law (P-2). $P \ {}_r{\oplus}\ Q = Q \ {}_{1-r}{\oplus}\ P$

Law (S-2). $(P \ {}_r{\oplus}\ Q) \mathbin{;} R = (P \mathbin{;} R) \ {}_r{\oplus}\ (Q \mathbin{;} R)$

Law (S-3). $(P \,\Box\, Q) \mathbin{;} R = (P \mathbin{;} R) \,\Box\, (Q \mathbin{;} R)$

Law (A-1). $(x := e) \mathbin{;} (P \ {}_r{\oplus}\ Q) = (x := e \mathbin{;} P) \ {}_{r[x\backslash e]}{\oplus}\ (x := e \mathbin{;} Q)$

Law (A-2). $(x := e \mathbin{;} x := f) = (x := f[e\backslash x])$

Law (A-3). $(x := e \mathbin{;} P \,\Box\, Q) = (x := e \mathbin{;} P) \,\Box\, (x := e \mathbin{;} Q)$

Since standard conditional is a particular case of probabilistic choice, laws S-2 and A-1 hold for that, too.

Author Index