Darko Stefanovic
Andrew Turberfield (Eds.)

LNCS 7433

DNA Computing and Molecular Programming

18th International Conference, DNA 18
Aarhus, Denmark, August 2012
Proceedings

Springer

Darko Stefanovic   Andrew Turberfield (Eds.)

DNA Computing and Molecular Programming

18th International Conference, DNA 18
Aarhus, Denmark, August 14-17, 2012
Proceedings

Springer

Volume Editors

Darko Stefanovic
University of New Mexico
Department of Computer Science
MSC01 1130, 1 University of New Mexico
87131 Albuquerque, NM, USA
E-mail: darko@cs.unm.edu

Andrew Turberfield
University of Oxford
Department of Physics, Clarendon Laboratory
Parks Road, Oxford OX 1 3PU, UK
E-mail: a.turberfield@physics.ox.ac.uk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-32207-5 e-ISBN 978-3-642-32208-2
DOI 10.1007/978-3-642-32208-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012942854

CR Subject Classification (1998): F.1, F.2.2, J.3, E.1, I.2, G.2, F.4

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

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

Printed on acid-free paper

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

Preface

This volume contains the papers presented at DNA 18: the 18th International Conference on DNA Computing and Molecular Programming, held August 14–17, 2012 at Aarhus University, Aarhus, Denmark.

Research in DNA computing and molecular programming draws together mathematics, computer science, physics, chemistry, biology, and nanotechnology to address the analysis, design, and synthesis of information-based molecular systems. This annual meeting is the premier forum where scientists of diverse backgrounds come together with the common purpose of advancing the engineering and science of biology and chemistry from the point of view of computer science, physics, and mathematics. Continuing this tradition, under the auspices of the International Society for Nanoscale Science, Computation, and Engineering (ISNSCE), DNA18 focused on the most recent experimental and theoretical results that promise the greatest impact.

The DNA 18 Program Committee received 37 paper submissions, of which 11 were selected for oral presentation and inclusion in the proceedings, and another 11 for oral presentation. Others were selected for poster presentations. Additional poster presentations came from a poster-only submission track.

The conference program included two tutorials—Milan Stojanovic, Columbia University: "Aptamers in Sensing and Molecular Computing"; and Damien Woods, California Institute of Technology: "A Crash Course in the Theory of Computing."

The conference program also included a plenary Turing Lecture by Grzegorz Rozenberg, University of Leiden: "Processes Inspired by Interactions of Chemical Reactions in Living Cells", and invited talks by Drew Berry, Walter and Eliza Hall Institute of Medical Research: "Visualizations of the Molecular Machines That Create Flesh and Blood"; Jeremy Gunawardena, Harvard Medical School: "Protein Computing"; Radhika Nagpal, Harvard University: "The TERMES Project: An Expedition in Large-Scale Self-assembly"; and Peng Yin, Harvard University: "Modular Self-Assembly of Molecular Shapes."

On the day before the conference, dnatec2012, a one-day workshop on structural DNA nanotechnology, was held, with contributions from Ned Seeman, New York University: "Using DNA's Inherent Chemical Information to Control Structure"; Andy Ellington, University of Texas: "DNA Nanotechnology: Too Small and Too Costly"; William Shih, Harvard Medical School: "Self-Assembled DNA-Nanostructure Tools for Molecular Biophysics"; Hao Yan, Arizona State University: "Designer DNA Architectures for Programmable Self-Assembly"; Andrew Turberfield, Oxford University: "Molecular Machinery from DNA"; Itamar Willner, Hebrew University of Jerusalem: "Nanobiotechnology with DNA"; Chengde Mao, Purdue University: "Self-Assembled DNA Nanocages"; Masayuki Endo, Kyoto University: "Direct Observation of Single Enzymatic and

Chemical Reactions in the Designed DNA Nanostructures"; Friedrich Simmel, Technische Universität München: "DNA Devices and Circuits as Components for Cell-Like Microcompartments"; Thom H. LaBean, North Carolina State University: "Building Agency into Molecular Materials"; and Luc Jaeger, University of California Santa Barbara: "Is RNA Self-assembly the Same as DNA Self-assembly?".

The editors would like to thank the members of the Program Committee and the external reviewers for their hard work in reviewing the papers and providing comments to the authors. They also thank the members of the Organizing Committee and the Steering Committee, and particularly the Committee Chairs, Kurt Vesterager Gothelf and Natasha Jonoska, for their support and advice. Generous financial support by the conference sponsors—the Danish National Research Foundation, Aarhus University, and DNA Technology A/S (Risskov, Denmark)—is gratefully acknowledged.

June 2012

Darko Stefanovic
Andrew Turberfield

Organization

DNA 18 was organized by the Danish National Foundation Center for DNA Nanotechnology at Aarhus University in cooperation with the International Society for Nanoscale Science, Computation, and Engineering (ISNSCE).

Steering Committee

Natasha Jonoska (Chair)	University of South Florida, USA
Luca Cardelli	Microsoft Research, UK
Anne Condon	University of British Columbia, Canada
Masami Hagiya	University of Tokyo, Japan
Lila Kari	University of Western Ontario, Canada
Satoshi Kobayashi	University of Electro-Communication, Chofu, Japan
Chengde Mao	Purdue University, USA
Satoshi Murata	Tohoku University, Japan
John Reif	Duke University, USA
Grzegorz Rozenberg	University of Leiden, The Netherlands
Nadrian Seeman	New York University, USA
Friedrich Simmel	Technische Universität München, Germany
Andrew Turberfield	Oxford University, UK
Hao Yan	Arizona State University, USA
Erik Winfree	California Institute of Technology, USA

Organizing Committee

Kurt Vesterager Gothelf (Chair)	Aarhus University, Denmark
Ebbe Sloth Andersen	Aarhus University, Denmark
Stinne Høst	Aarhus University, Denmark
Sarah Helmig	Aarhus University, Denmark
Thom LaBean	North Carolina State University, USA

Program Committee

Darko Stefanovic (Co-chair)	University of New Mexico, USA
Andrew Turberfield (Co-chair)	Oxford University, UK
Luca Cardelli	Microsoft Research, UK
Eugen Czeizler	Aalto University, Finland
Erik Demaine	Massachusetts Institute of Technology, USA
David Doty	California Institute of Technology, USA

Andrew Ellington	University of Texas at Austin, USA
Kurt Gothelf	Aarhus University, Denmark
Natasha Jonoska	University of South Florida, USA
Lila Kari	University of Western Ontario, Canada
Yamuna Krishnan	National Centre for Biological Sciences, India
Chengde Mao	Purdue University, USA
Satoshi Murata	Tokyo Institute of Technology, Japan
Jennifer Padilla	New York University, USA
John Reif	Duke University, USA
Rebecca Schulman	Johns Hopkins University, USA
Georg Seelig	University of Washington, USA
Friedrich Simmel	Technische Universität München, Germany
David Soloveichik	University of California, San Francisco, USA
Erik Winfree	California Institute of Technology, USA
Hao Yan	Arizona State University, USA
Peng Yin	Harvard University, USA

Referees

Andersen, Ebbe	Lakin, Matthew	Simjour, Amir
Chandran, Harish	Marblestone, Adam	Song, Tianqi
Chen, Xi	Minnich, Amanda	Sun, Wei
Gao, Yuan	Patitz, Matthew	Wei, Bryan
Gopalkrishnan, Nikhil	Ran, Tom	Williams, Lance
Karpenko, Daria	Schaus, Thomas	Zhang, David
Kopecki, Steffen	Seki, Shinnosuke	Zizza, Rosalba

Sponsoring Institutions

The Danish National Research Foundation
Aarhus University
DNA Technology A/S

Table of Contents

Turing Universality of Step-Wise and Stage Assembly at Temperature 1

Bahar Behsaz[1], Ján Maňuch[1,2], and Ladislav Stacho[1]

[1] Department of Mathematics, Simon Fraser University, Burnaby, Canada
[2] Department of Computer Science, University of British Columbia, Vancouver, Canada
{bbehsaz,lstacho}@sfu.ca, jmanuch@cs.ubc.ca

Abstract. In this paper, we investigate the computational power of two variants of Winfree's abstract Tile Assembly Model [14] at temperature 1: the Stage Tile Assembly Model and the Step-wise Tile Assembly Model. In the Stage Tile Assembly Model, the intermediate assemblies are assembled in several "bins" and they can be mixed in prescribed order and attach together to form more complex structures. The Step-wise Tile Assembly Model is a simplified model of stage assembly in which only one bin is used and assembly happens by attaching tiles one by one to the growing structure.

An interesting and still open question is whether the abstract Tile Assembly Model at temperature 1 is Turing Universal, i.e., it can simulate a Turing machine. It is known that various slight modifications of the model are indeed Turing Universal. Namely, deterministic self-assembly in 3D and probabilistic self-assembly in 2D at temperature 1 [3] and self-assembly model at temperature 1 with a single negative glue [10] are known to be able to simulate a Turing machine. In this paper we show that the Step-wise Tile Assembly Model and the Stage Tile Assembly Model are also Turing Universal at temperature 1.

1 Introduction

Informally, self-assembly is a bottom-up process in which a small number of types of components automatically assemble to form a more complex structure.

In 1998, Winfree [14] introduced the (abstract) Tile Assembly Model (TAM) – which utilizes the idea of Wang tiling [13] – as a simplified mathematical model of the DNA self-assembly pioneered by Seeman [12]. Nature provides many examples: Atoms react to form molecules. Molecules react to form crystals and supermolecules. Similarly, self-assembly of cells plays a part in the development of organisms. Simple self-assembly schemes are already widely used in chemical syntheses. It has been suggested that more complicated schemes will ultimately be useful for circuit fabrication, nano-robotics, DNA computation, and amorphous computing [2,15,11,7,6,1]. Several promising experiments as well has been suggested and practiced. In accordance with its practical importance, self-assembly has received increased theoretical attention over the last few years [9,14,2,15,8].

One of the interesting theoretical questions about this process is how "powerful" the model of Winfree is. In the past years Self Assembly Model has been studied by many researchers from computational aspect of view. Many of these results are focused on the systems with temperature 2 or higher [3,5]. In such models strength of at least two is needed for a tile to permanently bind to an assembly. It is known that at temperatures equal or greater than 2, Self Assembly Systems are quite powerful, in fact it is shown that they are Turing Universal [3]. But for temperature 1, where the system is allowed to place a tile at a position if any positive strength bond exist among the tile and the neighbors at that position, the computational "power" is unknown.

From practical point of view, temperature 1 systems are more well-mannered and easier to control for laboratory experiments. Thus recent attention has been redirected to some variants of temperature 1 Self Assembly Model. It is shown that by adding some constraints and features to the original model more "power" is achievable. Cook et al. [3] proved the Turing Universality for deterministic self-assembly in 3D and probabilistic self-assembly in 2D at temperature 1, and Patitz et al. [10] introduced a self-assembly model at temperature 1 with a single negative glue, and showed it is Turing Universal. All results use the idea of "geometric tiles".

In this paper we study the power of two variants of the Self Assembly Model at temperature 1: "the Stage Tile Assembly Model" and "the Step-wise Tile Assembly Model". These models are defined in Section 2. We reuse the idea of "geometric tiles", and prove that these models are Turing Universal as well. In Section 3, we introduced the Zig-Zag Assembly which can simulate Turing machines at temperature 2. To show our main results it is enough to simulate the Zig-Zag assembly at temperature 1 in these two models. Details of these simulations are presented in Sections 4 and 5, respectively.

2 Abstract Tile Assembly Model

We present a brief description of the tile assembly model based on Rothemund and Winfree, for a more detailed description we refer the reader to [14]. We will be working on a $\mathbb{Z}\times\mathbb{Z}$ grid of unit square locations. The set of directions $D = \{N, E, W, S\}$ is used to indicate relative positions in the grid. Formally, they are functions $\mathbb{Z}\times\mathbb{Z} \to \mathbb{Z}\times\mathbb{Z}$ such that:

$$N(i,j) = (i,j+1),\ E(i,j) = (i+1,j),\ S(i,j) = (i,j-1) \text{ and } W(i,j) = (i-1,j).$$

The inverse directions are defined in a natural way, for example, $N^{-1}(i,j) = S(i,j)$.

Let Σ be a set of *binding domains (glues)*. The set Σ contains a special binding domain *null* that represents no glue. A *tile type* t is a 4-tuple $(t_N, t_E, t_S, t_W) \in \Sigma^4$ indicating the associated binding domains on the north, east, south and west sides, respectively. Note that tile types are oriented, thus a rotated version of a tile type is considered to be a different tile type. We denote the set of tile types with T. There is a special tile type *empty* $= (null, null, null, null)$ in T which represents an empty space, i.e., no tile has been placed in that position. A configuration is a function C, mapping $\mathbb{Z} \times \mathbb{Z}$ to the set of tile types in T. Hence, $C(i, j)$ is the tile at the position (i, j) in the configuration C. Let a *structure* $S(C)$ of a configuration be the set of positions that are not mapped to *empty* in C. A configuration may contain finite or infinte number of tiles.

Under Tile Assembly Model a tile system is a 5-tuple $\langle \Sigma, T, \phi, g, \tau \rangle$, where T is the finite set of tile types with binding domains from Σ and contains the tile *empty*, ϕ is a set of configurations on T called set of seed configurations, g is a function which determines the strength of glues, and τ is a threshold parameter called temperature. In this paper we will omit parameters Σ and g when they are defined already.

A strength function $s_\Sigma : \Sigma \times \Sigma \rightarrow \mathbb{N}$ measures the strength of interaction between binding domains. For example, for simple strength function at temperature 1 $\tau = 1$, denoted by s_Σ, it must satisfy $s_\Sigma(\sigma, \sigma') = 1$, if $\sigma = \sigma' \neq null$, and $s_\Sigma(\sigma, \sigma') = 0$ otherwise. A tile t can be added to a configuration at position (i, j), if and only if the sum of the interaction strengths of t with its neighbors at position (i, j) reaches or exceeds τ (the temperature). If the assembly process reaches a point when no more attachments are possible, the produced assembly is denoted terminal and is considered the output assembly of the system. The smallest number of tile types and glues required to assemble a given shape are called the tile and glues complexity of the shape, respectively.

2.1 Staged Tile Assembly Model

Erik D. Demaine et al. [4], presented the Staged Tile Assembly Model, a generalization of the tile assembly model that captures the temporal aspect of the laboratory experiment, and enables more flexibility in the design and fabrication of complex shapes using a small tile and glue complexity. The main feature of staged assembly is the ability of gradual addition of specific tiles in a sequence of stages. In addition, any tiles that have not yet attached as part of a growing structure can be washed away and removed (in practice, using a weight-based filter, for example) at the end of the stage. More generally, we can have any number of bins, each containing tiles and/or assemblies that self-assemble as in the standard tile assembly model. During a stage, any collection of operations of two types are allowed: (1) add (arbitrarily many copies of) new tiles to an existing bin; and (2) pour one bin into another bin, mixing the contents of the former bin into the latter bin. In both cases, any tiles that do not assemble into larger structures are removed at the end of the stage. These operations let us build intermediate terminal assemblies in isolation and then combine different terminal assemblies to more complex structures. Two new complexity measures in addition to tile and glue complexity arise: the number of stages, or stage complexity, measures the time required by the operator of the experiment, and the number of bins, or bin complexity, measures the space required for the experiment. (When both of these complexities are 1, the model is equivalent to the regular tile assembly model.)

2.2 Step-Wise Tile Assembly Model

Reif proposed a related, but simpler, model called the Step-wise Assembly Model [11]. This model is the special case of Staged Assembly Model, in which the bin complexity is 1. Step-wise Assembly Model uses several sets of tiles (one for each step). Let T_i be the tile set for step i. $\{T_i\}_{i=1}^{k}$ is a sequence of finite sets of tiles. At first step we immerse a seed tile into one of the sets, filter out the assembled shape from this set and place the assembled shape into the next set of tiles where it now acts as a seed tile and assembly continues. Step-wise Assembly Model enables production of more efficient assemblies in terms of tile types at the price of the work of an operator. Here one new complexity measure can be defined as well, the number of steps, or step complexity, which measures the time required for the experiment.

3 Zig-Zag Tile Assembly Model

Informally, a Zig-Zag assembly is an assembly in which the assembly can grow only left to right (or right to left) up to the point at which a first tile is placed into the next row north, and the direction of growth is reversed in the next top row.

Temperature 2 Zig-Zag assembly system: A tile system $\Gamma = \langle T, s, 2\rangle$ is called temperature 2 Zig-Zag system if the tiles in T assures the assembly grows under these two conditions:

1. The assembly sequence, which specifies the order in which tile types are attached to the system is unique, i.e., the system is deterministic.
2. If the $(i-1)$th tile added to the assembly sequence is placed in an even row (counting from the row 0, containing the seed tile) in position (x, y) of the grid, then the ith tile is either placed in position $E(x, y)$ or $N(x, y)$. And if the $(i-1)$th tile added is placed in an odd row, then the next tile to be added (i.e. ith tile in the sequence) is placed either in the position $W(x, y)$ or $N(x, y)$.

Because $\tau = 2$, the key property of Zig-Zag system is interaction of strength at least two between the sides of a tile and size neighbors is required to be able to add the ith tile to the sequence. Right after adding a tile t to the assembly the glues on the sides of t that already have neighbors, are called "*inputs*" and the glues on the sides of t that are exposed are called "*outputs*" of tile t. Cook et al. proved that the deterministic Zig-Zag Tile Assembly Model at temperature 2 is Turing universal [10], and using this they proved the deterministic assembly in 3D and the probabilistic assembly in 2D at temperature 1, are Turing Universal by simulating Zig-Zag assembly System with the idea of "geometric tiles". We will use the same technique to prove our main results.

4 Turing Universality of Step Tile Assembly System

This section and Section 5 contain the main results of this paper. First we prove that every Zig-Zag tile assembly system at temperature 2 can be simulated by a step-wise assembly tile system at temperature 1.

Theorem 1. *For any temperature 2 Zig-Zag tile assembly system $\Gamma = \langle T, s, 2\rangle$ there exists a Step-wise tile assembly system Γ' at temperature 1 that simulates Γ at vertical scale 5, horizontal scale $O(\log(|T|))$, step complexity and tile complexity $O(|T|\log(|T|))$.*

Proof. We follow the idea of the proof in [3]. We represent each tile in Γ with a set of tiles in Γ' called "*macro tiles*" and "fake" the cooperative attachment of tiles in temperature 2.

Let G be the set of all strength 1 glues on north and south side of tiles that appear in the system Γ. Label each type of these glues with numbers 0 to $|G|-1$ in an arbitrary order. For any glue $g \in G$ let $b(g)$ be the binary representation of the label of g. The main idea is to geometrically represent number $b(g)$ on the side of macro tile that is representing the north or south glue of each $t \in T$. We symbolize each bit of $b(g)$, 0 or 1, with bumps. For east and west side glues and for strength 2 glues we do not use geometrical representation. Those are represented with simple strength 1 glues. By alternating between the two tile sets, the system will simulate the input from the south with its bumps, and from left and right the inputs are read with a strength 1 glue. After reading the input, the outputs are unpacked (written) using the same geometrical representation of the bits of the binary coding of output glue on the north (if any) and a glue of strength 1 on east or west side (according to the side of the outputs of the tile in the Zig-Zag system that the system is assembling).

Each macro tile has two bumps of vertical length 2 on beginning and end of its north side as shown in Figure 1. These special bumps will guarantee that macro tiles in the assembly will be aligned. The actions of reading and unpacking glues for a macro tile m that corresponds to a tile $t \in \tau$ with south and west inputs and north and east outputs are discussed in more details in what follows. Examples of tile with other combinations of input and output are illustrated in Figure 1.

Reading the South and West Inputs: Each macro tile starts from the south. The first tile that starts the macro tile has the west input on its west side with one single glue. Then the assembly continues by reading along the north side of the macro tile beneath it. Figure 2 pictures the basic idea behind reading each bit. As it is shown in Figure 2 each bit has width 4 and we recognize whether the bit is 0 or 1 by the location of the bump. Also, each bit is being read in exactly two steps and with exactly two tile sets T_1 an T_2, represented in Figure 2. The final tile sets T_1 and T_2 is a union of all required tiles for different tiles respectively in step 1 and 2.

The tiles shown in Figures 3a and 3c are the key tile types in each tile set (tile set 1 and 2) responsible for reading bit i of the glue on the south side. The

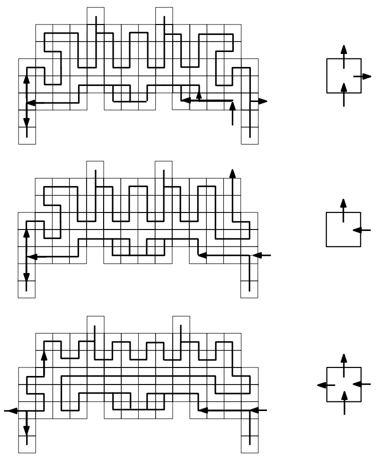


Fig. 1. Macro tiles simulating the tile types from Zig-Zag Tile Assembly System shown on the right side of each macro tile

final assembly of this part of the macro tile which reads the glue on the south of the macro tile for a tile $t \in T$ with glue 110 on the south side, is illustrated in Figure 4.

For reading each bit of the binary representation of the glue on the south side of each macro-tile, say m, there are $O(1)$ unique tiles in tile sets T_1 and $|T_2|$. These tiles assemble together and read the ith bit of the glue on the south side of m when the glue is being read in process of creating m.

After reading $\log(|G|)$ bits, the glue on the south is read completely. The west input is read via the glue on west side of the first step in the assembly sequence of the macro tile. Thus, at this point both inputs are read and the outputs can be determined deterministically (since Zig-Zag TAS is deterministic) and this mapping is done with the tile shown in Figure 3e.

Unpacking the North and East Outputs: When the outputs are determined the assembly crawls back on itself (in this case it crawls back to east). Then it starts "writing" the binary representation of the north glue with illustrating 0 an 1 geometrically. This process is in essence the same as what discussed above but naturally with reverse functionality. Instead of adding the next right digit in this case we delete the right most digit each time at each step. The key tile types necessary for unpacking a glue is illustrated in Figure 5. Note that it unpacks the east glue again with a single unique strength 1 glue on the tile at the very south east position of macro-tile. The complexity of tile types is the same as the reading tiles.

For each bit of the binary representation of the glue on the south or north side of each macro-tile, say m, there are $O(1)$ unique tiles in the tile sets T_1 and T_2. Each macro-tile needs $O(\log(|T|))$ unique tile types. The terminal assembly of Zig-Zag assembly Γ has $O(|T|)$ tiles and all of the $O(\log(|T|)$ tiles assembling

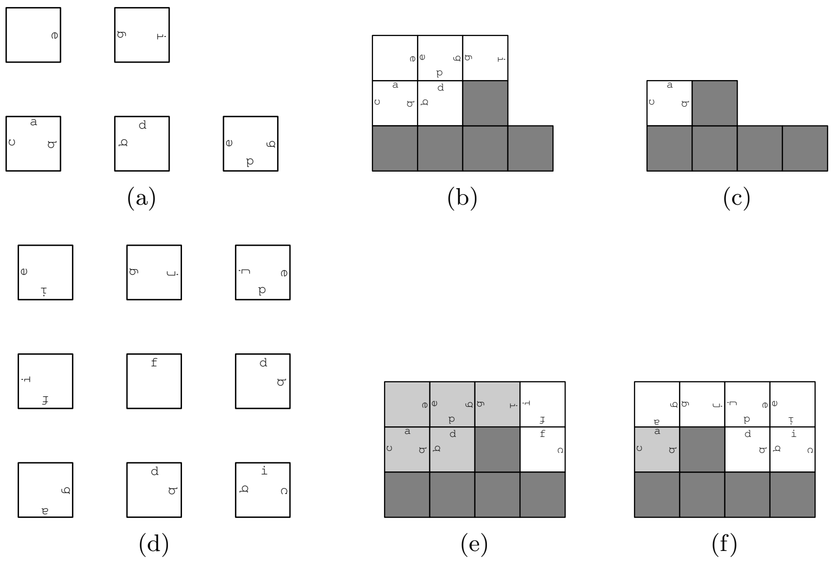


Fig. 2. (a) The first tile set T_1. **(b)** The final assembly for 0 after the first step (after exposing to T_1). The tiles colored light gray belong to the first tile set and the tiles colored with dark gray belong to the macro-tile beneath. The assembly is growing from left to right. **(c)** The final assembly for 1 after the first step (after exposing to T_1). **(d)** The first tile set T_1. **(e)** The final assembly for 0 after the first step (after exposing to T_1). **(f)** The final assembly for 1 after the first step (after exposing to T_1).

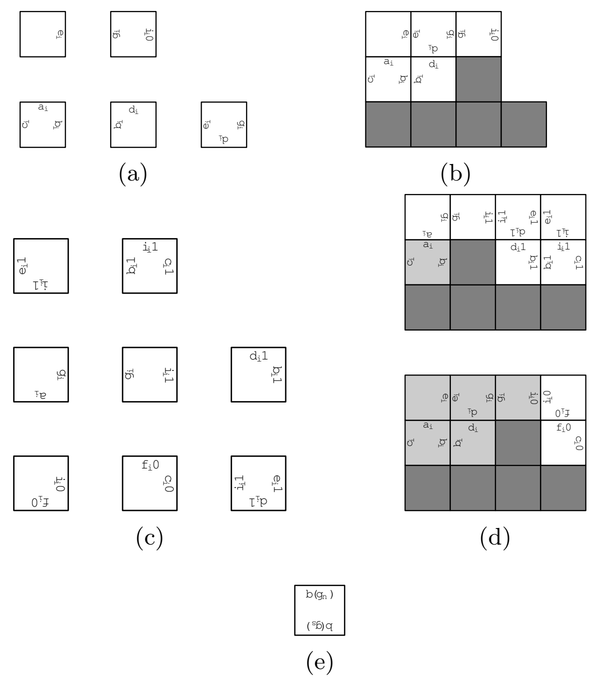


Fig. 3. (a) The subset of T_1 reading the ith bit from left to right. $a_i, \ldots, j_i$ are different representations of the first $i-1$ bits. **(b)** The final assembly reading bit 0 after exposing to the first tile set. The tiles that are colored light gray belong to the first tile set and the tiles colored with dark gray belong to the "reading part" of this macro-tile. **(c)** The subset of T_2 reading the ith bit from left to right. **(d)** The final assembly tile set reading bits 0 and 1 after exposing to second tile set. **(e)** The tile mapping the input of a macro tile that is read to the corresponding output. At this point, the assembly starts crawling back on the assembly and start unpacking.

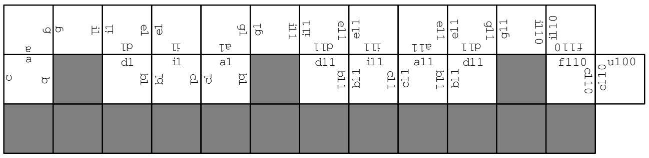


Fig. 4. An example of reading the glue 110 on south side of a macro tile bit by bit from left to right

macro-tile m has to store the original tile that m belongs to in Γ. Thus, we use $O(|T|\log(|T|))$ unique tile types. Also, each macro-tile is constructed in $O(\log(|G|))$ steps. Hence, the tile and step complexity of the final assembly is $O(|T|(\log|T|))$. The assembly system described here clearly simulates Γ.

For every Turing machine M and $w \in \Sigma^*$ there exists a Zig-Zag Tile Assembly Model Γ that simulates M on input w [3]. By Theorem 1 there is a step-Tile Assembly System that simulates Γ. Therefore, we have the following result:

Theorem 2. *The Step-wise Tile Assembly Model is Turing Universal.*

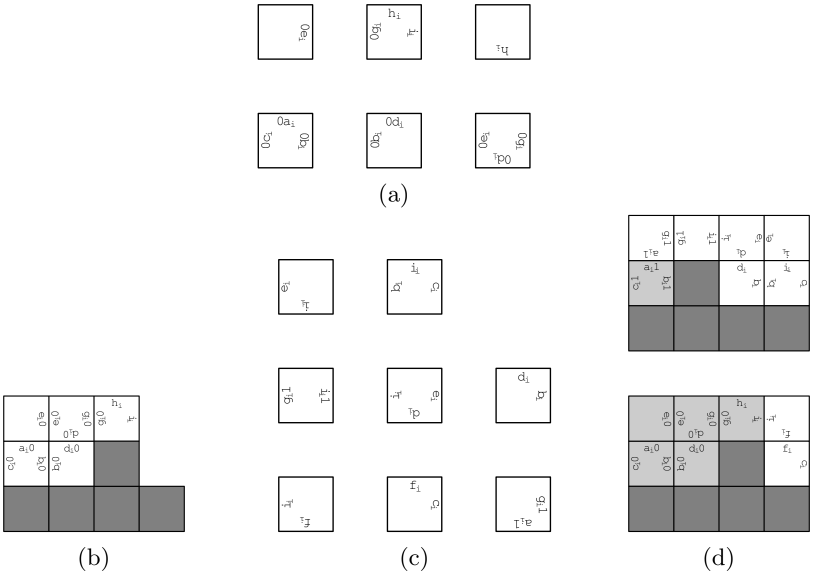


Fig. 5. **(a)** The subset of T_1 unpacking the ith bit from left to right. $a_i, \ldots, j_i$ are different representations of the first $i-1$ bits. **(b)** The final assembly unpacking bit 0 after exposing to the first tile set. The tiles that are colored light gray belong to the first tile set and the tiles colored with dark gray belong to the macro-tile beneath. **(c)** The subset of T_2 unpacking the ith bit from left to right. **(d)** The final assembly tile set unpacking bits 0 and 1 after exposing to second tile set.

5 Turing Universality of Stage-Assembly System

By performing the exact procedure which is described in Section 4, the Turing Universality of Stage Assembly Model can be proved. But for Stage-wise Assembly Model we can utilize the construction with the "bins" and reduce the number of stages. For this, we use a similar technique as in the construction in Theorem 1. We take an instance of Zig-Zag assembly system, say Γ and simulate it with an instance of stage-assembly system which is constructed with macro tiles as described in the previous section. But here with the use of different bins we can decrease the stage complexity to $O(\log(|T|))$.

Theorem 3. *For any temperature 2 Zig-Zag tile assembly system $\Gamma = \langle T, s, 2\rangle$ there exists a Stage Tile Assembly system Γ' at temperature 1 that simulates Γ at vertical scale 5, horizontal scale $O(\log(|T|))$, stage complexity $O(\log(|T|))$ and tile complexity $O(|T|\log(|T|))$.*

Proof. The construction of Γ' is almost the same as proof of Theorem 1. Again let G be the set of north/south glues. For each $t \in T$ we create a unique macro-tile. The tile sets of Γ' are T_1' and T_2', which are the same as T_1 and T_2 in Γ which is described in the proof of Theorem 1, respectively. The difference is that the tiles in the south side of each macro-tile do not read a binary number. Instead they unpack (write) the corresponding binary number, which is the binary number of the glue of the south side of t. The tile types which assist unpacking this glue are the same as the tiles types that are used to read this glue for the macro tile corresponding to t in the proof of Theorem 1. For each tile $t \in T$ we use a unique bin. The seed tile here is the tile which maps the input to the output of the macro tile which is shown in Figure 3c. So the assembly sequence for the assembly that unpacks the south glue is the reverse of the assembly sequence which reads the south glue in Section 4. After each macro-tile assembled in each bin we will pour the desired assembled macro tiles in one bin with choosing the macro-tile in Γ' corresponding to the seed tile in Γ be the seed configuration in Γ'.

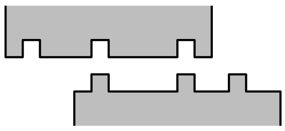


Fig. 6. The possible slips for two macro-tiles one with south side depicting (110) the other with north side (101). The two segments of (10) can be matched together without the vertical teeth of length two on the south of macro tiles.

In each bin it takes $O(\log(|G|))$ stages before the assembly of the macro tile is final, this yields to stage complexity $O(\log |T|)$. Since the tile types are the same in Section 4, the tile complexity is $O(|T| \log(|T|))$.

Now, since Γ is a deterministic system the macro-tiles automatically assemble by matching inputs and outputs uniquely in one step. The vertical bump of length 2 on south side of each macro-tile guarantees that the beginning and the end of each macro-tile matches exactly with the beginning and the end of other macro-tiles in north and south side of the macro-tile. Without these bumps horizontal slips of macro-tiles along each other is possible as shown in Figure 6.

6 Conclusion

In this paper, we proved Turing Universality at temperature 1 for two tile assembly models: Step-wise Tile Assembly Model and Stage Tile Assembly Model. The question whether the Tile Assembly Model at temperature 1 is Turing Universal remains open.

References

1. Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight Jr., T.F., Nagpal, R., Rauch, E., Sussman, G.J., Weiss, R.: Amorphous computing. Commun. ACM 43(5), 74–82 (2000)
2. Adleman, L., Cheng, Q., Goel, A., Huang, M.-D., Kempe, D., de Espanés, P.M., Rothemund, P.W.K.: Combinatorial optimization problems in self-assembly. In: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, pp. 23–32 (2002)
3. Cook, M., Fu, Y., Schweller, R.: Temperature 1 self-assembly: Deterministic assembly in 3D and probabilistic assembly in 2D. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (2011)
4. Demaine, E.D., Demaine, M.L., Fekete, S.P., Ishaque, M., Rafalin, E., Schweller, R.T., Souvaine, D.L.: Staged Self-assembly: Nanomanufacture of Arbitrary Shapes with $O(1)$ Glues. In: Garzon, M.H., Yan, H. (eds.) DNA 13. LNCS, vol. 4848, pp. 1–14. Springer, Heidelberg (2008)
5. Doty, D., Patitz, M.J., Summers, S.M.: Limitations of self-assembly at temperature 1. Theor. Comput. Sci. 412(1-2) (January 2011)
6. Gómez-López, M., Preece, J.A., Fraser Stoddart, J.: The art and science of self-assembling molecular machines. Nanotechnology 7(3), 183 (1996)
7. Lagoudakis, M.G., Labean, T.H.: 2D DNA self-assembly for satisfiability. In: Proceedings of the 5th DIMACS Workshop on DNA Based Computers (1999)
8. Mao, C., Labean, T.H., Reif, J.H., Seeman, N.C.: Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature 407, 493–496 (2000)
9. Maňuch, J., Stacho, L., Stoll, C.: Step-wise tile assembly with a constant number of tile types. In: Natural Computing (to appear)
10. Patitz, M.J., Schweller, R.T., Summers, S.M.: Exact Shapes and Turing Universality at Temperature 1 with a Single Negative Glue. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 175–189. Springer, Heidelberg (2011)
11. Reif, J.H.: Local parallel biomolecular computing. In: DNA Based Computers III. DIMACS, vol. 48 (1999)
12. Seeman, N.C.: Nucleic-acid junctions and lattices. Journal of Theoretical Biology 1, 237–247 (1982)
13. Wang, H.: Proving theorems by pattern recognition II. The Bell System Technical Journal XL 1, 1–41 (1961)
14. Winfree, E.: Algorithmic Self-assembly of DNA. PhD thesis, California Institute of Technology (1998)
15. Winfree, E., Yang, X., Seeman, N.C.: Universal computation via self-assembly of dna: Some theory and experiments. In: DNA Based Computers II. DIMACS, vol. 44, pp. 191–213 (1996)

A Type System for DNAQL

Robert Brijder, Joris J.M. Gillis*, and Jan Van den Bussche

Hasselt University and transnational University of Limburg,
Belgium

Abstract. Recently we have introduced a formal graph-based data model for DNA complexes geared towards database applications. The model is accompanied by the programming language DNAQL for querying databases in DNA. Due to natural restrictions on the implementability and termination of operations on DNA, programs in DNAQL are not always well defined on all possible inputs. Indeed, a problem left open by our previous work has been to devise a type system for DNAQL, with a soundness property to the effect that well-typed programs are well defined on all inputs adhering to given input types. The contribution of the present paper is to propose such a type system and to establish soundness. Moreover, we show that the type system is flexible enough so that any database manipulation expressible in the relational algebra is also expressible in DNAQL in a well-typed manner.

1 Introduction

Since Adleman's experiment [2], many different models for DNA computing have been invented and investigated, as can be learned from the books [3,16] and more recent developments [11,24,18]. At the same time, DNA computing has also high potential for database applications [4,7,25,20]. In this spirit, in recent work [13,5], we have defined the programming language DNAQL: a programming language specifically designed for the querying of databases in DNA. The goal of the present paper is to provide DNAQL with a sound type system.

DNAQL is a query language rather than a general-purpose programming language. It includes basic operators on DNA complexes in solution. Apart from the application of these operators, programs are formed using a let-construct and an if-then-else construct based on the detection of DNA in a test tube. Last but not least, the language includes a for-loop construct for iterating over the bits of a data entry, encoded as a vector of DNA codewords. Indeed, the number of operations performed during the execution of a DNAQL program, on any input, is bounded by a polynomial that depends solely on the dimension of the data, i.e., the number of bits needed to represent a single data entry. This makes that the execution time of programs scales well with the size of the input database.

A difficulty with DNAQL, and with DNA computing in general, however, is that various manipulations of DNA must make certain assumptions on their

* Ph.D. fellow of the Research Foundation - Flanders (FWO).

input so as to be effectively implementable and produce a well-defined output. Even when these assumptions are well understood for each operation in isolation, the problem is exacerbated in an applicative programming language like DNAQL, where the output of one operation serves as input for another. Indeed the problem of deciding whether a given program will have well-defined behavior on all possible intended inputs is typically undecidable. While this undecidability is well known for Turing-complete programming languages, it remains so for database languages that are typically not Turing-complete [6].

The standard solution to ensure well-definedness of programs is to use a type system and check programs syntactically so as to allow only well-typed programs. Well-devised type systems have a soundness property to the effect that, once a program has been checked to be well-typed for a given input type, the behavior of the program is then guaranteed to be well defined on all inputs of the given type [17,14]. In the present paper, we propose a type system for DNAQL and establish a soundness theorem. Moreover, we show that the type system is flexible enough so that arbitrary relational databases can be represented as typed DNA complexes, and so that arbitrary relational algebra expressions on these data can be expressed by well-typed DNAQL programs. The relational algebra is the applicative language at the core of standard database query languages such as SQL [9,12,1].

We would like to make clear in what sense the present paper enhances previous work. That the relational algebra can be simulated in DNAQL has already been shown [13], but only insofar as the dynamic behavior at run-time is concerned. Here we show that the simulation can be syntactically guaranteed to be possible with well-typed DNAQL programs only. Also in recent work [5] we formulated a syntactic test on the well-definedness of hybridization, similar to weak satisfiability [15]. This syntactic test is but one component of the type system presented here, and here it is also extended to account for components of DNA complexes that are immobilized on separation surfaces such as magnetic beads.

Most importantly, a crucial feature of the type system presented here is a wildcard mechanism to account for the fact that the length (in bits), as well as the actual values, of data entries are unknown at compile time. This mechanism is integrated in a type-checking system that keeps track of mandatory components in DNA complexes, as well as their hybridization status. The result is a type system that allows a natural and flexible representation of structured data in DNA, in a way so that a significant class of data manipulations can be typed as programs in DNAQL.

2 Sticker Complexes

We recall [13,5] the data model of DNA sticker complexes, a graph-theoretically defined formalization of DNA complexes of a limited format geared towards data representation. Due to space limitations, we must be brief.

From the outset we assume a finite alphabet Σ. As customary in formal models of DNA computing [16], each letter represents a *string* over the DNA alphabet $\{A, C, G, T\}$, such that the resulting set of sequences forms a set of DNA codewords [8,22,23]. This should always be kept in mind. The alphabet Σ is matched with its negative version $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$, disjoint from Σ. Thus there is a bijection between Σ and $\bar{\Sigma}$, which is called *complementarity* and is denoted by overlining; we also set $\bar{\bar{a}} = a$ so complementarity is symmetric. Obviously, $\bar{a}$ stands for the Watson-Crick complement of the DNA sequence represented by a. The elements of Σ are called *positive symbols* and the elements of $\bar{\Sigma}$ are called *negative symbols*.

For the purpose of data formatting we further assume that $\Sigma = \Lambda \cup \Omega \cup \Theta$ is composed of three disjoint parts: the set Λ of *atomic value symbols*; the set Ω of *attribute names*; and the set $\Theta = \{\#_1, \#_2, \#_3, \#_4, \#_5, \#_6, \#_7, \#_8, \#_9\}$ of *tags*.

The overall structure of a DNA complex is abstracted in the notion of *pre-complex*. Formally, a pre-complex is a 6-tuple $(V, L, \lambda, \mu, \iota, \beta)$, where

1. V is a finite set of nodes;
2. $L \subseteq V \times V$ is a set of directed edges without self-loops;
3. $\lambda : V \to \Sigma \cup \overline{\Sigma}$ is a total function labeling the nodes with positive and negative alphabet symbols;
4. $\mu \subseteq [V]^2 = \{\{u, v\} \mid u, v \in V \text{ and } u \neq v\}$ is a partial matching on the nodes, i.e., each node occurs in at most one pair μ. Note that the pairs in μ are unordered.
5. $\iota \subseteq V$ is the set of *immobilized* nodes; and
6. $\beta \subseteq V$ is the set of *blocked* nodes.

Let C be a pre-complex as above. A *strand* of C is simply a connected component of the directed graph (V, L), so ignoring μ. The *length* of a strand is its number of nodes. A *sticker complex* (or *complex* for short) now is a pre-complex satisfying the following restrictions:

1. Each node has at most one incoming and at most one outgoing edge. Thus, each strand has the form of a chain or a cycle.
2. Strands are homogeneously labeled, in the sense that either all nodes are labeled with positive symbols, or all with negative symbols. Naturally, a strand with positive (negative) symbols is called a positive (negative) strand.
3. Every negative strand has length one or two; if it has length two, then it must have a single edge (i.e., it cannot be a 2-cycle). Negative strands are also referred to as “stickers”.
4. Matchings by μ only occur between complementarily labeled nodes: formally, if $\{x, y\} \in \mu$ then $\lambda(y) = \overline{\lambda(x)}$.
5. A node can be immobilized only if it is the sole node of a negative strand.
6. Each component can contain at most one immobilized node.
7. Nodes in β do not occur in μ.

We see that the edges of a sticker complex indicate the sequence order within strands, and the matching μ makes explicit where stickers have annealed to positive strands. The predicate β represents longer stretches of double strands and is used to restrict the places where hybridization can still occur [21]. Immobilized nodes represent probes attached to magnetic beads or surfaces that can be separated from the rest of the solution.

Components and Redundancy. Two strands s and s' are *bonded* if there is a node v in s and some node v' in s' with $\{v, v'\} \in \mu$. When two strands are connected (possibly indirectly) by this bonding relation, we say they belong to the same component. Thus a *component* of a pre-complex is a substructure formed by a maximal set of strands connected by the bonding relation. Note that a component of a pre-complex is in itself a pre-complex. We use $comp(C)$ to denote the set of components of pre-complex C. Conversely, we can view a set of sticker complex components as a single sticker complex, basically by taking the union.

The intention of our model is that a complex defines the structural content of a test tube, which, however, will hold copies in surplus quantity of each component. Thus, each component of a complex stands for multiple occurrences. We formalize this using the notions of subsumption, equivalence, and minimality.

A pre-complex C_1 *is subsumed* by pre-complex C_2 if for each component D_1 in C_1 there is an isomorphic component D_2 in C_2. Two pre-complexes are *equivalent* if they subsume each other. A component D in pre-complex C is *redundant* if there exists a component D' in C such that D and D' are isomorphic. Note that removing D from C yields an equivalent sticker complex. A pre-complex is *minimal* if there are no redundant components.

Saturated Complexes. We call a complex C *saturated* if there do not exist any two nodes v and w such that adding the pair $\{v, w\}$ still results in a legal sticker complex. Intuitively, when a complex is saturated, hybridization is finished in the complex.

Representation of Data Entries. Dimension of a Complex: The three disjoint parts of the alphabet $\Sigma = \Lambda \cup \Omega \cup \Theta$ serve distinct roles. Nodes labeled with tags from Θ indicate regions in the complex that have a function for data manipulation, as cleavage sites, or sites where stickers can anneal so as to circularize or concatenate strands. Nodes labeled with attribute names from Ω are used as annotations to data entries. Finally, the data entries themselves are represented using nodes labeled by atomic value symbols from Λ.

Atomic value symbols fulfill the same function as bits in a digital computer. A sequence of atomic value symbols represent a value, much like `100` is the binary representation of the number 8. Similar to the word size (number of bits) used in a digital computer to represent single data elements (such as integers), we will use sequences of atomic value symbols of a fixed length ℓ, called the dimension. Let $s = s_1 \dots s_\ell$ be a sequence of ℓ consecutive nodes of a strand of a sticker complex. If all nodes are labeled with atomic value symbols, s is called an *ℓ-core*. Let $s = s_0 \dots s_{\ell+1}$ be a sequence of $\ell+2$ consecutive nodes of a strand of a sticker complex. Such a sequence is called an *ℓ-vector* if s_0 is labeled with $\#_3$, $s_{\ell+1}$ is labeled with $\#_4$ and $s_1 \dots s_\ell$ is an ℓ-core.

The *dimension* is now defined as follows. For a fixed value of ℓ, we say that sticker complex C has dimension ℓ, if all nodes labeled with an atomic value symbol occur in an ℓ-vector. We then call C an *ℓ-complex.*

3 DNAQL

DNAQL [13] is an applicative programming language for expressing functions from ℓ-complexes to ℓ-complexes. A crucial feature of DNAQL is that the same program can be applied uniformly to complexes of any dimension ℓ. DNAQL is not computationally complete, as it is meant as a query language and not a general-purpose programming language. The language is based on a basic set of operations on complexes, some distinguished constants, an emptiness test (if-then-else), let-variable binding, counters that can count up to the dimension of the complex, and a limited for-loop for iterating over a counter. The syntax of DNAQL is given in Figure 1. Note that expressions can contain two kinds of variables: variables standing for complexes, and counters, ranging from 1 to the dimension. Complex variables can be bound by let-constructs, and counters can be bound by for-constructs. The free (unbound) complex variables of a DNAQL expression stand for its inputs. A DNAQL *program* is a DNAQL expression without free counters. So, in a program, all counters are introduced by for-loops.

$$
\begin{array}{rcl}
\langle \mathit{expression} \rangle & ::= & \langle \mathit{complexvar} \rangle \mid \langle \mathit{foreach} \rangle \mid \langle \mathit{if} \rangle \mid \langle \mathit{let} \rangle \mid \langle \mathit{operator} \rangle \mid \langle \mathit{constant} \rangle \\
\langle \mathit{foreach} \rangle & ::= & \texttt{for } \langle \mathit{complexvar} \rangle := \langle \mathit{expression} \rangle \texttt{ iter } \langle \mathit{counter} \rangle \texttt{ do } \langle \mathit{expression} \rangle \\
\langle \mathit{if} \rangle & ::= & \texttt{if empty}(\langle \mathit{complexvar} \rangle) \texttt{ then } \langle \mathit{expression} \rangle \texttt{ else } \langle \mathit{expression} \rangle \\
\langle \mathit{let} \rangle & ::= & \texttt{let } x := \langle \mathit{expression} \rangle \texttt{ in } \langle \mathit{expression} \rangle \\
\langle \mathit{operator} \rangle & ::= & ((\langle \mathit{expression} \rangle) \cup (\langle \mathit{expression} \rangle)) \mid ((\langle \mathit{expression} \rangle) - (\langle \mathit{expression} \rangle)) \\
& \mid & \texttt{hybridize}(\langle \mathit{expression} \rangle) \mid \texttt{ligate}(\langle \mathit{expression} \rangle) \mid \texttt{flush}(\langle \mathit{expression} \rangle) \\
& \mid & \texttt{split}(\langle \mathit{expression} \rangle, \langle \mathit{splitpoint} \rangle) \mid \texttt{block}(\langle \mathit{expression} \rangle, \Sigma - \Lambda) \\
& \mid & \texttt{blockfrom}(\langle \mathit{expression} \rangle, \Sigma - \Lambda) \mid \texttt{blockexcept}(\langle \mathit{expression} \rangle, \langle \mathit{counter} \rangle) \\
& \mid & \texttt{cleanup}(\langle \mathit{expression} \rangle) \\
\langle \mathit{constant} \rangle & ::= & \Sigma^+ \mid \left(\overline{\Sigma} - \overline{\Lambda}\right)\left(\overline{\Sigma} - \overline{\Lambda}\right) \mid \texttt{immob}(\overline{\Sigma}) \mid \texttt{empty} \\
\langle \mathit{splitpoint} \rangle & ::= & \#_2 \mid \#_3 \mid \#_4 \mid \#_6 \mid \#_8
\end{array}
$$

Fig. 1. Syntax of DNAQL

The constant expressions provide particular complexes as constants. A word $w \in \Sigma^+$ stands for a single, linear, positive strand that spells the word w. A two-letter word $\bar{a}\bar{b}$, for $a, b \in \Sigma - \Lambda$, stands for a single, linear, negative strand of length two of the $1 \to 2$ with $\lambda(1) = \bar{b}$ and $\lambda(2) = \bar{a}$. The expression $\texttt{immob}(\bar{a})$, for $a \in \Sigma$, stands for a single, negative, immobilized node labeled $\bar{a}$: we call such a node a *probe.* The expression `empty` stands for the empty complex. The split operation is implemented by a restriction enzyme. As the number of restriction enzymes is limited, and to ensure biological feasability of DNAQL, we allow only a limited number of split points.

The operation $\cup$ takes the disjoint union of two complexes. The difference $C - D$ of complexes C and D, which may be implemented using a subtractive hybridization technique [10], keeps only the strands of C that do not appear in D, and is only well defined when C and D consist solely of positive, equal-length strands. The operation `hybridize` performs hybridization as formalized [5] and extended here to take immobilized components into account, and may be undefined due to nonterminating behavior. Moreover, `ligate` behaves as ligase; `flush` removes supernatant (keeps only immobilized components), and `split` cleaves complexes. The blocking operations block a single node (`block`) or block a range starting from a primer (`blockfrom`); `blockexcept`(C, i) blocks, in each ℓ-vector $s_0, s_1, \dots, s_\ell, s_{\ell+1}$ in the ℓ-complex C, all nodes except s_i. For the blocking operations to be well defined, the complex must be saturated. Finally, `cleanup` undoes matchings and blockings and removes all strands except the longest positive ones.

The for-loop iterates its body with the counter running from 1 to ℓ, thus allowing access to specific bits in data entries with the aid of the `blockexcept` construct.

Example 1. We give an example of a DNAQL program, over the input variables x_1 and x_2, with a behavior similar to the selection operator and the cartesian product operator from the relational algebra. Below, a and b are assumed to be atomic value symbols.

```
let $y_1$ := cleanup(flush(hybridize($x_1 \cup$ immob($\bar{a}$)))) in
let $y_2$ := cleanup(flush(hybridize($x_2 \cup$ immob($\bar{b}$)))) in
if empty($y_1$) then empty else
if empty($y_2$) then empty else
cleanup(ligate(hybridize($y_1 \cup y_2 \cup \overline{\#_5\#_1}$)))
```

Assume complex C_1 holds a set of strands of the form $\#_3{*}\#_4\#_5$, where $*$ stands for a data entry in the form of an ℓ-core, and C_2 similarly holds a set of strands of the form $\#_1\#_3{*}\#_4$. Then the program applied to C_1 and C_2 filters from C_1 (C_2) the strands whose data entry contains the letter a (b); if both intermediate results are nonempty, the program then uses the stickers $\overline{\#_5\#_1}$ to concatenate each remaining strand from C_1 with each remaining strand from C_2.

4 Sticker Complex Types

Intuitively, a sticker complex type is an ℓ-complex where all data entries have been replaced by wildcards. What remains is a structural description of the components that may appear in the complex, with attribute names and tags explicit, but the dimension and actual values of data entries hidden. In order to obtain a powerful type system for DNAQL, these "weak" types S are augmented to "strong" types that have an indication $\odot$ of the mandatory components, which must occur, and a bit $\mathfrak{h}$ indicating that the type is saturated. The former is needed to type common DNAQL programs that use hybridization, and the latter is needed to type blocking operators in a DNAQL program.

Formally, we begin by introducing four symbols assumed not present in $\Sigma \cup \bar{\Sigma}$:

1. $*$ (*free*) represents an ℓ-core with none of the nodes matched or blocked;
2. $\underline{*}$ (*blocked*) represents an ℓ-core with all nodes blocked;
3. $\hat{*}$ (*open*) is the result of a block-except operator on an ℓ-core;

Let N denote the set $\{*, \underline{*}, \hat{*}\}$. The positive alphabet without atomic value symbols, but with the above new symbols is denoted $\Sigma_N = \Omega \cup \Theta \cup N$.

The fourth new symbol, denoted by '?' will be used to represent a probe, i.e., a single negative atomic value symbol that has been immobilized. The negative alphabet without the negative atomic value symbols, but with ? is denoted $\overline{\Sigma_N} = \overline{\Omega} \cup \overline{\Theta} \cup \{?\}$. Note that ? is considered to be a negative symbol. The complementarity relation is extended by $\overline{*} = ?$ and $\overline{\hat{*}} = ?$. Complementarity is thus no longer a bijection, but a relation.

A *sticker complex type* is very similar to a sticker complex: it is a structure $S = (V, L, \lambda, \mu, \iota, \beta)$ that satisfies the same definition as that of a sticker complex with the following exceptions:

– the range of the node labeling function λ is now $\Sigma_N \cup \overline{\Sigma_N}$ instead of $\Sigma \cup \overline{\Sigma}$;
– $\beta \subseteq V$ is not allowed to contain nodes labeled with a symbol from N;
– a node can be labeled '?' only if it is immobilized;
– there are no redundant components.

Next, we define the important notion of when a sticker complex $C = (V, L, \lambda, \mu, \iota, \beta)$ of some dimension ℓ is said to be well typed. Thereto, recall the intuitive meaning of the new symbols $\{*, \underline{*}, \hat{*}, ?\}$. Formally, consider an ℓ-core r occurring in C. We say that r is of type $*$ if no node of r is involved in μ nor in β; r is of type $\underline{*}$ if all nodes of r belong to β; and r is of type $\hat{*}$ if all but one node of r belong to β. Now we call C *well typed* if every ℓ-core occurring in C is of type $*$, $\underline{*}$ or $\hat{*}$. Moreover, if C is well typed, we define *stype*(C) as the sticker complex type obtained by replacing every ℓ-core occurring in C by a single node labeled by the type of the ℓ-core ($*$, $\underline{*}$ or $\hat{*}$), and replacing the label of any probe by ?.

The subsumption relation among sticker complexes (Section 2) can be adapted naturally to sticker complex types. We finally say that a well-typed sticker complex C *is of* some sticker complex type S, denoted by $C : S$, if *stype*(C) is subsumed by S. For sticker complex C, *stype*(C) is the "smallest" type, in the sense that there is no sticker complex type S' such that $C : S'$ and S' is strictly subsumed by S.

A sticker complex type is weak, in the sense that any well-typed sticker complex having as *stype* a subset of the components of a sticker complex type is of that type. In particular, the empty sticker complex is of every sticker complex type. This is too weak to type common DNAQL programs involving hybridization, where we need to know about components that are sure to be present. Thereto, we define a *strong sticker complex type* as a triple $\tau = (S, \odot, \mathfrak{h})$, where S is a sticker complex type, $\odot$ is a sticker complex type subsumed in S, $\mathfrak{h}$ is a boolean, and moreover if $\mathfrak{h} = true$, then $C \cup \odot$ is saturated for all every component C of S. Sticker complex type S is called the *weak type* of τ, the components of $\odot$ are called *mandatory* in τ, and $\mathfrak{h}$ is called the $\mathfrak{h}$-bit of τ.

A type τ is called *saturated* if all complexes having type τ are saturated.

For a well-typed complex C and a strong sticker complex type $\tau = (S, \odot, \mathfrak{h})$, we now say that C has type τ if C is of type S; the complex $\odot$ is subsumed by $\mathit{stype}(C)$; and C is saturated if $\mathfrak{h} = \mathit{true}$.

From now on, we will refer to sticker complex types as *weak types* and to strong sticker complex types as *types*.

5 A Type System for DNAQL

Given a DNAQL program $e(x_1, \dots, x_k)$ with free complex variables $x_1, \dots, x_k$, and given types $\tau_1, \dots, \tau_k$ for the respective input variables, we would like to determine whether e is *safe* under these input types, meaning that for any dimension ℓ and for any input complexes $C_1, \dots, C_k$ of dimension ℓ and of the given types $\tau_1, \dots, \tau_k$, the result $e(C_1, \dots, C_k)$ on these inputs is well defined. Since types do not restrict the dimension of complexes, if a type involves wildcards, there are infinitely many complexes of that type. Hence safety is not easy to guarantee, indeed safety is undecidable: this will follow from our later Theorem 2 and an easy reduction from satisfiability of well-typed relational algebra expressions, which is undecidable [1].

The best we can do is to come up with a type system that tries to infer the output types from given input types. We have developed a type system that, given e and $\Gamma = \tau_1, \dots, \tau_k$ as above, determines whether e is *well-typed* under Γ, and, if so, infers an output type τ, this is denoted by $\Gamma \vdash e : \tau$. The DNAQL type system enjoys the following soundness property:

Theorem 1. *If $\Gamma \vdash e : \tau$ then e is safe under Γ, and the resulting complex of e applied to any inputs of type Γ will be of type τ.*

The full type-checking system and soundness proof are omitted from this conference paper. Here we give some intuitions and examples.

Obviously devising a sound type-checking system in itself is no challenge, as it suffices to judge every program ill typed so that soundness becomes trivial! The challenge is to have a sound type-checking system that still judges most useful DNAQL programs to be typed. Our type system checks DNAQL expressions bottom-up by applying the DNAQL operations on complexes symbolically, on the type level. The operations may fail on the type level, in case we cannot deduce from the type that the operation will be well-defined on all inputs of the given type. If we can deduce well-definedness, we output a tight result type and the type-checking continues. Furthermore, the typing inference made for if-then-else constructs, shown in Figure 2, are designed so as to maximally benefit from knowledge that complexes are nonempty. These rules maximally infer the presence of mandatory components, which allows later hybridization operations to be typed.

Example 2. Recall the program from Example 1 in Section 3. Consider the weak types $S_1 = \#_3{*}\#_4\#_5$ and $S_2 = \#_1\#_3{*}\#_4$. The program is well-typed under

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x := \tau_1] \vdash e_2 : \tau_1}{\Gamma \vdash \mathtt{for}\ x := e_1\ \mathtt{iter}\ i\ \mathtt{do}\ e_2 : \tau_1}$$

$$\frac{\Gamma \vdash x : (S_x, \emptyset, \mathfrak{h}_x) \qquad S_x = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \qquad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash \mathtt{if\ empty}(x)\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \tau_1}$$

$$\frac{\Gamma \vdash x : (S_x, \odot_x, \mathfrak{h}_x) \qquad \odot_x \neq \emptyset \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathtt{if\ empty}(x)\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : \tau_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash x : (S_x, \odot_x, \mathfrak{h}_x) \qquad \odot_x = \emptyset \qquad |comp(S_x)| = 1 \\ \Gamma \vdash e_1 : (S_1, \odot_1, \mathfrak{h}_1) \qquad \Gamma[x := (S_x, comp(S_x), \mathfrak{h}_x)] \vdash e_2 : (S_2, \odot_2, \mathfrak{h}_2)\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{if\ empty}(x)\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : (S_1 \cup S_2, \odot_1 \cap \odot_2, \mathfrak{h}) \\ \mathfrak{h} = (S_1 \cup S_2 \text{ is saturated})\end{array}}$$

$$\frac{\begin{array}{c}\Gamma \vdash x : (S_x, \odot_x, \mathfrak{h}_x) \\ \odot_x = \emptyset \qquad |comp(S_x)| > 1 \qquad \Gamma \vdash e_1 : (S_1, \odot_1, \mathfrak{h}_1) \qquad \Gamma \vdash e_2 : (S_2, \odot_2, \mathfrak{h}_2)\end{array}}{\begin{array}{c}\Gamma \vdash \mathtt{if\ empty}(x)\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : (S_1 \cup S_2, \odot_1 \cap \odot_2, \mathfrak{h}) \\ \mathfrak{h} = (S_1 \cup S_2 \text{ is saturated})\end{array}}$$

Fig. 2. Typing relation for the control flow of DNAQL

the types $\tau_1 = (S_1, S_1, \mathit{false})$ for x_1 and $\tau_2 = (S_2, \emptyset, \mathit{false})$ for x_2. Since S_1 is mandatory in τ_1, we know that input x_1 will be nonempty. Note also that the $\mathfrak{h}$-bit in τ_1 is false, although complexes of type S_1 are necessarily saturated. The subexpression $e_1 = \mathtt{hybridize}(x_1 \cup \mathtt{immob}(\bar{a}))$ is typed as $(S_1^?, \emptyset, \mathit{true})$, where $S_1^?$ consists of the following components: (i) S_1 itself; (ii) $\mathtt{immob}(?)$; and the complex formed by the union of (i) and (ii) and matching the node $*$ with the node ?. Note that there are no mandatory components, since on inputs without an a, only (i) and (ii) will occur, whereas on inputs where all strands have an a, only (iii) will occur. The $\mathfrak{h}$-bit is now true since a complex resulting from hybridization is always saturated. Applying `flush` to e_1 yields output type $(S_1^{?\prime}, \emptyset, \mathit{true})$, where $S_1^{?\prime}$ consists of components (ii) and (iii) above. Finally the variable y_1 in the `let`-construct is assigned the type $(S_1, \emptyset, \mathit{true})$. Similarly, y_2 gets the type $(S_2, \emptyset, \mathit{true})$. Yet, by the design of the if-then-else typing rules, the subexpression on the last line of the program will be typed under the strong types $(S_1, S_1, \mathit{true})$ for y_1 and $(S_2, S_2, \mathit{true})$ for y_2. Because all components are now mandatory, the type inferred for subexpression $\mathtt{hybridize}(y_1 \cup y_2 \cup \overline{\#_5\#_1})$ will be $(S_{12}, S_{12}, \mathit{true})$, where S_{12} is the weak type obtained from the union of S_1, S_2 and $\overline{\#_5\#_1}$ by matching the $\#_5$ and $\overline{\#_5}$ and the $\#_1$ and $\overline{\#_1}$ nodes, respectively. After ligate and cleanup the output type is (S, S, true) where S consists of the single strand $\#_3{*}\#_4\#_5\#_1\#_3{*}\#_4$. The final output type of the entire program, combining the then- and else-branches, is $(S, \emptyset, \mathit{true})$.

For another example, consider the program

$$\texttt{hybridize}(\texttt{hybridize}(x \cup \bigcup_{a \in \Lambda} \texttt{immob}(\overline{a})) \cup \overline{\#_3\#_4}).$$

This program is ill typed under the type $\tau = (S, S, \mathit{true})$ for x with $S = \#_3{*}\#_4$. Indeed, the nested hybridize subexpression is still well-typed, yielding the output type $(S^?, \emptyset, \mathit{true})$ without any mandatory components. Adding the component $\overline{\#_3\#_4}$ to $S^?$, however, yields a complex with nonterminating hybridization [5], so the type checker will reject the top-level hybridize.

Yet, this program will have a well-defined output on every input C of type τ. Indeed, every strand in C contains some $a \in \Lambda$, so the minimal type of the result of the nested hybridize will actually have a single complex component formed by the union of S and $\texttt{immob}(?)$ with $*$ and $?$ matched. Then the top-level hybridize will terminate since each sticker complex can have at most immobilized node.

This example shows that well-defined programs may be ill typed; this is unavoidable in general since safety is undecidable.

6 Relational Algebra Simulation

In this section we strengthen an earlier result [13] to the effect that relational algebra expressions can be simulated by DNAQL programs: we show that the simulation is already possible by *well-typed* programs. This illustrates the power of our type system (cf. the comment made after Theorem 1).

Basically we assume a universe $\mathbb{U}$ of data elements. A *relation schema* R is a finite set of attribute names. We can use the same alphabet Ω for these attribute names. A *tuple* over R is a mapping from R to U. A *relation instance* over R is a finite set of tuples over R.

The syntax of the relational algebra [9,12,1] is generated by the following grammar:

$$e ::= x \mid (e \cup e) \mid (e - e) \mid (e \times e) \mid \sigma_{A=B}(e) \mid \widehat{\pi}_A(e) \mid \rho_{A/B}(e)$$

Here, x stands for a relation variable, and A and B stand for attributes. Our version of the relational algebra is slightly nonstandard in that our version of projection ($\widehat{\pi}$) projects away some given attribute, as opposed to the standard projection which projects on some given subset of the attributes.

The relational algebra obeys a simple type system where expressions are typed by relation schemes [6]. Given a relational algebra expression $e(x_1, \ldots, x_k)$ over the input relation variables $x_1, \ldots, x_k$, and given input relation schemas $\Gamma = R_1, \ldots, R_k$, we can determine whether e is well typed under Γ, and, if so, infer a result relation schema R, denoted by $\Gamma \vdash e : R$ or just $e : R$ if Γ is understood. The typing rules are simple. If $e_1 : R$ and $e_2 : R$ then $(e_1 \cup e_2) : R$ and $(e_1 - e_2) : R$; if $e_1 : R_1$ and $e_2 : R_2$ for disjoint R_1 and R_2, then $(e_1 \times e_2) : R_1 \cup R_2$; if $e : R$ and $A, B \in R$ then $\sigma_{A=B}(e) : R$; if $e : R$ and $A \in R$ then $\widehat{\pi}_A(e) : R \setminus \{A\}$; if $e : R$ and $A \in R$ and $B \notin R$ then $\rho_{A/B}(e) : (R \setminus \{A\}) \cup \{B\}$.

The semantics of the relational algebra is well known and we omit a formal definition. Provided $\Gamma \vdash e : R$, on any input relation instances $I_1, \dots, I_k$ over $R_1, \dots, R_k$, the result $e(I_1, \dots, I_k)$ is well defined and is a relation instance over R.

We want now to represent relation instances by complexes. We will store data elements as vectors of atomic value symbols. So formally, we use the set of strings Λ^* as our universe $\mathbb{U}$. Then a tuple t (relation instance I) is said to be of dimension ℓ if all data elements appearing in $t(I)$ are strings of length ℓ. Let t be a tuple of dimension ℓ over relation schema R. We may assume a fixed order on all attribute names. Let the attributes of R in order be $A, \dots, B$. We then represent t by the following ℓ-complex:

$$\mathit{complex}(t) = \#_2 A \#_3 t(A) \#_4 \dots \#_2 B \#_3 t(B) \#_4.$$

A relation instance I of dimension ℓ is then represented by the ℓ-complex

$$\mathit{complex}(I) = \bigcup \{\mathit{complex}(t) \mid t \in I\}.$$

Note that $\mathit{complex}(I)$ is of strong type $\tau_R = (\mathit{complex}(R), \emptyset, \mathit{true})$, where $\mathit{complex}(R)$ is $\#_2 A \#_3 {*} \#_4 \dots \#_2 B \#_3 {*} \#_4$.

We are now in a position to state our main theorem.

Theorem 2. *Let $e(x_1, \dots, x_k)$ be an arbitrary well-typed relational algebra expression, let $\Gamma = R_1, \dots, R_k$ be input relation schemas, and let R be an output relation schema such that $\Gamma : e \vdash R$. Then e can be translated into a DNAQL program $e^{DNA}(x_1, \dots, x_k)$, such that the following holds:*

1. *e^{DNA} is well-typed, i.e., $\tau_{R_1}, \dots, \tau_{R_k} \vdash e^{DNA} : \tau_R$.*
2. *e^{DNA} simulates e uniformly over all dimensions ℓ, i.e., for each natural number ℓ and for any ℓ-dimensional input relation instances $I_1, \dots, I_k$ over $R_1, \dots, R_k$ respectively,*

$$e^{DNA}(\mathit{complex}(I_1), \dots, \mathit{complex}(I_k)) = \mathit{complex}(e(I_1, \dots, I_k))$$

(up to isomorphism).

The proof of the first statement is omitted in this conference paper. The second statement has been proven in previous work [13].

7 Conclusion

An interesting problem is to understand the precise expressive power of well-typed DNAQL programs. Theorem 2 provides a lower bound; a corresponding upper bound, to the effect that every well-typed DNAQL program can be simulated in the relational algebra (on relational structures representing the typed input complexes) would establish DNAQL as the DNA-computing equivalent of the relational algebra. We note that untyped operations, e.g., the difference operator applied to arbitrary complexes of unknown type, are strictly more powerful than the relational algebra.

On the practical level, the obvious research direction is to verify some non-trivial DNAQL programs experimentally, or simulate them in silico. Indeed, we have gone to great efforts to design an abstraction that is as plausible as possible. A static analysis of the error rates of DNAQL programs on the type level is another necessary topic for further research.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Adleman, L.: Molecular computation of solutions to combinatorial problems. Science 226, 1021–1024 (1994)
3. Amos, M.: Theoretical and Experimental DNA Computation. Springer (2005)
4. Arita, M., Hagiya, M., Suyama, A.: Joining and rotating data with molecules. In: ICEC 1997 (1997)
5. Brijder, R., Gillis, J.J.M., Van den Bussche, J.: Graph-Theoretic Formalization of Hybridization in DNA Sticker Complexes. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 49–63. Springer, Heidelberg (2011)
6. Van den Bussche, J., Van Gucht, D., Vansummeren, S.: A crash course in database queries. In: PODS 2007 (2007)
7. Chen, J., Deaton, R., Wang, Y.Z.: A DNA-based memory with in vitro learning and associative recall. Natural Computing 4(2), 83–101 (2005)
8. Condon, A., Corn, R., Marathe, A.: On combinatorial DNA word design. Journal of Computational Biology 8(3), 201–220 (2001)
9. Date, C.: An Introduction to Database Systems. Addison-Wesley (2004)
10. Diatchenko, L., Lau, Y., et al.: Suppression subtractive hybridization: a method for generating differentially regulated or tissue-specific cDNA probes and libraries. PNAS 93(12), 6025–6030 (1996)
11. Dirks, R., Pierce, N.: Triggered amplification by hybridization chain reaction. PNAS 101(43), 15275–15278 (2004)
12. Garcia-Molina, H., Ullman, J., Widom, J.: Database Systems: The Complete Book. Prentice Hall (2009)
13. Gillis, J.J.M., Van den Bussche, J.: A Formal Model for Databases in DNA. In: Horimoto, K., Nakatsui, M., Popov, N. (eds.) ANB 2010. LNCS, vol. 6479, pp. 18–37. Springer, Heidelberg (2012)
14. Gunter, C., Mitchell, J. (eds.): Theoretical Aspects of Object-Oriented Programming. MIT Press (1994)
15. Jonoska, N., McColm, G., Staninska, A.: On stoichiometry for the assembly of flexible tile DNA complexes. Natural Computing 10(3), 1121–1141 (2011)
16. Paun, G., Rozenberg, G., Salomaa, A.: DNA Computing. Springer (1998)
17. Pierce, B.: Types and Programming Languages. MIT Press (2002)
18. Qian, L., Soloveichik, D., Winfree, E.: Efficient Turing-Universal Computation with DNA Polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
19. Reif, J.: Parallel biomolecular computation: models and simulations. Algorithmica 25(2-3), 142–175 (1999)

20. Reif, J.H., LaBean, T.H., Pirrung, M., Rana, V.S., Guo, B., Kingsford, C., Wickham, G.S.: Experimental Construction of Very Large Scale DNA Databases with Associative Search Capability. In: Jonoska, N., Seeman, N.C. (eds.) DNA 7. LNCS, vol. 2340, pp. 231–247. Springer, Heidelberg (2002)
21. Rozenberg, G., Spaink, H.: DNA computing by blocking. TCS 292, 653–665 (2003)
22. Sager, J., Stefanovic, D.: Designing Nucleotide Sequences for Computation: A Survey of Constraints. In: Carbone, A., Pierce, N.A. (eds.) DNA 11. LNCS, vol. 3892, pp. 275–289. Springer, Heidelberg (2006)
23. Shortreed, M., et al.: A thermodynamic approach to designing structure-free combinatorial DNA word sets. Nucleic Acids Research 33(15), 4965–4977 (2005)
24. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. PNAS online (2010)
25. Yamamoto, M., Kita, Y., Kashiwamura, S., Kameda, A., Ohuchi, A.: Development of DNA Relational Database and Data Manipulation Experiments. In: Mao, C., Yokomori, T. (eds.) DNA12. LNCS, vol. 4287, pp. 418–427. Springer, Heidelberg (2006)

Deterministic Function Computation with Chemical Reaction Networks*

Ho-Lin Chen[1], David Doty[2], and David Soloveichik[3]

[1] National Taiwan University, Taipei, Taiwan
holinc@gmail.com
[2] California Institute of Technology, Pasadena, California, USA
ddoty@caltech.edu
[3] University of California, San Francisco, San Francisco, CA, USA
david.soloveichik@ucsf.edu

Abstract. Chemical reaction networks (CRNs) formally model chemistry in a well-mixed solution. CRNs are widely used to describe information processing occurring in natural cellular regulatory networks, and with upcoming advances in synthetic biology, CRNs are a promising language for the design of artificial molecular control circuitry. Nonetheless, despite the widespread use of CRNs in the natural sciences, the range of computational behaviors exhibited by CRNs is not well understood.

CRNs have been shown to be efficiently Turing-universal when allowing for a small probability of error. CRNs that are guaranteed to converge on a correct answer, on the other hand, have been shown to decide only the semilinear predicates. We introduce the notion of function, rather than predicate, computation by representing the output of a function $f : \mathbb{N}^k \to \mathbb{N}^l$ by a count of some molecular species, i.e., if the CRN starts with $x_1, \ldots, x_k$ molecules of some "input" species $X_1, \ldots, X_k$, the CRN is guaranteed to converge to having $f(x_1, \ldots, x_k)$ molecules of the "output" species $Y_1, \ldots, Y_l$. We show that a function $f : \mathbb{N}^k \to \mathbb{N}^l$ is deterministically computed by a CRN if and only if its graph $\{(\mathbf{x}, \mathbf{y}) \in \mathbb{N}^k \times \mathbb{N}^l \mid f(\mathbf{x}) = \mathbf{y}\}$ is a semilinear set.

Finally, we show that each semilinear function f can be computed on input $\mathbf{x}$ in expected time $O(\text{polylog } \|\mathbf{x}\|_1)$.

1 Introduction

The engineering of complex artificial molecular systems will require a sophisticated understanding of how to *program chemistry*. A natural language for describing abstract chemical systems in a well-mixed solution is that of (finite) chemical reaction networks (CRNs), i.e., finite sets of chemical reactions such as $A+B \to A+C$. When the goal is to model the behavior of individual molecules in

* The first author was supported by the Molecular Programming Project under NSF grant 0832824, the second and third authors were supported by a Computing Innovation Fellowship under NSF grant 1019343. The third author was supported by NIGMS Systems Biology Center grant P50 GM081879.

a well-mixed solution, CRNs are assigned semantics through *stochastic chemical kinetics* [7], in which reactions occur probabilistically with rate proportional to the product of the molecular count of their reactants and inversely proportional to the volume of the reaction vessel.

Traditionally CRNs have been used as a descriptive language to analyze naturally occurring chemical systems. However, recent investigations of CRNs as a programming language for engineering artificial chemical systems have shown CRNs to have surprisingly powerful computational ability. For example, bounded-space Turing machines can be simulated with an arbitrarily small, non-zero probability of error by a CRN with only a polynomial slowdown [1], and even Turing universal computation is possible with an arbitrarily small, non-zero probability of error over all time [11]. This is surprising since finite CRNs necessarily must represent binary data strings in a unary encoding, since they lack positional information to tell the difference between two molecules of the same species. Other work has investigated the power of CRNs to simulate Boolean circuits [9], digital signal processing [8], the (un)decidability of whether a CRN will reach a state where no further reaction is possible [13], and the robustness of CRNs to tolerate multiple copies of the network running in parallel [6]. Finally, recent work proposes concrete chemical implementations of arbitrary CRN programs, particularly using nucleic-acid strand-displacement cascades as the physical reaction primitive [5,12].

Angluin, Aspnes and Eisenstat [2] investigated the computational power of deterministic CRNs (under a different name, that of the equivalent distributed computing model known as *population protocols*). Some CRNs, when started in an initial configuration assigning nonnegative integer counts to each of k different input species, are guaranteed to converge on a single "yes" or "no" answer, in the sense that there are two special "voting" species L^1 and L^0 so that eventually either L^1 is present and L^0 absent to indicate "yes", or vice versa to indicate "no." The set of inputs $S \subseteq \mathbb{N}^k$ that cause the system to answer "yes" is then a representation of the decision problem solved by the CRN. Angluin, Aspnes and Eisenstat showed that the input sets S decidable by some CRN are precisely the *semilinear* subsets of $\mathbb{N}^k$ (see below).

We extend these prior investigations of decision problem or predicate computation to study deterministic *function* computation. Consider the three examples in Fig. 1(top). These CRNs have the property that they converge to the right answer no matter in what order the reactions occur. Formally, we say a function $f : \mathbb{N}^k \to \mathbb{N}^l$ is computed by a CRN $\mathcal{C}$ if the following is true. There are "input" species $X_1, \ldots, X_k$ and "output" species $Y_1, \ldots, Y_l$ such that, if $\mathcal{C}$ is initialized with $x_1, \ldots, x_k$ copies of $X_1, \ldots, X_k$, then it is guaranteed to reach a configuration in which the counts of $Y_1, \ldots, Y_l$ are described by the vector $f(x_1, \ldots, x_k)$, and these counts never again change. For example, the CRN $\mathcal{C}$ with the single reaction $X \to 2Y$ computes the function $f(x) = 2x$ in the sense that, if $\mathcal{C}$ starts in an initial configuration with x copies of X and 0 copies of Y, then $\mathcal{C}$ is guaranteed to stabilize to a configuration with $2x$ copies of Y (and no copies of X). Similarly, the function $f(x) = \lfloor x/2 \rfloor$ is computed by the single reaction

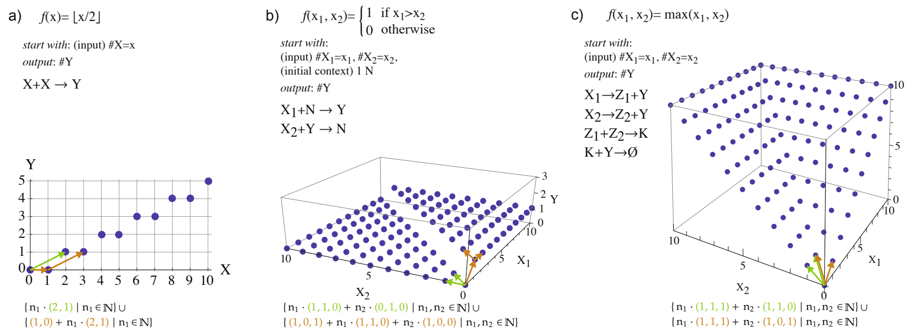


Fig. 1. Examples of deterministically computable functions. (Top) Three functions and examples of CRNs deterministically computing them. Example (a) computes via the relative stoichiometry of reactants and products of a single reaction. In example (b), a single molecule, converted between N or Y forms, goes back and forth consuming X_1 and X_2, and whether it gets stuck in the N or Y form indicates the excess of X_1 or X_2. To see that the CRN in (c) correctly computes the maximum, note that the first two reactions eventually produce $x_1 + x_2$ molecules of Y, while the third reaction eventually produces $\min(x_1, x_2)$ molecules of K. Thus the last reaction eventually consumes $\min(x_1, x_2)$ molecules of Y leaving $x_1 + x_2 - \min(x_1, x_2) = \max(x_1, x_2)$ Y's. (Bottom) Graphs of the three functions. The set of points belonging to the graph of each of these functions is a semilinear set. Under each plot this semilinear set is written in the form of a union of linear sets corresponding to equation 1.1. The defining vectors are shown as colored arrows in the graph.

$2X \to Y$ (Fig. 1(a)), in that the final configuration is guaranteed to have exactly $\lfloor x/2 \rfloor$ copies of Y (and 0 or 1 copies of X, depending on whether x is even or odd). Note that the CRN in Fig. 1(b) can be thought to compute the predicate "$x_1 > x_2$?" with species N voting "no" and Y voting "yes". Examples (a) and (c) cannot be phrased in the form of predicate computation.

What do the functions in Fig. 1(top) have in common such that the CRNs computing them can inevitably progress to the right answer no matter what order the reactions occur in? What other functions can be computed similarly? Answering these questions may seem difficult because it appears like the three examples operate on different principles and use different ideas.

We show that the functions deterministically computable by CRNs are precisely the semilinear functions, where we define a function to be *semilinear* if its *graph* $\{(\mathbf{x}, \mathbf{y}) \in \mathbb{N}^k \times \mathbb{N}^l \mid f(\mathbf{x}) = \mathbf{y}\}$ is a semilinear subset of $\mathbb{N}^k \times \mathbb{N}^l$. (See Fig. 1(bottom) for the graphs of the functions just mentioned.) This means that the graph of the function is a union of a finite number of *linear* sets – i.e. sets that can be written in the form

$$\left\{ \mathbf{b} + n_1\mathbf{u}^1 + \ldots + n_p\mathbf{u}^p \;\middle|\; n_1, \ldots, n_p \in \mathbb{N} \right\} \tag{1.1}$$

for some vectors $\mathbf{b}, \mathbf{u}^1, \ldots, \mathbf{u}^p \in \mathbb{N}^k \times \mathbb{N}^l$. Fig. 1(bottom) shows the graphs of the three example functions expressed as a union of sets of this form. Informally, semilinear functions can be thought of as "piecewise linear functions" with a finite number of pieces, and linear domains of each piece.[1]

This characterization implies, for example, that such functions as $f(x_1, x_2) = x_1 x_2$ or $f(x) = x^2$ are not deterministically computable. For instance, the graph of the function $f(x_1, x_2) = x_1 x_2$ consists of infinitely many lines of different slopes, and thus, while each line is a linear set, the graph is not a finite union of linear sets.

Our result employs the predicate computation characterization of Angluin, Aspnes and Eisenstat [2], together with some nontrivial additional technical machinery. In particular, we introduce the notion of "reducing" the computation of one CRN to that of another, essentially using one CRN as a black box in constructing another. This is more difficult than in standard programming languages since there is in general no way of knowing when a CRN is done computing, or whether it will change its answer in the future.

Having established what functions are deterministically computable by CRNs given unbounded time, we turn our attention to the time required for CRNs to converge to the answer. We show that every semilinear function can be deterministically computed on input $\mathbf{x}$ in expected time polylog($\|\mathbf{x}\|$). This is done by a similar technique used by Angluin, Aspnes, and Eisenstat [2] to show the equivalent result for predicate computation. They run a slow deterministic computation in parallel with a fast randomized computation, allowing the deterministic computation to compare the two answers and update the randomized answer only if it is incorrect, which happens with low probability. However, novel techniques are required since it is not as simple to "nondestructively compare" two integers (so that the counts are only changed if they are unequal) as to compare two Boolean values.

2 Preliminaries

Throughout the paper we use both superscripts and subscripts to index variables to make for easier reading; the superscript never means exponentiation. Apologies in advance.

Given a vector $\mathbf{x} \in \mathbb{N}^k$, let $\|\mathbf{x}\| = \|\mathbf{x}\|_1 = \sum_{i=1}^{k} |\mathbf{x}_i|$, where $\mathbf{x}_i$ denotes the ith coordinate of $\mathbf{x}$. If $f : \mathbb{N}^k \to \mathbb{N}^l$ is a function, define the *graph* of f to be the set $\left\{ (\mathbf{x}, \mathbf{y}) \in \mathbb{N}^k \times \mathbb{N}^l \mid f(\mathbf{x}) = \mathbf{y} \right\}$. A set $A \subseteq \mathbb{N}^k$ is *linear* if there exist vectors $\mathbf{b}, \mathbf{u}^1, \ldots, \mathbf{u}^p \in \mathbb{N}^k$ such that

$$A = \left\{ \mathbf{b} + n_1 \mathbf{u}^1 + \ldots + n_p \mathbf{u}^p \mid n_1, \ldots, n_p \in \mathbb{N} \right\}.$$

[1] Semilinear sets have a number of characterizations. They are often thought of as generalizations of arithmetic progressions. They are also exactly the sets that are definable in Presburger arithmetic [10]: the first-order theory of the natural numbers with addition. Equivalently, they are the sets accepted by boolean combinations of "modulo" and "threshold" predicates [2]. Semilinear functions are less well-studied. The "piecewise linear" intuitive characterization is formalized in Lemma 4.3.

A is *semilinear* if it is a finite union of linear sets. We say a partial function $f : \mathbb{N}^k \dashrightarrow \mathbb{N}^l$ is *affine* if there exist kl rational numbers $a_1^1, \ldots, a_k^l \in \mathbb{Q} \cap [0, \infty)$ and $l + k$ integers $b_1, \ldots, b_l, c_1, \ldots, c_k \in \mathbb{Z}$ such that for each $1 \leq j \leq l$, $\mathbf{y}_j = b_j + \sum_{i=1}^{k} a_i^j (\mathbf{x}_i + c_i)$. In other words, the graph of f, when projected onto the $(k+1)$-dimensional space defined by the k coordinates corresponding to $\mathbf{x}$ and the single coordinate corresponding to $\mathbf{y}_j$, is a subset of a k-dimensional hyperplane.

2.1 Chemical Reaction Networks

If Λ is a finite set (in this paper, of chemical species), we write $\mathbb{N}^\Lambda$ to denote the set of functions $f : \Lambda \to \mathbb{N}$. Equivalently, we view an element $C \in \mathbb{N}^\Lambda$ as a vector of $|\Lambda|$ nonnegative integers, with each coordinate "labeled" by an element of Λ. Given $X \in \Lambda$ and $C \in \mathbb{N}^\Lambda$, we refer to $C(X)$ as the *count of X in C*. We write $C \leq C'$ to denote that $C(X) \leq C'(X)$ for all $X \in \Lambda$. Given $C, C' \in \mathbb{N}^\Lambda$, we define the vector component-wise operations of addition $C + C'$, subtraction $C - C'$, and scalar multiplication nC for $n \in \mathbb{N}$. If $\Delta \subset \Lambda$, we view a vector $C \in \mathbb{N}^\Delta$ equivalently as a vector $C \in \mathbb{N}^\Lambda$ by assuming $C(X) = 0$ for all $X \in \Lambda \setminus \Delta$.

Given a finite set of chemical species Λ, a *reaction* over Λ is a triple $\alpha = \langle \mathbf{r}, \mathbf{p}, k \rangle \in \mathbb{N}^\Lambda \times \mathbb{N}^\Lambda \times \mathbb{R}^+$, specifying the stoichiometry of the reactants and products, respectively, and the *rate constant* k. If not specified, assume that $k = 1$ (this is the case for all reactions in this paper), so that the reaction $\alpha = \langle \mathbf{r}, \mathbf{p}, 1 \rangle$ is also represented by the pair $\langle \mathbf{r}, \mathbf{p} \rangle$. For instance, given $\Lambda = \{A, B, C\}$, the reaction $A + 2B \to A + 3C$ is the pair $\langle (1, 2, 0), (1, 0, 3) \rangle$. A *(finite) chemical reaction network (CRN)* is a pair $N = (\Lambda, R)$, where Λ is a finite set of chemical *species*, and R is a finite set of reactions over Λ. A *configuration* of a CRN $N = (\Lambda, R)$ is a vector $C \in \mathbb{N}^\Lambda$. We also write $\#_C X$ to denote $C(X)$, the *count* of species X in configuration C, or simply $\#X$ when C is clear from context.

Given a configuration C and reaction $\alpha = \langle \mathbf{r}, \mathbf{p} \rangle$, we say that α is *applicable* to C if $\mathbf{r} \leq C$ (i.e., C contains enough of each of the reactants for the reaction to occur). If α is applicable to C, then write $\alpha(C)$ to denote the configuration $C + \mathbf{p} - \mathbf{r}$ (i.e., the configuration that results from applying reaction α to C). If $C' = \alpha(C)$ for some reaction $\alpha \in R$, we write $C \to_N C'$, or merely $C \to C'$ when N is clear from context. An *execution* (a.k.a., *execution sequence*) $\mathcal{E}$ is a finite or infinite sequence of one or more configurations $\mathcal{E} = (C_0, C_1, C_2, \ldots)$ such that, for all $i \in \{1, \ldots, |\mathcal{E}| - 1\}$, $C_{i-1} \to C_i$. If a finite execution sequence starts with C and ends with C', we write $C \to_N^* C'$, or merely $C \to^* C'$ when the CRN N is clear from context. In this case, we say that C' is *reachable* from C.

Let $\Delta \subseteq \Lambda$. We say that $P \in \mathbb{N}^\Delta$ is a *partial configuration (with respect to Δ)*. We write $P = C \upharpoonright \Delta$ for any configuration C such that $C(X) = P(X)$ for all $X \in \Delta$, and we say that P is the *restriction of C to Δ*. Say that a partial configuration P with respect to Δ is *reachable* from configuration C' if there is a configuration C reachable from C' and $P = C \upharpoonright \Delta$. In this case, we write $C' \to^* P$.

An infinite execution $\mathcal{E} = (C_0, C_1, C_2, \ldots)$ is *fair* if, for all partial configurations P, if P is infinitely often reachable then it is infinitely often reached.[2] In other words, no reachable partial configuration is "starved". This definition of fairness is stricter than that used by Angluin, Aspnes, and Eisenstat [2], which used only full configurations rather than partial configurations. We choose this definition to prevent intuitively unfair executions from vacuously satisfying the definition of "fair" simply because of some species whose count is monotonically increasing with time (preventing any configuration from being infinitely often reachable).[3]

Note that the definition given above, applied to finite executions, deems all of them fair vacuously. We wish to distinguish between finite executions that can be extended by applying another reaction and those that cannot. Say that a configuration is *terminal* if no reaction is applicable to it. We say that a finite execution is *fair* if and only if it ends in a terminal configuration.

In order to show that CRNs can compute semilinear functions, we need to have some guarantee that the execution sequence is not chosen such that some reactions are simply not allowed to happen. (For example, $\{X \to 2Y, A \to B, B \to A\}$ cannot correctly compute $y = 2x$ if an "adversary" simply does not let the first reaction occur, always preferring the second or third.) We consider two models. In the first part of this paper, we follow [2] in applying the above condition of fairness on the allowed execution sequences, which captures combinatorially what we need of the execution sequence. In the second part of the paper (Section 4) we consider the kinetic model, which ascribes probabilities to execution sequences. The kinetic model also defines the time of reactions, allowing us to study the computational complexity of the CRN computation. Note that in the kinetic model, if the reachable configuration space is bounded for any start configuration (i.e. if from any starting configuration there are finitely many configurations reachable) then any observed execution sequence will be fair with probability 1. (This will be the case for our construction in Section 4.)

2.2 Stable Decidability of Predicates

We now review the definition of stable decidability of predicates introduced by Angluin, Aspnes, and Eisenstat [2]. Those authors use the term "stably *compute*", but we reserve the term "compute" to apply to the computation of non-Boolean functions. Intuitively, some species "vote" for a yes/no answer, and a CRN N is a stable decider if, for all initial configurations, N is guaranteed (under fair executions) to reach a consensus vote, which is potentially different for different initial configurations but consistent over all fair executions starting from a fixed initial configuration.

A *chemical reaction decider* (CRD) is a tuple $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon, \phi, \sigma)$, where (Λ, R) is a CRN, $\Sigma \subseteq \Lambda$ is the *set of input species*, $\Upsilon \subseteq \Lambda$ is the set of *voters*,

[2] i.e. $(\forall \Delta \subseteq \Lambda)(\forall P \in \mathbb{N}^{\Delta})[((\exists^{\infty} i \in \mathbb{N})\ C_i \to^* P) \implies ((\exists^{\infty} j \in \mathbb{N})\ P = C_j \upharpoonright \Delta)]$.

[3] Such a definition is unnecessary in the work of Angluin, Aspnes, and Eisenstat [2] because population protocols by definition have a finite state space, since they enforce that every reaction has precisely two reactants and two products.

$\phi : \Upsilon \to \{0,1\}$ is the *(Boolean) output function*, and $\sigma \in \mathbb{N}^{\Lambda \setminus \Sigma}$ is the *initial context*. Intuitively, the goal is for the CRD to get all voters to be eventually unanimous and correct (and for at least one to be present). An input to $\mathcal{D}$ will be a vector $I_0 \in \mathbb{N}^{\Sigma}$. Thus a CRD together with an input vector defines an initial configuration I defined by $I(X) = I_0(X)$ if $X \in \Sigma$, and $I(X) = \sigma(X)$ otherwise. We say that such a configuration is a *valid initial configuration*, i.e., $I \upharpoonright (\Lambda \setminus \Sigma) = \sigma$. If we are discussing a CRN understood from context to have a certain initial configuration I, we write $\#_0 X$ to denote $I(X)$.

We extend ϕ to a partial function $\Phi : \mathbb{N}^{\Lambda} \dashrightarrow \{0,1\}$ as follows. $\Phi(C)$ is undefined if either $C(X) = 0$ for all $X \in \Upsilon$, or if there exist $X_0, X_1 \in \Upsilon$ such that $C(X_0) > 0$, $C(X_1) > 0$, $\phi(X_0) = 0$ and $\phi(X_1) = 1$. Otherwise, there exists $b \in \{0,1\}$ such that $(\forall X \in \Upsilon)(C(X) > 0 \implies \phi(X) = b)$; in this case, the *output* $\Phi(C)$ of configuration C is b.

A configuration C is *output stable* if $\Phi(C)$ is defined and, for all C' such that $C \to^* C'$, $\Phi(C') = \Phi(C)$. We say that a CRD $\mathcal{D}$ is *stable* if, for any valid initial configuration $I \in \mathbb{N}^{\Lambda}$, there exists $b \in \{0,1\}$ such that *every* fair execution $\mathcal{E} = (I, C_1, C_2, \ldots)$ contains an output stable configuration C with $\Phi(C) = b$ (i.e., $\mathcal{D}$ always converges to a defined output on input I, and this output is the same for any fair execution starting from I). If $\mathcal{D}$ is stable, then some unique subset $S_0 \subseteq \mathbb{N}^{\Sigma}$ of all possible initial configurations always converges to output 0 and stays with that output, and the remainder $S_1 = \mathbb{N}^{\Sigma} \setminus S_0$ always converges to output 1 and stays with that output. We say that $\mathcal{D}$ *stably decides* the set S_1, or that $\mathcal{D}$ *stably decides* the predicate $\psi : \mathbb{N}^{\Sigma} \to \{0,1\}$ defined by $\psi(I_0) = 1$ if $I_0 \in S_1$ and $\psi(I_0) = 0$ if $I_0 \in S_0$.

The following theorem is due to Angluin, Aspenes, and Eisenstat [2]:

Theorem 2.1 ([2]). *A set $A \subseteq \mathbb{N}^k$ is stably decidable by a CRD if and only if it is semilinear.*

The definitions of [2] assume that $\Upsilon = \Lambda$ (i.e., every species votes). However, it is not hard to show that we may assume there are only two voting species, L^0 and L^1, so that $\#L^0 > 0$ and $\#L^1 = 0$ means that the CRD is answering "no", and $\#L^0 = 0$ and $\#L^1 > 0$ means that the CRD is answering "yes." This convention will be more convenient in this paper.

2.3 Stable Computation of Functions

We now define a notion of stable computation of *functions* similar to those above for predicates.[4] Intuitively, the inputs to the function are the initial counts of inputs species $X_1, \ldots, X_k$, and the outputs are the counts of "output" species $Y_1, \ldots, Y_l$, such that the CRN is guaranteed to eventually reach a configuration in which the counts of the output species have the correct values and never change from that point on.

[4] The extension from Boolean predicates to functions described by Aspnes and Ruppert [4] applies only to finite-range functions, where one can choose $|\Lambda| \geq |Y|$ for output range Y.

We now formally define what it means for a CRN to stably compute a function. Let $k, l \in \mathbb{Z}^+$. A *chemical reaction computer (CRC)* is a tuple $\mathcal{C} = (\Lambda, R, \Sigma, \Gamma, \sigma)$, where (Λ, R) is a CRN, $\Sigma \subset \Lambda$ is the *set of input species*, $\Gamma \subset \Lambda$ is the *set of output species*, such that $\Sigma \cap \Gamma = \varnothing$, $|\Sigma| = k$, $|\Gamma| = l$, and $\sigma \in \mathbb{N}^{\Lambda \setminus \Sigma}$ is the *initial context*. Write $\Sigma = \{X_1, X_2, \ldots, X_k\}$ and $\Gamma = \{Y_1, Y_2, \ldots, Y_l\}$. We say that a configuration C is *output count stable* if, for every C' such that $C \to^* C'$ and every $Y_i \in \Gamma$, $C(Y_i) = C'(Y_i)$ (i.e., the counts of species in Γ will never change if C is reached). As with CRD's, we require initial configurations I of $\mathcal{C}$ with input $I_0 \in \mathbb{N}^{\Sigma}$ to obey $I(X) = I_0(X)$ if $X \in \Sigma$ and $I(X) = \sigma(X)$ otherwise, calling them *valid initial configurations*. We say that $\mathcal{C}$ *stably computes* $f : \mathbb{N}^k \to \mathbb{N}^l$ if, for every valid initial configuration $I \in \mathbb{N}^{\Lambda}$, every fair execution $\mathcal{E} = (I, C_1, C_2, \ldots)$ contains an output count stable configuration C such that $f(I(X_1), I(X_2), \ldots, I(X_k)) = (C(Y_1), C(Y_2), \ldots, C(Y_l))$. In other words, the counts of species in Γ are guaranteed to converge to the value of $f(x_1, x_2, \ldots, x_k)$ when started in an initial configuration with x_i copies of X_i for each $i \in \{1, \ldots, k\}$. We say that such a CRC is *count stable*. For any species $A \in \Lambda$, we write $\#_\infty A$ to denote the eventual convergent count of A if $\#A$ is guaranteed to stabilize; otherwise, $\#_\infty A$ is undefined.

As an example of a formally defined CRC consider the function $f(x) = \lfloor x/2 \rfloor$ shown in Fig. 1(a). This function is deterministically computed by the CRC $(\Lambda, R, \Sigma, \Gamma, \sigma)$ where (Λ, R) is the CRN consisting of a single reaction $2X \to Y$, $\Sigma = \{X\}$ is the set of inputs species, $\Gamma = \{Y\}$ is the set of output species, and the initial context σ is zero for all species in $\Lambda \setminus \Sigma$. In Fig. 1(b) the initial context $\sigma(N) = 1$, and is zero for all other species in in $\Lambda \setminus \Sigma$.

In sections 3 and 4 we will describe systematic (but much more complex) constructions for these and all functions with semilinear graphs.

2.4 Kinetic Model

In this paper, the rate constants of all reactions are 1, and we define the kinetic model with this assumption. A reaction is *unimolecular* if it has one reactant and *bimolecular* if it has two reactants. We use no higher-order reactions in this paper when using the kinetic model.

The kinetics of a CRN is described by a continuous-time Markov process as follows. Given a fixed volume v and current configuration C, the *propensity* of a unimolecular reaction $\alpha : X \to \ldots$ in configuration C is $\rho(C, \alpha) = \#_C X$. The propensity of a bimolecular reaction $\alpha : X + Y \to \ldots$, where $X \neq Y$, is $\rho(C, \alpha) = \frac{\#_C X \#_C Y}{v}$. The propensity of a bimolecular reaction $\alpha : X + X \to \ldots$ is $\rho(C, \alpha) = \frac{1}{2} \frac{\#_C X(\#_C X - 1)}{v}$. The propensity function determines the evolution of the system as follows. The time until the next reaction occurs is an exponential random variable with rate $\rho(C) = \sum_{\alpha \in R} \rho(C, \alpha)$ (note that $\rho(C) = 0$ if no reactions are applicable to C). The probability that next reaction will be a particular α_{next} is $\frac{\rho(C, \alpha_{\text{next}})}{\rho(C)}$.

The kinetic model is based on the physical assumption of well-mixedness valid in a dilute solution. Thus, we assume the *finite density constraint*, which stipulates that a volume required to execute a CRN must be proportional to the maximum molecular count obtained during execution [11]. In other words, the total concentration (molecular count per volume) is bounded. This realistically constrains the speed of the computation achievable by CRNs. Note, however, that it is problematic to define the kinetic model for CRNs in which the reachable configuration space is unbounded for some start configurations, because this means that arbitrarily large molecular counts are reachable.[5] We apply the kinetic model only to CRNs with configuration spaces that are bounded for each start configuration.

3 Deterministic Function Computation

In this section we use Theorem 2.1 to show that only "simple" functions can be stably computed by CRCs. This is done by showing how to reduce the computation of a function by a CRC to the decidability of its graph by a CRD, and vice versa. In this section we do not concern ourselves with kinetics. Thus the volume is left unspecified, and we consider the combinatorial-only condition of fairness on execution sequences.

We say a function is semilinear if its graph is semilinear. The next lemma shows that every function computable by a chemical reaction network is semilinear.

Lemma 3.1. *Every function stably computable by a CRC is semilinear.*

Proof. Let $\mathcal{C} = (\Lambda, R, \Sigma, \Gamma, \sigma)$ be the CRC that stably computes $f : \mathbb{N}^k \to \mathbb{N}^l$, with input species $\Sigma = \{X_1, \ldots, X_k\}$ and output species $\Gamma = \{Y_1, \ldots, Y_l\}$. Modify $\mathcal{C}$ to obtain the following CRD $\mathcal{D} = (\Lambda', R', \Sigma', \Upsilon', \phi', \sigma')$. Let $\mathcal{Y}^C = \{Y_1^C, \ldots, Y_l^C\}$, where each $Y_i^C \notin \Lambda$ are new species. Let $\mathcal{Y}^P = \{Y_1^P, \ldots, Y_l^P\}$, where each $Y_i^P \notin \Lambda$ are new species. Intuitively, $\#Y_i^P$ represents the number of Y_i's produced by $\mathcal{C}$ and $\#Y_i^C$ the number of Y_i's consumed by $\mathcal{C}$. The goal is for $\mathcal{D}$ to stably decide the predicate $f(\#_0 X_1, \ldots, \#_0 X_k) = (\#_0 Y_1^C, \ldots, \#_0 Y_l^C)$. In other words, the initial configuration of $\mathcal{D}$ will be the same as that of $\mathcal{C}$ except for some copies of Y_i^C, equal to the purported output of f to be tested by $\mathcal{D}$. Since every predicate stably decidable by a CRD is semilinear (Theorem 2.1), this will prove the lemma.

Let $\Lambda' = \Lambda \cup \mathcal{Y}^C \cup \mathcal{Y}^P \cup \{L^0, L^1\}$. Let $\Sigma' = \Sigma \cup \mathcal{Y}^C$. Let $\Upsilon' = \{L^0, L^1\}$, with $\phi(L^0) = 0$ and $\phi(L^1) = 1$. Let $\sigma'(S) = 0$ for all $S \in \Lambda' \setminus \Sigma'$. Modify R by adding reactions to obtain R' as follows. For each reaction α that consumes a net number n of Y_i molecules, append n products Y_i^C to α. For each reaction α that produces a net number n of Y_i molecules, append n products Y_i^P to α. For example, the reaction $A + 2B + Y_1 + 3Y_3 \to Z + 3Y_1 + 2Y_3$ becomes $A + 2B + Y_1 + 3Y_3 \to Z + 3Y_1 + 2Y_3 + 2Y_1^P + Y_3^C$. Since $\mathcal{C}$ is count-stable, eventually no reactions producing or consuming net copies of Y_i are possible, whence $\mathcal{D}$ as defined so far is count output stable with respect to Y_i^P and Y_i^C as well.

[5] One possibility is to have a "dynamically" growing volume as in [11].

Then add the following additional reactions to R', for each $i \in \{1, \ldots, l\}$,

$$Y_i^P + Y_i^C \to L^1 \tag{3.1}$$
$$Y_i^P + L^1 \to Y_i^P + L^0 \tag{3.2}$$
$$Y_i^C + L^1 \to Y_i^C + L^0 \tag{3.3}$$
$$L^0 + L^1 \to L^1 \tag{3.4}$$

In the following, we use $\#_\infty^\uparrow Y_i^P$ to denote the total number of Y_i^P ever produced and $\#_\infty^\uparrow Y_i^C$ to denote $\#_0 Y_i^C$ plus the total number of Y_i^C's ever produced. Note that, if and only if $f(\#_0 X_1, \ldots, \#_0 X_k) = (\#_0 Y_1^C, \ldots, \#_0 Y_l^C)$, then eventually, for each i, $\#Y_i^P$ and $\#Y_i^C$ stabilize to equal values in the absence of reaction (3.1); in other words, if and only if $\#_\infty^\uparrow Y_i^P = \#_\infty^\uparrow Y_i^C$.

Since Y_i^P and Y_i^C are possibly produced but not consumed by reactions other than (3.1), we may think of reaction (3.1) as if it does not occur until $\#Y_i^P$ and $\#Y_i^C$ have stabilized, even though reaction (3.1) may consume some copies of Y_i^P and Y_i^C before all eventual copies have been produced.

Reactions (3.1)-(3.4) ensure that if $\#_\infty^\uparrow Y_i^P = \#_\infty^\uparrow Y_i^C$ for all $i \in \{1, \ldots, l\}$, then $\#_\infty L^1 > 0$ and $\#_\infty L^0 = 0$, and if $\#_\infty^\uparrow Y_i^P \neq \#_\infty^\uparrow Y_i^C$ for some $i \in \{1, \ldots, l\}$, then $\#_\infty L^1 = 0$ and $\#_\infty L^0 > 0$. To show that this holds, we have two cases for each $i \in \{1, \ldots, l\}$. In the following, we write $f(\#X_1, \ldots, \#X_k)_i$ to denote the value $\#Y_i$ if $f(\#X_1, \ldots, \#X_k) = (\#Y_1, \ldots, \#Y_l)$.

1. $\underline{f(\#_0 X_1, \ldots, \#_0 X_k)_i = \#_0 Y_i^C \text{ for all } i \in \{1, \ldots, l\}}$: Then $\#_\infty^\uparrow Y_i^P = \#_\infty^\uparrow Y_i^C$ for all $i \in \{1, \ldots, l\}$, so eventually every Y_i^P and Y_i^C disappears through reaction (3.1). At this point there are some number of L^0's and L^1's remaining. The number of L^1's must be positive since the final execution of reaction (3.1) created a copy of L^1. Since none of reactions (3.1)-(3.3) are possible, $\#L^1$ stays positive forever. After this time, reaction (3.4) eventually removes all copies of L^0.
2. $\underline{f(\#_0 X_1, \ldots, \#_0 X_k)_i \neq \#_0 Y_i^C \text{ for some } i \in \{1, \ldots, l\}}$: Then $\#_\infty^\uparrow Y_i^P \neq \#_\infty^\uparrow Y_i^C$ for some $i \in \{1, \ldots, l\}$, so reaction (3.1) ensures that eventually either 1) $\#_\infty Y_i^C = 0$ and $\#_\infty Y_i^P > 0$, or 2) $\#_\infty Y_i^C > 0$ and $\#_\infty Y_i^P = 0$. Eventually reaction (3.1) is not possible for any $j \in \{1, \ldots, l\}$ because either $\#_\infty Y_j^P = 0$ or $\#_\infty Y_j^C = 0$, and at that point, no more copies of L^1 are produced. From then on, reaction (3.2) (in case (1)) or reaction (3.3) (in case (2)) ensures that eventually all copies of L^1 are converted to L^0. Reaction (3.4) may convert some copies of L^0 back to L^1 before this happens, but this strictly decreases the quantity $(\#L^1 + \#L^0)$. If this quantity reaches 1 then reaction (3.4) is no longer possible. Thus eventually all existing copies of L^1 are converted to L^0 and reaction (3.4) is no longer possible.

Thus, if $f(\#_0 X_1, \ldots, \#_0 X_k) = (\#_0 Y_1^C, \ldots, \#_0 Y_l^C)$, then $\#_\infty L^1 > 0$ and $\#_\infty L^0 = 0$, and otherwise, $\#_\infty L^1 = 0$ and $\#_\infty L^0 > 0$, showing that $\mathcal{D}$ stably decides the graph of f. □

The next lemma shows the converse of Lemma 3.1. Intuitively, it uses a random search of the output space to look for the correct answer to the function and uses a predicate decider to check whether the correct solution has been found.

Lemma 3.2. *Every semilinear function is stably computable by a CRC.*

Proof. Let $f : \mathbb{N}^k \to \mathbb{N}^l$ be a semilinear function, and let

$$F = \{\ (\mathbf{x}, \mathbf{y}) \in \mathbb{N}^k \times \mathbb{N}^l \ \mid\ f(\mathbf{x}) = \mathbf{y}\ \}$$

denote the graph of f. We then consider the set

$$\widehat{F} = \{\ (\mathbf{x}, \mathbf{y}^P, \mathbf{y}^C) \in \mathbb{N}^k \times \mathbb{N}^l \times \mathbb{N}^l \ \mid\ f(\mathbf{x}) = \mathbf{y}^P - \mathbf{y}^C\ \}.$$

Intuitively, $\widehat{F}$ defines the same function as F, but with each output variable expressed as the difference between two other variables. Note that $\widehat{F}$ is not the graph of a function since for each $\mathbf{y} \in \mathbb{N}^l$ there are an infinite number of pairs $(\mathbf{y}^P, \mathbf{y}^C)$ such that $\mathbf{y}^P - \mathbf{y}^C = \mathbf{y}$. However, we only care that $\widehat{F}$ is a semilinear set so long as F is a semilinear set.

Then by Theorem 2.1, $\widehat{F}$ is stably decidable by a CRD $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon, \phi, \sigma)$, where

$$\Sigma = \{X_1, \ldots, X_k, Y_1^P, \ldots, Y_l^P, Y_1^C, \ldots, Y_l^C\},$$

and we assume that Υ contains only species L^1 and L^0 such that for any output-stable configuration of $\mathcal{D}$, exactly one of $\#L^1$ or $\#L^0$ is positive to indicate a yes or no answer, respectively.

Define the CRC $\mathcal{C} = (\Lambda', R', \Sigma', \Gamma', \sigma')$ as follows. Let $\Sigma' = \{X_1, \ldots, X_k\}$. Let $\Gamma' = \{Y_1, \ldots, Y_l\}$. Let $\Lambda' = \Lambda \cup \Gamma'$. Let $\sigma'(S) = 0$ for all $S \in \Lambda \setminus (\Sigma \cup \{L^0\})$, and let $\sigma'(L^0) = 1$. Intuitively, we will have L^0 change the value of $\mathbf{y}$ (by producing either Y_i^P or Y_i^C molecules), since L^0's presence indicates that $\mathcal{D}$ has not yet decided that the predicate is satisfied. It essentially searches for new values of $\mathbf{y}$ that do satisfy the predicate. This indirect way of representing the value $\mathbf{y}$ is useful because $\mathbf{y}^P$ and $\mathbf{y}^C$ can both be increased monotonically to change $\mathbf{y}$ in either direction. If we wanted to test a lower value of $\mathbf{y}_i$, then this would require consuming a copy of Y_i, but this may not be possible if $\mathcal{D}$ has already consumed all of them.

Let R' be R plus the following reactions for each $1 \le i \le l$:

$$L^0 \to L^0 + Y_i^P + Y_i \tag{3.5}$$

$$L^0 + Y_i \to L^0 + Y_i^C \tag{3.6}$$

It is clear that reactions (3.5) and (3.6) enforce that at any time, $\#Y_i$ is equal to the total number of Y_i^P's produced by reaction (3.5) minus the total number of Y_i^C's produced by reaction (3.6) (although some of each of Y_i^P or Y_i^C may have been produced or consumed by other reactions in R).

Suppose that $f(\mathbf{x}) \neq (\#Y_1, \ldots, \#Y_l)$. Then if there are no L^0 molecules present, the counts of Y_i^P and Y_i^C are not changed by reactions (3.5) and (3.6). Therefore only reactions in R proceed, and by the correctness of $\mathcal{D}$, eventually an L^0 molecule is produced (since eventually $\mathcal{D}$ must reach an output-stable configuration answering "no", although L^0 may appear before $\mathcal{D}$ reaches an output-stable configuration, if some L^1 are still present). Once L^0 is present, by the fairness condition (choosing $\Delta = \{Y_1, \ldots .Y_l\}$), eventually the value of $(\#Y_1, \ldots, \#Y_l)$ will change by reaction (3.5) or (3.6). In fact, *every* value of $(\#Y_1, \ldots, \#Y_l)$ is possible to explore by the fairness condition.

Suppose then that $f(\mathbf{x}) = (\#Y_1, \ldots, \#Y_l)$. Perhaps L^0 is present because the reactions in R have not yet reached an output-stable "yes" configuration. Then perhaps the value of $(\#Y_1, \ldots, \#Y_l)$ will change so that $f(\mathbf{x}) \neq (\#Y_1, \ldots, \#Y_l)$. But by the fairness condition, a correct value of $(\#Y_1, \ldots, \#Y_l)$ must be present infinitely many times, so again by the fairness condition, since from such a configuration it is possible to eliminate all L^0 molecules before producing Y_i^P or Y_i^C molecules, this must eventually happen. When all L^0 molecules are gone while $f(\mathbf{x}) = (\#Y_1, \ldots, \#Y_l)$, it is no longer possible to change the value of $(\#Y_1, \ldots, \#Y_l)$, whence $\mathcal{C}$ has reached a count-stable configuration with the correct answer. Therefore $\mathcal{C}$ stably computes f. □

Lemmas 3.1 and 3.2 immediately imply the following theorem.

Theorem 3.3. *A function $f : \mathbb{N}^k \to \mathbb{N}^l$ is stably computable by a CRC if and only if it is semilinear.*

One unsatisfactory aspect of Lemma 3.2 is that we "peek inside the black box" of $\mathcal{D}$ by using the fact that we know it is deciding a semilinear predicate. Lemma 3.1, on the other hand, uses only the fact that $\mathcal{C}$ is computing some function. Although we know that $\mathcal{C}$, being a chemical reaction computer, is only capable of computing semilinear functions, if we imagine that some external powerful "oracle" controlled the reactions of $\mathcal{C}$ to allow it to stably compute a non-semilinear function, then $\mathcal{D}$ would decide that function's graph. Thus Lemma 3.1 is more like the black-box oracle Turing machine reductions employed in computability and complexity theory, which work no matter what mythical device is hypothesized to be responsible for answering the oracle queries.

4 Speed of Deterministic Function Computation

Lemma 3.2 describes how a CRN can deterministically compute any semilinear function. However, there are problems with this construction if we attempt to use it to evaluate the speed of semilinear function computation in the kinetic model. First, the configuration space is unbounded for any input since the construction searches over outputs without setting bounds. Thus, more care must be taken to ensure that any infinite execution sequence will be fair with probability 1 in the kinetic model. What is more, since the maximum molecular count is unbounded, it is not clear how to set the volume for the time analysis. Even if we attempt to properly define kinetics, it seems like any reasonable time analysis of the random search process will result in expected time at least exponential in the size of the output.[6]

Angluin, Aspnes, and Eisenstat combined the results of [2] with a fast, error-prone simulation of a bounded-space Turing machine to show that semilinear predicates can be computed without error in expected polylogarithmic time [1]. In this section we show that a similar technique implies that semilinear functions can be computed by CRNs without error in expected polylogarithmic time in the kinetic model.

For our asymptotic time analysis, let $n = \|\mathbf{x}\|$. In this section, the total molecular count attainable will always be $O(n)$; thus, for by finite density constraint, the volume $v = O(n)$.

Theorem 4.1. *Let $f : \mathbb{N}^k \to \mathbb{N}^l$ be semilinear. Then there is a CRC $\mathcal{C}$ that stably computes f, and the expected time for $\mathcal{C}$ to reach a count-stable configuration on input $\mathbf{x}$ is $O(\text{polylog } \|\mathbf{x}\|)$.*

Proof. (proof sketch) Our CRC will use the counts of Y_j for each output dimension $\mathbf{y}_j$ as the global output, and begins by running in parallel[7]:

1. A fast, error-prone CRC $\mathcal{F}$ for $\mathbf{y}, \mathbf{b}, \mathbf{c} = f(\mathbf{x})$. It is constructed based on [1]. By [1], for any constant $c > 0$, we may design $\mathcal{F}$ so that it is correct and finishes in time $O(\log^5 n)$ with probability at least $1 - n^{-c}$. We modify it so that upon halting, it copies an "internal" output species $\widehat{Y}_j$ to Y_j (the global output), B_j, and C_j through reactions $H + \widehat{Y}_j \to Y_j + B_j + C_j$ (in asymptotically negligible time). Here, H is some molecule that is guaranteed with high probability not to be present until $\mathcal{F}$ has halted, and to be present in large ($\Omega(n)$) count so that the conversion is fast. In this way we are guaranteed that the amount of Y_j produced by $\mathcal{C}$ is the same as the amounts of B_j and C_j no matter whether its computation is correct or not.
2. A slow, deterministic CRC $\mathcal{S}$ for $\mathbf{y}' = f(\mathbf{x})$. It is constructed as in Lemma 4.2, running in expected $O(n \log n)$ time.
3. A slow, deterministic CRD $\mathcal{D}$ for the semilinear predicate "$\mathbf{b} = f(\mathbf{x})$?". It is constructed as in [3] and runs in expected $O(n \log n)$ time.

Following Angluin, Aspnes, and Eisenstat [1], we construct a "timed trigger" as follows, using a leader molecule, a marker molecule, and $n = \|\mathbf{x}\|$ interfering molecules. (These interfering molecules can simply be the input species and some of their "descendants" such that their count is held constant.) The leader fires the trigger if it encounters the marker molecule d times without any intervening

[6] The random walk is biased downward because of the increasing propensities of the reactions consuming Y_i's.

[7] Throughout this section, we use the technique of "running multiple CRNs in parallel" on the same input. To accomplish this it is necessary to split the inputs $X_1, \ldots, X_k$ into separate molecules using a reaction $X_i \to X_i^1 + X_i^2 + \ldots + X_i^p$, which will add only $O(\log n)$ to the time complexity, so that each of the p separate parallel CRNs do not interfere with one another. For brevity we omit stating this formally when the technique is used.

reactions with the interfering molecules. This happens rarely enough that with high probability the trigger fires after $\mathcal{F}$ and $\mathcal{D}$ finishes (time analysis is presented below). When the trigger fires, it checks if $\mathcal{D}$ is outputting a "no" (e.g. has a molecule of L_0), and if so, produces a molecule of P_{fix}. This indicates that the output of the fast CRC $\mathcal{F}$ is not to be trusted, and the system should switch from the possible erroneous result of $\mathcal{F}$ to the sure-to-be correct result of $\mathcal{S}$.

Once a P_{fix} is produced, the system converts the output molecules Y_j' of the slow, deterministic CRC $\mathcal{S}$ to the global output Y_j, and kills enough of the global output molecules to remove the ones produced by the fast, error-prone CRC:

$$P_{\text{fix}} + Y_j' \to P_{\text{fix}} + Y_j \tag{4.1}$$

$$P_{\text{fix}} + C_j \to P_{\text{fix}} + \overline{Y}_j \tag{4.2}$$

$$Y_j + \overline{Y}_j \to \varnothing. \tag{4.3}$$

Finally, P_{fix} triggers a process consuming "essential components" of $\mathcal{F}$ in expected $O(\log n)$ time so that afterward, $\mathcal{F}$ cannot produce any output molecules. While this step is not required for correctness, it is necessary for the time analysis in order to ensure that $\mathcal{F}$ does not take too long to output (if $\mathcal{F}$ fails it could produce its output even after $\mathcal{S}$).

First, observe that the output will always eventually converge to the right answer, no matter what happens: If P_{fix} is eventually produced, then the output will eventually be exactly that given by $\mathcal{S}$ which is guaranteed to converge correctly. If P_{fix} is never produced, then the fast, error-prone CRC must produce the correct amount of Y_j — otherwise, $\mathcal{D}$ will detect a problem.

For the expected time analysis, let us first analyze the trigger. The probability that the trigger leader will fire on any particular reaction number is at most n^{-d}. In time n^2, the expected number of leader reactions is $O(n^2)$. Thus, the expected number of firings of the trigger in n^2 time is n^{-d+2}. This implies that the probability that the trigger fires before n^2 time is at most n^{-d+2}. The expected time for the trigger to fire is $O(n^d)$.

We now consider the contribution to the total expected time from 3 cases:

1. $\mathcal{F}$ is correct, and the trigger fires after time n^2. There are two subcases: (a) $\mathcal{F}$ finishes before the trigger fires. Conditional on this, the whole system converges to the correct answer, never to change it again, in expected time $O(\log^5 n)$. This subcase contributes at most $O(\log^5 n)$ to the total expected time. (b) $\mathcal{F}$ finishes after the trigger fires. In this case, we may produce a P_{fix} molecule and have to rely on the slow CRC $\mathcal{S}$. The probability of this case happening is at most n^{-c}. Conditional on this case, the expected time for the trigger to fire is still $O(n^d)$. The whole system converges to the correct answer in expected time $O(n^d)$, because everything else is asymptotically negligible. Thus the contribution of this subcase to the total expectation is at most $O(n^{-c} \cdot n^d) = O(n^{-c+d})$.
2. $\mathcal{F}$ is correct, but the trigger fires before n^2 time. In this case, we may produce a P_{fix} molecule and have to rely on the slow CRC $\mathcal{S}$ for the output. The probability of this case occurring is at most n^{-d+2}. Conditional on this case

occurring, the expected time for the whole system to converge to the correct answer can be bounded by $O(n^2)$. Thus the contribution of this subcase to the total expectation is at most $O(n^{-d+2} \cdot n^2) = O(n^{-d+4})$.

3. $\mathcal{F}$ fails. In this case we'll have to rely on the slow CRC $\mathcal{S}$ for the output again. Since this occurs with probability at most n^{-c}, and the conditional expected time for the whole system to converge to the correct answer can be bounded by $O(n^d)$ again, the contribution of this subcase to the total expectation is at most $O(n^{-c} \cdot n^d) = O(n^{-c+d})$.

So the total expected time is bounded by $O(\log^5 n) + O(n^{-c+d}) + O(n^{-d+4}) + O(n^{-c+d}) = O(\log^5 n)$ for $d > 4, c > d$. □

Lemma 4.2. *Let $f : \mathbb{N}^k \to \mathbb{N}^l$ be semilinear. Then there is a CRC $\mathcal{C}$ that stably computes f, and the expected time for $\mathcal{C}$ to reach a count-stable configuration on input $\mathbf{x}$ is $O(\|\mathbf{x}\| \log \|\mathbf{x}\|)$ (where the $O()$ constant depends on f).*

Proof. By Lemma 3.2, there is a CRC $\mathcal{C}_s$ that stably computes f. However, that CRC is too slow to use in this proof. We provide an alternative proof that every semilinear function can be computed by a CRC in expected time $O(\|\mathbf{x}\| \log \|\mathbf{x}\|)$. Rather than relying on a random search of the output space as in Lemma 3.2, it computes the function more directly.

By Lemma 4.3, there is a finite set $F = \{f_1 : \mathbb{N}^k \dashrightarrow \mathbb{N}^l, \ldots, f_m : \mathbb{N}^k \dashrightarrow \mathbb{N}^l\}$ of affine partial functions, where each dom f_i is a linear set, such that, for each $\mathbf{x} \in \mathbb{N}^k$, if $f_i(\mathbf{x})$ is defined, then $f(\mathbf{x}) = f_i(\mathbf{x})$. We compute f on input $\mathbf{x}$ as follows. Since each dom f_i is a linear (and therefore semilinear) set, we compute each predicate $\phi_i =$ "$\mathbf{x} \in$ dom f_i and $(\forall 1 \leq i' < i)$ $\mathbf{x} \notin$ dom $f_{i'}$?" by separate parallel CRD's. (The latter condition ensures that for each $\mathbf{x}$, precisely one of the predicates is true.)

By Lemma 4.5, we can compute each f_i by parallel CRC's. Assume that for each $1 \leq i \leq m$ and each $1 \leq j \leq l$, the jth output of the ith function is represented by species $\widehat{Y}^i_j$. Each $\widehat{Y}^i_j$ is an "inactive" version of "active" output species Y^i_j.

For each $1 \leq i \leq m$, we assume that the CRD computing the predicate ϕ_i represents its output by voting species L^i_1 to represent "yes" and L^i_0 to represent "no". Then add the following reactions for each $1 \leq i \leq m$ and each $1 \leq j \leq l$:

$$\begin{aligned} L^i_1 + \widehat{Y}^i_j &\to L^i_1 + Y^i_j + Y_j \\ L^i_0 + Y^i_j &\to M^i_j \\ M^i_j + Y_j &\to L^i_0 + \widehat{Y}^i_j. \end{aligned}$$

That is, a "yes" answer for function i activates the ith output and a "no" answer deactivates the ith output. Eventually each CRD stabilizes so that precisely one i has L^i_1 present, and for all $i' \neq i$, $L^{i'}_0$ is present. At this point, all outputs for the correct function f_i are activated and all other outputs are deactivated. Since eventually the count of Y^i_j stabilizes to 0 for all but one value of i, this ensures that $\#Y_j$ stabilizes to the correct value of output $\mathbf{y}_j$.

It remains to analyze the expected time to stabilization. Let $n = \|\mathbf{x}\|$. By Lemma 4.5, the expected time for each affine function computation to complete is $O(n \log n)$. Since the $\widehat{Y}_i^j$ are produced monotonically, the most Y_i^j molecules that are ever produced is $\#_\infty \widehat{Y}_i^j$. Since we have m computations in parallel, the expected time for all of them to complete is $O((n \log n)m) = O(n \log n)$ (since m depends on f but not n). We must also wait for each predicate computation to complete. By Theorem 5 of [2], each of these predicates takes expected time $O(n)$ to complete, so all of them complete in expected time $O(nm) = O(n)$.

At this point, the L_1^i leaders must convert inactive output species to active, and $L_0^{i'}$ (for $i' \neq i$) must convert active output species to inactive. A similar analysis to the proof of Lemma 4.5 shows that each of these requires at most $O(n \log n)$ expected time, therefore they all complete in expected time $O((n \log n)m) = O(n \log n)$. □

Lemma 4.3. *Let $f : \mathbb{N}^k \to \mathbb{N}^l$ be a semilinear function. Then there is a finite set $\{f_1 : \mathbb{N}^k \dashrightarrow \mathbb{N}^l, \ldots, f_m : \mathbb{N}^k \dashrightarrow \mathbb{N}^l\}$ of affine partial functions, where each* dom *f_i is a linear set, such that, for each $\mathbf{x} \in \mathbb{N}^k$, if $f_i(\mathbf{x})$ is defined, then $f(\mathbf{x}) = f_i(\mathbf{x})$.*

Proof. Let $F = \{\ (\mathbf{x}, \mathbf{y}) \in \mathbb{N}^k \times \mathbb{N}^l \mid f(\mathbf{x}) = \mathbf{y}\ \}$ be the graph of f. Since F is semilinear, it is a finite union of linear sets $\{L_1, \ldots, L_n\}$. It suffices to show that each of these linear sets L_m is the graph of an affine partial function. Let L'_m be the $(k+1)$-dimensional projection of L_m onto the coordinates defined by $\mathbf{x}$ and $\mathbf{y}_i$, which is linear because L_m is. Since L'_m is linear, there exist vectors $\mathbf{b}, \mathbf{u}^1, \ldots, \mathbf{u}^p \in \mathbb{N}^{k+1}$ such that $L'_m = \{\ \mathbf{b} + n_1\mathbf{u}^1 + \ldots + n_p\mathbf{u}^p \mid n_1, \ldots, n_p \in \mathbb{N}\ \}$.

It suffices to show that L'_m is a subset of a k-dimensional hyperplane. This is true if at most k of the $\mathbf{u}^1, \ldots, \mathbf{u}^p$ are linearly independent. Suppose not; then there are $k+1$ linearly independent vectors among the list. Assume without loss of generality that they are $\mathbf{u}^1, \ldots, \mathbf{u}^{k+1}$. For each $1 \leq i \leq k+1$, let $\mathbf{v}^i$ be $\mathbf{u}^i$ projected onto the first k coordinates. Since there are $k+1$ vectors and they are k-dimensional, $\mathbf{v}^1, \ldots, \mathbf{v}^{k+1}$ must be linearly dependent. By Lemma 4.4, there exist two lists of natural numbers $N = (n_1, \ldots, n_{k+1})$ and $M = (m_1, \ldots, m_{k+1})$ such that $N \neq M$ and $\sum_{i=1}^{k+1} n_i\mathbf{v}^i = \sum_{i=1}^{k+1} m_i\mathbf{v}^i$. Then the points

$$\mathbf{z}^1 = \mathbf{b} + \sum_{i=1}^{k+1} n_i\mathbf{u}^i \text{ and } \mathbf{z}^2 = \mathbf{b} + \sum_{i=1}^{k+1} m_i\mathbf{u}^i$$

are in L'_m. They must have different y-coordinates, or else we would have $\mathbf{z}^1 = \mathbf{z}^2$ (since their first k coordinates agree), which would contradict the linear independence of $\mathbf{u}^1, \ldots, \mathbf{u}^{k+1}$. Therefore L'_m does not define the graph of a function since these two identical inputs map to two different outputs, a contradiction. □

Lemma 4.4. *Let $\mathbf{v}^1, \ldots, \mathbf{v}^t \in \mathbb{N}^k$ be linearly dependent vectors. Then there are two lists of natural numbers $N = (n_1, \ldots, n_t) \in \mathbb{N}^t$ and $M = (m_1, \ldots, m_t) \in \mathbb{N}^t$ such that $N \neq M$ and $\sum_{i=1}^{t} n_i\mathbf{v}^i = \sum_{i=1}^{t} m_i\mathbf{v}^i$.*

Proof. By the definition of linear dependence, there exist two lists of real numbers $N' = (n'_1, \ldots, n'_t) \in \mathbb{R}^t$ and $M' = (m'_1, \ldots, m'_t) \in \mathbb{R}^t$ such that $N' \neq M'$ and $\sum_{i=1}^t n'_i \mathbf{v}^i = \sum_{i=1}^t m'_i \mathbf{v}^i \in \mathbb{N}^k$ (since all points, integer or not, in the basis of $\mathbf{v}^1, \ldots, \mathbf{v}^t \in \mathbb{N}^k$ can be so expressed). Perhaps some of the coefficients are negative; however, by increasing both n'_i and m'_i by $\min\{n'_i, m'_i\}$ (which changes each sum by the same amount, keeping them equal), we may assume that all coefficients are nonnegative. Furthermore, since the sum $\sum_{i=1}^t n_i \mathbf{v}^i$ is integer-valued, $N', M' \in \mathbb{Q}^t$.

Let L be the least common multiple of the denominators of each n'_i and m'_i when expressed in lowest terms. By multiplying each coefficient by L, we obtain nonnegative integers $N = (n_1, \ldots, n_t) \in \mathbb{N}^t$ and $M = (m_1, \ldots, m_t) \in \mathbb{N}^t$ such that $N \neq M$ and $\sum_{i=1}^t n_i \mathbf{v}^i = \sum_{i=1}^t m_i \mathbf{v}^i$. □

Lemma 4.5. *Let $f : \mathbb{N}^k \dashrightarrow \mathbb{N}^l$ be a partial affine function. Then there is a CRC that computes f on input $\mathbf{x}$ in expected time $O(\|\mathbf{x}\| \log \|\mathbf{x}\|)$, such that the output molecules monotonically increase with time (i.e. none are ever consumed).*

Proof. If $\mathbf{y} = f(\mathbf{x})$, then there exist $kl + l + k$ integers $a_1^1, \ldots, a_k^l \in \mathbb{Q} \cap [0, \infty)$ and $b_1, \ldots, b_l, c_1, \ldots, c_k \in \mathbb{Z}$ such that each $\mathbf{y}_j = b_j + \sum_{i=1}^k a_i^j (\mathbf{x}_i + c_i)$. Define the CRC as follows. It has input species $\Sigma = \{X_1, \ldots, X_k\}$ and output species $\Gamma = \{Y_1, \ldots, Y_l\}$.

Let $b_1^j, \ldots, b_k^j \in \{-|b_j|, \ldots, |b_j|\}$ be integers such that $\sum_{i=1}^k b_i^j = b_j$. Let $c_i^1, \ldots, c_i^l \in \{-|c_i|, \ldots, |c_i|\}$ be integers such that $\sum_{j=1}^l c_i^j = c_i$. For each $1 \leq i \leq k$ and $1 \leq j \leq l$, start with a leader molecule $\widehat{L}^j_{i, c_i^j}$. For each $1 \leq i \leq k$ and $1 \leq j \leq l$, let $\frac{n_i^j}{d_i^j}$ be a_i^j expressed as a fraction such that $d_i^j > c_i$ and, if $b_j < 0$, $n_i^j \geq -b_j$. For each $1 \leq i \leq k$, add the reaction

$$X_i \to X_i^1 + X_i^2 + \ldots + X_i^l$$

For each $1 \leq i \leq k$, $1 \leq j \leq l$ and m such that $\min\{0, c_i^j\} \leq m \leq d_i^j$, add the reactions

$$\widehat{L}^j_{i,m} + X_i^j \to \begin{cases} \widehat{L}^j_{i,m+1}, & \text{if } m < d_i^j; \\ L^j_{i,0} + (n_i^j + b_i^j) Y_j, & \text{otherwise.} \end{cases}$$

$$L^j_{i,m} + X_i^j \to \begin{cases} L^j_{i,m+1}, & \text{if } m < d_i^j; \\ L^j_{i,0} + n_i^j Y_j, & \text{otherwise.} \end{cases}$$

Each initial leader $\widehat{L}$ starts counting (m is the "current count") at an initial value either above or below 0 (depending on the sign of c_i) to account for the initial offset c_i of X_i. Also, each initial leader releases a different amount of Y_j (again depending on the sign of b_j) to account for the initial offset b_j of Y_j. After counting to d_i^j, each initial leader $\widehat{L}$ converts to a normal leader L, which releases n_i^j Y_j molecules for every d_i^j X_i molecules encountered. Therefore (after accounting for initial offsets) each X_i molecule converts into a number of Y_j

molecules based on the weighted sum of the a_i^j coefficients. Therefore when all X_i^j molecules are consumed, the number of Y_j molecules is the proper value $\mathbf{y}_j = b_j + \sum_{i=1}^k a_i^j(\mathbf{x}_i + c_i)$.

It remains to analyze the expected completion time. Let $n = \|\mathbf{x}\|$. Since the total number of molecules in solution at any time is $O(n)$, the volume required is also $O(n)$. We measure the time to consume all X_i molecules for all i. We start with n such molecules, so the time for all of them to convert is the maximum of n exponential random variables, each with constant expected value, which is $O(\log n)$.

Then all L_i^j molecules must encounter every X_i^j molecule. By a coupon collector argument, this requires at most $O(n \log n)$ time. Therefore the CRC stabilizes in expected time $O(n \log n)$. □

Acknowledgements. We thank Damien Woods and Niranjan Srinivas for many useful discussions, and anonymous reviewers for helpful suggestions.

References

1. Angluin, D., Aspnes, J., Eisenstat, D.: Fast Computation by Population Protocols with a Leader. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 61–75. Springer, Heidelberg (2006)
2. Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: PODC, pp. 292–299 (2006)
3. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20(4), 279–304 (2007)
4. Aspnes, J., Ruppert, E.: An introduction to population protocols. Bulletin of the European Association for Theoretical Computer Science 93, 98–117 (2007)
5. Cardelli, L.: Strand algebras for DNA computing. Natural Computing 10(1), 407–428 (2011)
6. Condon, A., Hu, A., Maňuch, J., Thachuk, C.: Less Haste, Less Waste: On Recycling and Its Limits in Strand Displacement Systems. Journal of the Royal Society Interface (to appear, 2012); In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 84–99. Springer, Heidelberg (2011)
7. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)
8. Jiang, H., Riedel, M., Parhi, K.: Digital signal processing with molecular reactions. IEEE Design and Test of Computers (to appear, 2012)
9. Magnasco, M.O.: Chemical kinetics is Turing universal. Physical Review Letters 78(6), 1190–1193 (1997)
10. Presburger, M.: Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen. In: Welchem die Addition als einzige Operation hervortritt. Compte Rendus du I, Warsaw. Congrks des Mathematiciens des pays Slavs, pp. 92–101 (1930)
11. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing 7(4), 615–633 (2008)
12. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proceedings of the National Academy of Sciences 107(12), 5393 (2010)
13. Zavattaro, G., Cardelli, L.: Termination Problems in Chemical Kinetics. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 477–491. Springer, Heidelberg (2008)

Reachability Bounds for Chemical Reaction Networks and Strand Displacement Systems

Anne Condon[1], Bonnie Kirkpatrick[1], and Ján Maňuch[1,2]

[1] Department of Computer Science, University of British Columbia, Vancouver, British Columbia

[2] Department of Mathematics, Simon Fraser University, Burnaby, British Columbia

Abstract. Chemical reaction networks (CRNs) and DNA strand displacement systems (DSDs) are widely-studied and useful models of molecular programming. However, in order for some DSDs in the literature to behave in an expected manner, the initial number of copies of some reagents is required to be fixed. In this paper we show that, when multiple copies of all initial molecules are present, general types of CRNs and DSDs fail to work correctly if the length of the shortest sequence of reactions needed to produce any given molecule exceeds a threshold that grows polynomially with attributes of the system.

1 Introduction

DNA strand displacement systems (DSDs) [10,13] and chemical reaction networks (CRNs) [8,7] are important molecular programming models. DSDs provide sophisticated molecular realizations of logic circuits and even artificial neurons [4,5], while CRNs elegantly express chemical programs that can then be translated into DSDs [8,9]. CRNs and thus DSDs can in principle simulate Turing-general models of computation [6,3], and DSDs can be energy efficient [11,6,9,14]. It is also possible in principle to recycle molecules in DSDs by running reversible reactions or displacements in both forwards and reverse directions, so that t steps of the system use just $O(\log t)$ molecules [1].

However, correct behavior of some published DSDs [3,1] requires that an exact numbers of some reactants are present initially, and it is currently impractical to obtain the exact numbers in a wet lab. We previously considered the conditions for a class of CRNs to work correctly when multiple copies of all initial molecules are present and showed that the length of the shortest trace (sequence of reactions) needed to "reach", i.e., produce, any given molecule is bounded by a polynomial function of some attributes of a CRN in this class [1]. This reachability upper bound reveals important limits of molecular programs that fall in the class covered by our result: we cannot write such programs that run correctly in a closed chemical system and for which the number of steps (reactions) of the program is sufficiently large relative to the volume of initial reagents.[1]

[1] The volume is the physical volume of all the molecules. It can be approximated by the number of all the types of reagents in the initial configurations.

In this work we provide two new reachability upper bounds that significantly extend our earlier work. The first new theorem applies to *tagged* CRNs which, as we explain below, are important because they can be translated into DSDs of comparable volume that can simulate the CRNs traces. The second new theorem applies to a broader class of DSDs than does the translated version of our first result. In the rest of this introduction we motivate our results in more detail. Sections 2 and 3 provide technical details of both theorems; additional details are in the full version of the paper. We list some open questions in Section 4.

New result for chemical reaction networks (CRNs). Figure 1 illustrates a CRN of the type to which our new result applies (a formal definition of CRN is in Section 2). Each reaction r is reversible, and has unique *tag* species τ_r^+ and τ_r^- on its left and right sides respectively. We explain later why we focus on tagged CRNs, and also explain why we ignore reaction rate constants in our example and results. When a single copy of each species in the set $\{A, B, E, \tau_1^+, \tau_2^+, \tau_3^+, \tau_4^+\}$ is initially present, it takes six reaction steps to produce the product H, and to do so, reaction 1 must run in the forwards direction, then later run backwards, then forwards again. However, if another copy of A and B are present initially then H can be generated with just four reactions. The behavior of the system with two copies does not mirror its behavior with one copy; in this sense it is incorrect. While for this simple example it might not seem important how many steps are needed to produce a particular product, it is critically important in contexts where the product is the result of a computation and an erroneous result could be produced as a result of cross-talk, or short-circuiting of multiple copies of the intended computation.

$$\begin{array}{ll}
(1)\ \tau_1^+ + A + B & \rightleftharpoons C + D + \tau_1^- \\
(2)\ \tau_2^+ + C + D + E & \rightleftharpoons C + D + F + \tau_2^- \\
(3)\ \tau_3^+ + A + B + F & \rightleftharpoons A + B + G + \tau_3^- \\
(4)\ \tau_4^+ + C + D + G & \rightleftharpoons C + D + H + \tau_4^-
\end{array}$$

Fig. 1. Example of a simple tagged chemical reaction network (CRN)

In this paper, our notion of correctness is that of *copy tolerance* [1]. We say that a CRN $\mathbf{C}$ is x-copy-tolerant if the length of the shortest trace that produces any species s in $\mathbf{C}$ and in $\mathbf{C}^{(x)}$ is the same, where $\mathbf{C}^{(x)}$ is the CRN with the same reactions as $\mathbf{C}$ but with x initial copies of each initial molecule of $\mathbf{C}$. A system is copy-tolerant if it is x-copy-tolerant for all x. The CRN of Figure 1 is not 2-copy-tolerant. Copy-tolerance is a weak notion of correctness; if a CRN $\mathbf{C}$ is not 2-copy tolerant then, for example, $\mathbf{C}$ also fails to satisfy the stronger requirement that each possible trace of $\mathbf{C}$ in the 2-copy setting is an interleaving of two possible traces in the single copy setting. We chose to work with a weak notion of correctness because it makes our results stronger, i.e., they apply also to notions of correctness that are stronger than copy-tolerance.

Our first reachability upper bound, Theorem 2, shows that in order for a tagged CRN **C** to be copy-tolerant, the number of steps needed for **C** to produce any given species must be suitably bounded. The bound is a polynomial function of the volume and other attributes of **C**.

We prove our result for *tagged* CRNs—CRNs with a unique species on the left and right side of each reaction (see Figure 1)— for two reasons. First, the tags make it possible for us to prove strong results. The second reason stems from the fact that our ultimate goal is to prove limits on the power of DSDs, which can be realized with DNA strands, rather than for CRNs which are a useful theoretical abstraction. When translating an untagged CRN to a DSD, two sets of auxiliary DNA strand complexes, so-called transformer, are introduced per reaction of the CRN, one set for each side of the reaction. Each set of transformers includes unique strands that do not otherwise appear in the DSD. The CRN tag species represent the sets of transformer DNA strands. Put another way, to translate an untagged CRN to a DSD using current methods, it is necessary to first add tags to the CRN and then map the tags to the sets of transformer species. Thus, by proving a reachability upper bound for a tagged CRN, we are obtaining a result for the DSD realization of the corresponding untagged CRN. The result would apply also to other realizations of CRNs, perhaps even using molecules other than DNA, in which transformer molecules are needed in the realization. Our earlier result [1] did not apply to general tagged CRNs.

Unlike the example of Figure 1, chemical reactions have associated kinetic rate constants that, along with species counts, determine reaction propensities [8,7]. In particular, a CRN behaves stochastically if multiple reactions are applicable to the molecules available at one or more points in the sequence of reactions. However, in examples such as the stack machine of Qian et al. [3] and the Gray code counter of Condon et al. [1], correctness of the CRN does not depend on the relative propensities of applicable reactions (although efficiency of the CRN does). Since our results are expressed in terms of number of reactions rather than reaction propensities, they apply to stochastic CRNs. We can interpret our reachability result as a hitting time in the stochastic context where a hitting time is the minimum number of reactions required to reach a goal state from a initial state.

New result for strand displacement systems (DSDs). The second main contribution of this paper is a limit on the types of DSDs that are correct in multi-copy settings. In strand displacement (Figure 2), an initially unbound "signal" strand I binds to a "template" T, causing another signal strand O that was initially bound to T to become unbound. DSDs are collections of strands that can change configurations via successive strand displacements in a pre-programmed fashion [13,14,2]; we provide a formal definition later.

Our first result on tagged CRNs implies a reachability upper bound for DSD realizations of CRNs, but says nothing about DSDs more generally. In Theorem 3 we elucidate this simple upper bound which is obtained by applying the CRN result to limited types of DSDs, those whose signal strands consist of exactly two domains: a toehold and a long-domain. However, since the signal types are

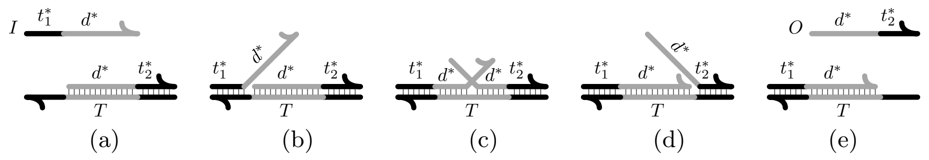


Fig. 2. Strand displacement. (a) An unbound DNA strand I, with a short toehold (dark line) and long-domain (lighter line), plus a duplex consisting of a template strand T and a third strand O that is bound to T. (b) I binds to T via its toehold. (c) Through a process of branch migration, the long-domain of I becomes bound to T, displacing bonds of O. (d) O is bound to T by only a toehold. (e) The toehold bonds break, making O unbound.

limited, this result does not apply to general DSDs. This is because, while tagged CRNs can be translated to DSDs having parameters such as the volume and the number of types of reactants polynomial in the volume of the CRN [9], it is not clear whether the converse is true. To see why, consider signal strands that have three domains: a toehold and two long-domains such that they each start with the same long-domain d^* and toehold t^*, and end with a distinct long-domain. Assume there are δ different types of these signal strands where δ is the number of long-domains on the template we will consider. Note for the DSD template having δ long-domains, over the course of several displacements, there are factorially many different configurations—ways in which signal strands are bound to the template. Figure 3 provides a simple example where any permutation of the signal species could bind to the template. Now, we want to create a tagged CRN that is equivalent to this DSD. Such a tagged CRN in which each template configuration is a distinct species would thus have the number of distinct species and reactions factorial in the volume (number of toehold and long-domains) of the DSD. Since each reaction in the tagged CRN requires a unique tag which needs to be present in the initial configuration, the overall volume of the tagged CRN would be also factorial in the volume of the DSD. It is not clear how else to translate such a DSD to a (tagged) CRN of comparable volume.

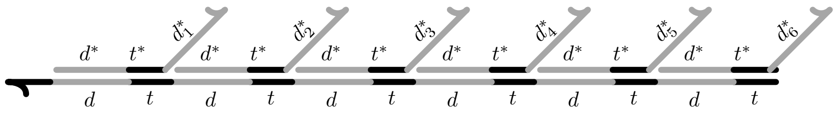


Fig. 3. A template with 6 long-domains and $6! = 720$ possible configurations. Dark lines are toeholds and the lighter ones are long-domains. The template contains $\delta = 6$ toehold-long-domain-toehold blocks. In each block, any one of the signal species may be bound. Thus the number of possible configurations of this template is $\delta! = 6!$.

Can "long" computations be correctly performed by DSDs, even in the presence of many copies? Our second reachability upper bounds for DSDs, Theorems 4 and 5, answer this in the negative, showing that, if sufficiently many copies are present, then any unbound DNA strand that can be produced (i.e., reached) by a sequence of strand displacements can always be reached within a number of displacements that grows at most polynomially in the volume of the single-copy DSD. Thus, for example, we cannot write DSD programs that run correctly in the multi-copy setting and for which the minimum number of displacements needed to produce some given signal strand is exponential in the initial volume.

As further motivation, we describe another application of our DSD reachability bound. The CRN of Figure 4 describes a traditional 3-bit binary counter. Initially, three species, namely $0_3, 0_2$ and 0_1 represent the bits 0 at each index of the counter. Exactly one reaction can advance the counter from each value (all in the forward direction), until the counter reaches $1_3 1_2 1_1$. For the n-bit generalization of this counter, the number of species is just $2n$ (two species per bit) while the number of steps is 2^n. Thus the volume is logarithmic in the number of steps. Another very nice feature of this CRN is that it works correctly even if multiple copies of the initial species are present, not only in the sense of being copy-tolerant but also in the sense that the trace of the multi-copy system is an interleaving of traces of the single-copy system, even in the presence of cross-talk (details omitted).

$$\begin{array}{ll}
(1)\ \mathbf{0}_1 & \rightleftharpoons \mathbf{1}_1 \\
(2)\ \mathbf{0}_2 + 1_1 & \rightleftharpoons \mathbf{1}_2 + 0_1 \\
(3)\ \mathbf{0}_3 + 1_2 + 1_1 & \rightleftharpoons \mathbf{1}_3 + 0_2 + 0_1
\end{array}$$

Fig. 4. Binary counter CRN

However, if tags are added to the counter in order that it can be translated to a DSD using tags as discussed previously, the volume of species for the DSD realization of the counter becomes exponential in n. This is because reaction (1) is executed in the forward direction 2^{n-1} times and is never executed in the reverse direction; thus 2^{n-1} copies of the tag on the left side of reaction (1) must be present initially. Is there an alternative (tag-less) DSD realization of the n-bit CRN binary counter whose volume grows polynomially in n? Our DSD result implies that there is no such realization. If there were, then our reachability upper bound implies that in the multi-copy setting the bit 1_n could be produced in a polynomial number of steps. But since we know that it takes 2^{n-1} steps to produce 1_n even in the multi-copy setting, we have a contradiction.

2 Reachability Upper Bound for CRNs

In this section we first provide formal definitions of tagged CRNs. We then provide our main technical result, and conclude with a restatement of this result to obtain our reachability upper bound theorem for copy-tolerant CRNs.

2.1 Definition of Tagged CRNs

Notation. If $\mathcal{S}$ is a multiset, we will denote the set of distinct elements in $\mathcal{S}$ as $[\![\mathcal{S}]\!]$. If S is a set and k is a positive integer, then $k \cdot S$ denotes the multiset containing k copies of each element in S. Similarly, if $\mathcal{S}$ is a multiset, then $k \cdot \mathcal{S}$ denotes the union of k copies of $\mathcal{S}$. The set operations on multisets are defined in a usual way. In addition, we define the intersection $\mathcal{S} \cap T$ of a multiset $\mathcal{S}$ and a set T as $\mathcal{S} \cap (|\mathcal{S}| \cdot T)$, i.e., $\mathcal{S} \cap T$ contains only elements in $[\![\mathcal{S}]\!] \cap T$, and for each $x \in [\![\mathcal{S}]\!] \cap T$, the number of copies of x in $\mathcal{S} \cap T$ is the same as the number of copies of x in $\mathcal{S}$.

Definition 1 (Tagged CRN). *A* tagged chemical reaction network *is a tuple* $\mathbf{C} = \langle S, T, R, \mathcal{S}_0, \mathcal{T}_0 \rangle$ *with variables defined as follows:*

- S *is a set of signal species and* T *is the set of tag species, and* $S \cap T = \emptyset$.
- R *is a set of reversible or irreversible reactions, where each* $r \in R$ *is an ordered pair* $(\mathcal{I}_r, \mathcal{P}_r)$ *of multisets of signal and tag molecules such that* $\mathcal{I}_r \cap T = \{\tau_r^+\}$ *and* $\mathcal{P}_r \cap T = \{\tau_r^-\}$. *Intuitively, a reaction* $r = (\mathcal{I}_r, \mathcal{P}_r)$ *either consumes the molecules in* $\mathcal{I}_r$ *and produces the molecules* $\mathcal{P}_r$, *or, if the reaction is reversible, it can also consume* $\mathcal{P}_r$ *and produce* $\mathcal{I}_r$. *In the first case, we say that the reaction was applied in the* forward *direction and denote it as* $+r$, *in the second case in the* backward *direction and denote it as* $-r$. *The symbols* $+r$ *and* $-r$ *will be called* oriented *reactions.*
- $\mathcal{S}_0$ *is a multiset of signal molecules and* $\mathcal{T}_0$ *is a multiset of tag molecules present initially at time-step zero. The* volume *of CRN* $\mathbf{C}$ *is the number of molecules in* $\mathcal{S}_0 \cup \mathcal{T}_0$.

Tags limit the number of times a reaction can be applied in the same direction without being applied in the reverse direction. For example, if r is a reversible reaction and $\mathcal{T}_0$ contains only one copy of τ_r^+ and no copies of τ_r^-, then in any valid trace, the oriented occurrences of r has to alternate, starting with $+r$. If r is an irreversible reaction and $\mathcal{T}_0$ contains x copies of τ_r^+, then in any valid trace, there are at most x occurrences of $+r$ (and no occurrences of $-r$). Limiting the number of tags forces a system to recycle molecules in long traces.

Definition 2. *Consider a tagged CRN system* $\mathbf{C} = \langle S, T, R, \mathcal{S}_0, \mathcal{T}_0 \rangle$. *Define the* bandwidth *of signal species* s *as the maximum number of occurrences of* s *on the left side* $\mathcal{I}_r$ *(respectively, left* $\mathcal{I}_r$ *or right side* $\mathcal{P}_r$*) of any irreversible (respectively, reversible) reaction* $r \in R$. *Define the* maximum bandwidth $b_{\mathbf{C}}$ *(respectively,* total bandwidth $B_{\mathbf{C}}$*) of* $\mathbf{C}$ *as the maximum (respectively, the sum) of bandwidth over all signal species in* S. *Similarly, the* proper bandwidth *of signal species* s, *the* maximum proper bandwidth $\tilde{b}_{\mathbf{C}}$ *and the* total proper bandwidth $\tilde{B}_{\mathbf{C}}$ *are defined analogously but using* $\mathcal{I}_r \setminus \mathcal{I}_r \cap \mathcal{P}_r$ *instead of* $\mathcal{I}_r$ *and* $\mathcal{P}_r \setminus \mathcal{I}_r \cap \mathcal{P}_r$ *instead of* $\mathcal{P}_r$. *For any reversible reaction* $r \in R$, *let* t_r *be the maximum of the number of occurrences of* τ_r^+ *or* τ_r^- *in* $\mathcal{T}_0$; *and for any irreversible reaction* $r \in R$, *let* t_r *be the number of occurrences of* τ_r^+ *in* $\mathcal{T}_0$. *Let* $T_{\mathbf{C}}$ *be the sum of* t_r*'s over all reactions* $r \in R$. *We define the* x-copy *of* $\mathbf{C}$, *for* $x \in \mathbb{Z}^+$, *as the CRN* $\langle S, T, R, x \cdot \mathcal{S}_0, x \cdot \mathcal{T}_0 \rangle$.

Let $\rho = r_1, r_2, ..., r_m$ be a sequence of oriented reactions where $r_i \in R$ for all i. For reaction r if sign$(r) = +$, let $\mathcal{A}_r = \mathcal{I}_r$ and $\mathcal{B}_r = \mathcal{P}_r$ where as if sign$(r) = -$, let $\mathcal{A}_r = \mathcal{P}_r$ and $\mathcal{B}_r = \mathcal{I}_r$. The configuration of the system at each step i is defined as $(\mathcal{S}_i, \mathcal{T}_i)$ where $\mathcal{S}_i = (\mathcal{S}_{i-1} \setminus (\mathcal{A}_r \cap S)) \cup (\mathcal{B}_r \cap S)$ and, similarly, $\mathcal{T}_i = (\mathcal{T}_{i-1} \setminus (\mathcal{A}_r \cap T)) \cup (\mathcal{B}_r \cap T)$. A reaction sequence ρ is *valid* if $\mathcal{A}_r \cap S \subseteq \mathcal{S}_{i-1}$ and $\mathcal{A}_r \cap T \subseteq \mathcal{T}_{i-1}$ for all i, meaning that for each molecule in $\mathcal{A}_r$ there must be one in $\mathcal{S}_{i-1} \cup \mathcal{T}_{i-1}$ to remove. A *trace* is a valid reaction sequence.

2.2 The Main Upper Bound

Our main upper bound, Theorem 1, shows that in the multi-copy setting, any product of a tagged CRN can be produced within a number of reactions that is bounded by a function of the number of signal species, the bandwidth, and the number of tags of the CRN.

Theorem 1. *Let* $\mathbf{C} = \langle S, T, R, S_0, \mathcal{T}_0 \rangle$ *be a tagged CRN and let* $s_{end} \in S$*. If some trace of* $\mathbf{C}$ *produces* s_{end}*, then in a* $(|S| - |[\![\mathcal{S}_0]\!]|)(b_\mathbf{C} + \tilde{b}_\mathbf{C}(T_\mathbf{C} - 1)) \leq |S| b_\mathbf{C} T_\mathbf{C}$*-copy CRN of* $\mathbf{C}$*,* s_{end} *can be produced in at most* $(|S| - |[\![\mathcal{S}_0]\!]|)(b_\mathbf{C} + \tilde{b}_\mathbf{C}(T_\mathbf{C} - 1))T_\mathbf{C} \leq (|S| - 1) b_\mathbf{C} T_\mathbf{C}^2$ *steps.*

Proof. Let $\rho = r_1, r_2, \ldots, r_m$ be a valid sequence of oriented reactions in a single-copy system producing s_{end} starting from the initial set S_0. Consider any prefix of this sequence, say $\rho_i = r_1, \ldots, r_i$. Construct a new sequence ρ'_i by canceling all pairs $+r, -r$ for any reaction $r \in R$. It does not matter how these pairs are formed. Let S' be the set of signal molecules appearing on the left hand side of reactions in ρ'_i. Now, let us see what happens if we apply this sequence on the initial set $\mathcal{S}_0 \cup \mathcal{T}_0 \cup k \cdot S'$, where k is sufficiently large so that the reaction sequence is valid. We can make the following observations:

(1) The final number of copies of each signal species is the same as if we would apply ρ_i on $\mathcal{S}_0 \cup \mathcal{T}_0 \cup k \cdot S'$.
(2) For each reaction $r \in R$, ρ'_i contains either only forward or only backward occurrences of r (or no occurrences), and their number is limited by the number t_r of corresponding tags in $\mathcal{T}_0$. As a consequence, the length of ρ'_i is at most $T_\mathbf{C}$.
(3) Consider a signal molecule $s \in S'$. Since each reaction in ρ'_i removes at most $b_\mathbf{C}$ copies of s and the length of ρ'_i is at most $T_\mathbf{C}$, before each reaction in ρ'_i, there are at least $k - \tilde{b}_\mathbf{C}(T_\mathbf{C} - 1)$ copies of s.
(4) Hence, it follows that if we set $k = b_\mathbf{C} + \tilde{b}_\mathbf{C}(T_\mathbf{C} - 1)$, then before each reaction in ρ'_i, there are at least $b_\mathbf{C}$ copies of any signal in S', and hence, the reaction sequence is valid. Note that this is true even if we randomly permute reactions in ρ'_i.

For each signal s appearing in the single-copy trace and not appearing in the initial set $\mathcal{S}_0$, let $r_{\text{index}(s)}$ be the first reaction in ρ which produces a copy (or more) of s. Let $s_1, ..., s_n$ be the sequence of all signals not in $\mathcal{S}_0$ ordered by their indices, i.e., index$(s_1) \leq$ index$(s_2) \leq \cdots \leq$ index(s_n). Furthermore, without loss of generality we can assume $s_n = s_{\text{end}}$. Let $S_i = \{s_1, ..., s_i\}$. We can make one additional observation:

(5) For each s_i, the left side of each reaction in $\rho'_{\text{index}(s_i)}$ contains only signals in $[\![\mathcal{S}_0]\!] \cup S_{i-1}$. By (4), if we start in a configuration which contains the multiset of signals and tags $\mathcal{S}_0 \cup \mathcal{T}_0 \cup (b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)) \cdot ([\![\mathcal{S}_0]\!] \cup S_{i-1})$, $\rho'_{\text{index}(s_i)}$ is a trace producing a copy of s_i.

Construction

(S1) Start with the initial set containing $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ copies of $[\![\mathcal{S}_0]\!]$ and the empty sequence of reactions.
(S2) For each $i = 1, .., n$: add $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ copies of $\mathcal{S}_0 \cup \mathcal{T}_0$ to the initial set and append $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ times sequence $\rho'_{\text{index}(s_i)}$ to the constructed sequence of reactions.

Claim 1. *After each step i in (S2), the constructed sequence is valid and the final configuration contains $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ copies of each signal in $[\![\mathcal{S}_0]\!] \cup S_i$.*

Proof. Proof by induction: *Base case:* For $i = 0$, after (S1), we have $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ copies of each signal in $[\![\mathcal{S}_0]\!]$ and the empty sequence of reactions is valid. *Induction step:* Inductive assumption: before step i, we have $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ copies of each signal in $[\![S_0]\!] \cup S_{i-1}$ and the sequence constructed so far is valid. By (5), if we add a copy of $\mathcal{S}_0$ and run the reaction sequence $\rho'_{\text{index}(s_i)}$ on the current configuration, the trace is valid. By (1), this newly added part (a copy of $\mathcal{S}_0$ and reactions in $\rho'_{\text{index}(s_i)}$) will not decrease the number of any signal. Finally, $\rho'_{\text{index}(s_i)}$ must contain the last reaction of $\rho_{\text{index}(s_i)}$, i.e., $r_{\text{index}(s_i)}$ which produces at least one copy of s_i. If we repeat this $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ times, we will still have at least $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ copies of signals in $[\![\mathcal{S}_0]\!] \cup S_{i-1}$ plus $b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1)$ copies of s_i. □

The bound: The construction uses $(n+1)(b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1))$ copies of $\mathcal{S}_0$, $n(b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1))$ copies of $\mathcal{T}_0$ and repeats $n(b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1))$ times the trace $\rho'_{\text{some index}}$. By (2), the length of each $\rho'_{\text{some index}}$ trace is at most $T_{\mathbf{C}}$, hence the total length of the constructed sequence is at most $n(b_{\mathbf{C}} + \tilde{b}_{\mathbf{C}}(T_{\mathbf{C}} - 1))T_{\mathbf{C}}$. Furthermore, n can be bounded by $|S| - |[\![\mathcal{S}_0]\!]|$. □

Finally, we restate Theorem 1 for copy-tolerant CRNs.

Theorem 2. *If a tagged CRN $\mathbf{C} = \langle S, T, R, S_0, \mathcal{T}_0 \rangle$ is $|S| b_{\mathbf{C}} T_{\mathbf{C}}$-copy-tolerant and s_{end} can be produced in $\mathbf{C}$, then the length of the shortest trace of $\mathbf{C}$ that produces s_{end} is at most $(|S| - 1) b_{\mathbf{C}} T_{\mathbf{C}}^2$.*

A natural question is whether we could improve the bound in condition (3) of the proof of Theorem 1. The following examples shows that it is not possible in general.

Example 1. Assume that ρ contains exactly an even number, T, of oriented reactions $+r_1, \ldots, +r_T$ designed as follows. First for every partition π of ρ into two sets ρ_1^π and ρ_2^π of same size, we introduce a new signal s_π. Let Π be the set of all such partitions. Next, we define reactions $r_1, \ldots, r_T$ in such a way that each of these signals is either an input or a product of each reactions:

$$\mathcal{I}_{r_i} = \{s_\pi;\ r_i \in R_1^\pi, \pi \in \Pi\},$$
$$\mathcal{P}_{r_i} = \{s_\pi;\ r_i \in R_2^\pi, \pi \in \Pi\}.$$

Note that after all reactions in ρ are applied, the number of copies of any of the signals s_π is not changed, since there is exactly $T/2$ reactions in ρ adding one copy of s_π and $T/2$ reactions removing one copy of s_π.

Now, we show that for any permutation of the reactions in ρ, there is a signal molecule with $k - T/2$ copies when the first $T/2$ reactions in this order are applied, and hence, k in (3) has to be set to at least $T/2$. Consider the partition π_0 of ρ into the first and the second $T/2$ reactions of this order. Then the signal s_{π_0} appears in the input set of the first $T/2$ reactions, and thus, the number of copies of s_{π_0} is $k - T/2$ after applying the first $T/2$ reactions.

3 Reachability Upper Bound for DSDs

In this section we first define the type of DSD to which our results apply, along with related notation needed for our results. We then provide our main upper bound, and conclude with a restatement of this result to obtain our reachability upper bound theorem for copy-tolerant DSDs.

3.1 Definition of DSDs

A *basic DNA strand displacement system (DSD)* is a pair $\Delta = (\mathcal{S}, \mathcal{C}_{init})$ of strands and initial configuration (secondary structure) for those strands, plus allowable *positional displacements*, defined as follows.

– $\mathcal{S}$ is a finite multiset of *strands*; $\mathcal{S}$ may contain many strands of a given type. Strands are composed of subsequences of finite strings of symbols, called *domains*. Domains are partitioned into two groups: *toeholds* and *long-domains*. Corresponding to each domain x is a complementary domain x^*; x is a toehold if and only if x^* is. The strands are partitioned into two groups: *signals* or *templates*. There is no bound on the number of toeholds and long-domains of a template or a signal. A *regional interval* is a sequence of domains beginning and ending with a toehold that alternates between toehold and long-domains. Each *template* strand is a concatenation of one or more regional intervals.

  We say that the DSD Δ has *simple signals*, if each signal in $\mathcal{S}$ is composed of exactly one toehold and one long-domain.

– $\mathcal{C}_{init}$ is an initial configuration, where a *configuration* is a secondary structure formed by the strands of $\mathcal{S}$ where domains can bind to their complements. Moreover, each signal strand is either unbound or is bound to a template strand by a single toehold and a single long domain that is adjacent to that toehold and each regional interval of its template must have exactly one open toehold. There are no intra-template bonds or intra-signal bonds. Note that this implies that configurations are pseudoknot-free and contain no hairpin loops. The *volume* of DSD Δ is the number of nucleotides, taken over all strands in the initial configuration $\mathcal{C}_{init}$.

Starting with the initial configuration, DSDs can progress through a sequence of configurations via positional strand displacements (PDs). PDs can move the open toehold of the regional interval to the right or to the left. A PD moving the open toehold to the right is specified by a positive number k, a template strand T and a signal strand called the invader, say of type I, see Figure 2(a), where we can now assume that only positions $k-1, k, k+1$ of template T are shown. The template should have at least $k+1$ domains. The domain d at position k of the template should be a long-domain and should be preceded at position $k-1$ by a toehold, say t. For the displacement to be applicable to a given configuration $\mathcal{C}$, it must be that in $\mathcal{C}$ an additional signal strand, which we refer to as the releasee, is bound to d at position k and to a toehold at position $k+1$ of the template T, and the toehold at position $k-1$ is unbound (open). The invader is unbound in $\mathcal{C}$ and contains the substring t^*d^*.

A displacement models the following steps in Figure 2(b,c,d), when toeholds and long-domains are actual DNA sequences. First, toehold t^* of the invader binds to the toehold t of the template at position $k-1$. Then a branch migration ensues, whereby domain d^* of the invader binds to d at position k of the template and the releasee is no longer bound at this position. Finally, if it exists, the bond between the releasee and the toehold at position $k+1$ is broken. Thus in the resulting configuration $\mathcal{C}'$, substring t^*d^* of the invader is bound to td on the template at positions $k-1$ and k and the releasee is unbound, see Figure 2(e).

Formally a *positional displacement (PD)* of DSD Δ is a tuple of the form (I, T, k, z), where I is a signal strand type, T is a template strand, k is a positive integer and $z \in \{L, R\}$. PD (I, T, k, z) is *applicable* to a configuration $\mathcal{C}$ if the following conditions hold:

1. Strand T has at least $k+1$ domains and the kth domain, say d, must be a long-domain. Also a strand of type O, called the releasee, is bound to the kth domain of T.
2. In the configuration $\mathcal{C}$, a strand of type I is unbound.
3. If $z = R$ the following conditions hold (conditions for $z = L$ are symmetric):

   (a) The $(k-1)$st domain of T must exist and be a toehold, say t.
   (b) A strand of type I must contain substring t^*d^*. (If $z = L$, it must contain d^*t^*.)
   (c) The releasee must also be bound to a toehold at position $k+1$ of T. No other domains of the releasee are bound.

(d) In the configuration $\mathcal{C}$ the toehold at position $k-1$ of strand T is unbound. We call this toehold the *input toehold* of PD (I, T, k, z).

The PD must *release* exactly one strand of type O. Suppose that PD (I, T, k, z) is applicable to $\mathcal{C}$. Let $\mathcal{C}'$ be obtained from $\mathcal{C}$ by removing the bonds between T and the releasee and by adding bonds either between any substring t^*d^* of an unbound strand of type I of $\mathcal{C}$ and the domains td at positions $k-1$ and k of T if $z = L$, or between any substring d^*t^* of I and the substring dt at positions k and $k+1$ of T if $z = R$. Then we say that (I, T, k, z) induces $\mathcal{C}'$ from $\mathcal{C}$. We say that a signal is *simple* if the whole string for the signal consists of t^*d^* or d^*t^*. A DSD is *simple* if all the signals in the DSD are simple. This definition excludes *cooperativity* where two invading strands release a single releasee or one invading strand releases two releasees, because, by definition, every PD must be initiated by one invader and release exactly one releasee.

A sequence of PDs $\rho = p_1, p_2, \ldots p_{|\rho|}$ is *valid* with respect to $\mathcal{C}_{init}$ if there is a sequence $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_{|\rho|+1}$ of configurations of Δ with $\mathcal{C}_1 = \mathcal{C}_{init}$ such that for all $i, 1 \leq i \leq |\rho|$, p_i is applicable to $\mathcal{C}_i$ and induces $\mathcal{C}_{i+1}$ from $\mathcal{C}_i$. When $\mathcal{C}_{init}$ is clear from the context, we simply say that ρ is valid. A valid sequence *produces* a strand $s \in \mathcal{S}$ if in $\mathcal{C}_{|\rho|+1}$, the strand s is unbound. Let Invaders(ρ) be the set of types of invaders of ρ. Let Unbound$(\rho, \mathcal{C}_{init})$ be the set of types of unbound signals in $\mathcal{C}_{|\rho|+1}$ and Unbound(ρ) the set of types of unbound signals in $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_{|\rho|+1}$.

Let $\rho = p_1, p_2, ..., p_{|\rho|}$ be a sequence of PDs. The *regional subsequence* $\rho(T[u, v])$ is the subsequence of ρ whose PDs $p_i = (I_i, T_i, k_i, z_i)$ have positions k_i inside $T[u, v]$.

3.2 The Upper Bounds

First, we use the fact that a DSD with simple strands can be simulated by a tagged CRN with volume that is polynomial in the volume of DSD, and thus we can use the bound in Theorem 1 to obtain the following result. If $\Delta = (\mathcal{S}, \mathcal{C}_{init})$ is a DSD, we define $\Delta^{(x)}$ to be the DSD $(x \cdot \mathcal{S}, x \cdot \mathcal{C}_{init})$.

Theorem 3. *Let Δ be a DSD with simple signals. Let B be the number of types of initially bound signal strands and D be the total number of long-domains of all templates. If Δ can produce s_{end}, then $\Delta^{(2D(2D+B))}$ can produce s_{end} via a sequence of at most $4D^2(2D + B)$ PDs.*

As shown in Figure 3, this strategy will not work in the case of general signal strands. Instead of simulating a DSD by a tagged CRN, in Theorem 4, we will prove a bound for general (i.e., not with simple signals) DSDs directly, reusing some ideas of the proof for tagged CRNs.

Let Δ be a DSD. Roughly, our goal is to show that if there is a valid sequence of PDs that produces a given signal s_{end} from $\mathcal{C}_{init}$, then in a DSD with many copies of $\mathcal{C}_{init}$ there is a valid sequence of PDs that produces s_{end} in a number of steps that is bounded by a polynomial in the volume of Δ. We will build up to the statement and proof through a series of definitions and claims. Our polynomial bound will be a function of two attributes of Δ: the number B of types of signal strands that are all bound (i.e., every copy is bound) in $\mathcal{C}_{init}$ and the total number D of long-domains of all templates in $\mathcal{C}_{init}$.

Let $\alpha = p_1, p_2, ..., p_{|\alpha|}$ be a valid sequence of PDs that produces s_{end}. For each type s of signal strand that is Unbound$(\alpha) \setminus \mathcal{C}_{init}$, let index$(s)$ be the index of the first PD of α that releases s. Let $s_1, \ldots, s_B$ be the sequence of all such signals ordered by their indexes, i.e., index$(s_1) <$ index$(s_2) < \ldots <$ index(s_B). Let $S_i = \{s_1, \ldots, s_i\}$. We assume without loss of generality that there are B such types and also that $s_B = s_{end}$. Let $\alpha_i = p_1, p_2, \ldots, p_{\text{index}(s_i)}$.

Let $T[u, v]$ be a regional interval, $d = (v - u)/2$ the number of long-domains in $T[u, v]$, and let $\alpha_i(T[u, v]) = p_1, p_2, ..., p_{|\alpha_i(T[u,v])|}$, where $p_j = (I_j, T_j, k_j, z_j)$ for every $j = 1, \ldots, |\alpha_i(T[u, v])|$. We construct a subsequence $\beta_i(T[u, v])$ of the PDs in $\alpha_i(T[u, v])$. The PDs in this subsequence will be of two types, *marked* and *connector.*

Markers. Mark the first PD p_1 of $\alpha_i(T[u, v])$, and then mark the *last* PD of $\alpha_i(T[u, v])$ to bind to each long-domain in the regional interval $T[u, v]$. Let $p_{m_1}, \ldots, p_{m_{d+1}}$ be the subsequence of marked PDs ($1 = m_1 < m_2 < \cdots < m_{d+1}$). It is easy to see that the sequence of PD positions, $k_{m_2}, \ldots, k_{m_d}$, consists of two interleaved monotonic subsequences: $U = u + 1, u + 3, \ldots, \ell - 2$ and $V = v - 1, v - 3, \ldots, \ell + 2$, where ℓ is the long-domain position of the last PD in $\alpha_i(T[u, v])$. Furthermore, the marked PDs with the long-domains in the first subsequence have direction R and in the second subsequence direction L.

Connector sequences. Now, we must connect the marked PDs by introducing *connector sequences* of PDs between each consecutive pair of marked PDs with the goal being for each subsequent PD to use the toehold opened by the previous PD.

Let $\bar{z}$ indicate the opposite direction from z. For the connector sequence connecting p_{m_1} and p_{m_2}, select as a connector the *first* PD in $\alpha_i(T[u, v])$ with direction $\bar{z}_{m_2}$ that binds to each long-domain of $T[u, v]$ between positions k_{m_1} and k_{m_2} inclusive. It is easy to see that either all selected connector PDs are before p_{m_2} in the sequence $\alpha_i(T[u, v])$, or $p_{m_1} = p_{m_2}$ and the connector sequence consists of the same PD. In the second case, p_{m_1} is the only PD in $\alpha_i(T[u, v])$ with the long-domain position $u + 1$ or $v - 1$.

Consider $j = 2, \ldots, s$. Each PD of the connector sequence connecting p_{m_j} to $p_{m_{j+1}}$ will be between p_{m_j} and $p_{m_{j+1}}$ in the sequence $\alpha_i(T[u, v])$. We will consider two cases.

1. If $z_{m_j} = z_{m_{j+1}}$, then no connector PDs are needed.
2. If $z_{m_j} \neq z_{m_{j+1}}$, then we select the connectors as follows. In the subsequence between reactions p_{m_j} and $p_{m_{j+1}}$, choose as a connector the *first* PD that binds to each position between k_{m_j} and $k_{m_{j+1}}$, excluding position k_{m_j} and including position $k_{m_{j+1}}$. Note that each PD in this connector sequence must have direction z_{m_j}.

The construction is illustrated in Figure 5. The sequence $\beta_i(T[u, v])$ contains all the marked PDs and all the connector PDs, with distinct indices. Note that

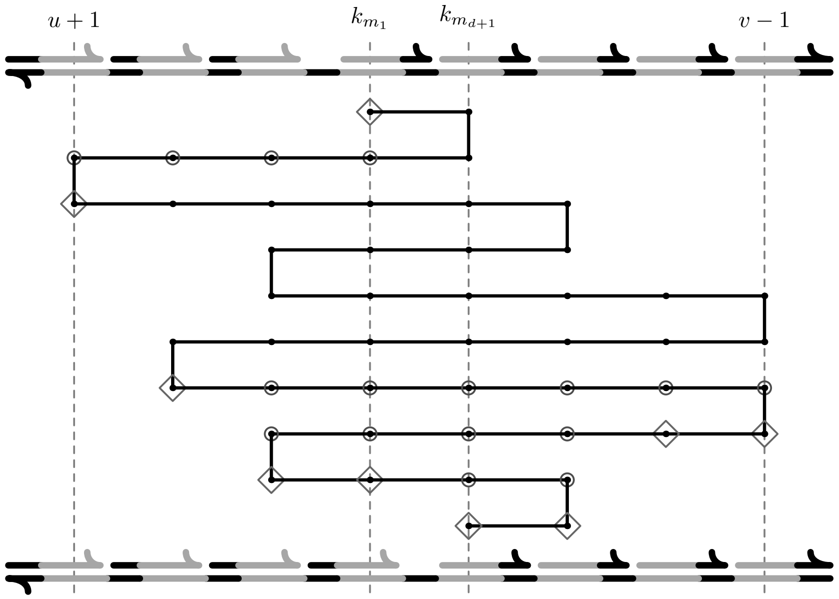


Fig. 5. An example of construction of the $\beta_i(T[u,v])$ subsequence. At the top is the form of the initial configuration of the regional interval and at the bottom the final configuration. Each dot represents a PD of regional subsequence $\alpha_i(T[u,v])$, each diamond a marked PD and each circle a connector PD. The sequence of PDs $\beta_i(T[u,v])$ is then a subsequence of $\alpha_i(T[u,v])$ which contains only the marked and connector PDs.

this is a subsequence of $\alpha_i(T[u,v])$ since for every $j = 1,\ldots,s$, the connector sequence connecting p_{m_j} to $p_{m_{j+1}}$ contains only PDs between between p_{m_j} and $p_{m_{j+1}}$.

We next provide a sequence of claims that we use to prove our main result. All proofs can be found in the full version of the paper.

Claim 2. *Each PD, p_{m_j} for $j \geq 2$ in sequence $\beta_i(T[u,v])$ can use the toehold opened by the previous PD in the sequence. The PD p_{m_1} can use the initially open toehold.*

Claim 3. *The length of $\beta_i(T[u,v])$ is at most $(d+1)(d+2)/2$, where $d = (v-u)/2$ is the number of long-domains in $T[u,v]$.*

Claim 4. *The length of β_i is at most $(D+1)(D+2)/2$ and thus $|\text{Invaders}(\beta_i)| \leq (D+1)(D+2)/2$. Also, $\text{Invaders}(\beta_i)$ contains only types of unbound strands of $\mathcal{C}_{init}$ or strand types in $S_{i-1} = \{s_1,\ldots,s_{i-1}\}$.*

Claim 5. *β_i is valid with respect to*

$$\mathcal{C}_{init} \cup (D+1)(D+2)/2 \cdot (\mathcal{C}_{init} \cup S_{i-1}).$$

Moreover,

$$(D+1)(D+2)/2 \cdot (\mathcal{C}_{init} \cup S_{i-1}) \subseteq \text{Unbound}(\beta_i, \mathcal{C}_{init} \cup (D+1)(D+2)/2 \cdot (\mathcal{C}_{init} \cup S_{i-1})).$$

Claim 6. *Let $\beta_i^{(D+1)(D+2)/2}$ denote the sequence β_i concatenated $(D+1)(D+2)/2$ times, modified just so that the PDs of each copy refer to templates of different copies of $(D+1)(D+2)/2 \cdot \mathcal{C}_{init}$. Then $\beta_i^{(D+1)(D+2)/2}$ is valid with respect to the configuration*

$$(D+1)(D+2)/2 \cdot \mathcal{C}_{init} \cup (D+1)(D+2)/2 \cdot (\mathcal{C}_{init} \cup S_{i-1}).$$

Moreover,

$$(D+1)(D+2)/2 \cdot (\mathcal{C}_{init} \cup S_i) \subseteq \text{Unbound}(\beta_i^{(D+1)(D+2)/2}, \\ (D+1)(D+2)/2 \cdot \mathcal{C}_{init} \cup (D+1)(D+2)/2 \cdot (\mathcal{C}_{init} \cup S_{i-1})).$$

The proof of our main technical result, Theorem 4, follows from the preceding claim.

Theorem 4. *Let Δ be a DSD with B types of initially bound signal strands and let D be the total number of long-domains of all templates. If Δ can produce s_{end}, then $\Delta^{((D+1)(D+2)(B+1)/2)}$ can produce s_{end} via a sequence of at most $(D+1)^2(D+2)^2B/4$ PDs.*

Finally, we restate Theorem 4 for copy-tolerant DSDs. We say that a DSD is x-copy-tolerant if the length of the shortest PD sequence that produces any signal strand s in Δ and in $\Delta^{(x)}$ is the same. A DSD is copy-tolerant if it is x-copy-tolerant for all x.

Theorem 5. *Let Δ be a DSD with B types of initially bound signal strands and let D be the total number of long-domains of all templates. If Δ can produce s_{end} and Δ is $(D+1)(D+2)(B+1)/2$-copy tolerant, then Δ can produce s_{end} via a sequence of at most $(D+1)^2(D+2)^2B/4$ PDs.*

4 Open Questions

There are many open questions about the potential for CRNs and DSDs to be correct in the multi-copy setting. First, can our reachability upper bound results be strengthened? There are two possible ways to strengthen our result for CRNs (Theorem 2): either by reducing the length of the shortest computation needed to produce s_{end} or to show that the system is not x-copy tolerant for some $x < |S|b_C T_C$. Similarly, there are two ways to strengthen the reachability upper bounds for DSDs.

Also, can our result on DSDs be extended to DSDs with more complex primitives, such as cooperative strand displacement [12] or irreversible reactions? What if long-domains can form intra-molecular bonds, e.g., forming hairpins, in addition to inter-molecular bonds?

This paper considers only reachability bounds, i.e., bounds on the number of reactions (steps) needed to reach (produce) a given product. However, real chemical reaction networks behave stochastically, with rates that depend on relative quantities of species. It is plausible that the lack of robustness implied by our theorems, i.e., errors that occur in the multi-copy setting in CRNs that fail to satisfy the conditions of the theorem, would be very unlikely to occur in some CRNs and thus would not be an issue in a real system. Analyses of robustness of CRNs under stochastic assumptions, perhaps computing expected hitting times, would help us better understand the degree to which robustness issues are a problem.

References

1. Condon, A., Hu, A.J., Maňuch, J., Thachuk, C.: Less haste, less waste: On recycling and its limits in strand displacement systems. J R Soc. Interface (2012)
2. Cardelli, L.: Two-domain DNA strand displacement. In: Proc. of Developments in Computational Models (DCM 2010). Electronic Proceedings in Theoretical Computer Science, vol. 26, pp. 47–61 (2010)
3. Qian, L., Soloveichik, D., Winfree, E.: Efficient Turing-Universal Computation with DNA Polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
4. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science 332, 1196–1201 (2011)
5. Qian, L., Winfree, E., Bruck, J.: Neural network computation with DNA strand displacement cascades. Nature 475, 368–372 (2011)
6. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science 314(5805), 1585–1588 (2006)
7. Soloveichik, D.: Robust stochastic chemical reaction networks and bounded tau-leaping. J. Comput. Biol. 16(3), 501–522 (2009)
8. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Nat. Comp. 7, 615–633 (2008)
9. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Nat. Acad. Sci. USA 107(12), 5393–5398 (2010)
10. Yurke, B., Mills, A.P.: Using DNA to power nanostructures. Genet. Program. Evolvable Mach. 4(2), 111–122 (2003)
11. Yurke, B., Turberfield, A.J., Mills Jr., A.P., Simmel, F.C., Neumann, J.L.: A DNA-fuelled molecular machine made of DNA. Nature 406, 605–608 (2000)
12. Zhang, D.Y.: Cooperative hybridization of oligonucleotides. J. Am. Chem. Soc. 133, 1077–1086 (2011)
13. Zhang, D.Y., Turberfield, A.J., Yurke, B., Winfree, E.: Engineering entropy-driven reactions and networks catalyzed by DNA. Science 318, 1121–1125 (2007)
14. Zhang, D.Y., Seelig, G.: Dynamic DNA nanotechnology using strand displacement reactions. Nature Chemistry 3, 103–113 (2011)

Synthesizing Minimal Tile Sets for Complex Patterns in the Framework of Patterned DNA Self-Assembly

Eugen Czeizler[1] and Alexandru Popa[2]

[1] Department of Information and Computer Science,
School of Science
[2] Department of Communications and Networking,
School of Electrical Engineering,
Aalto University P.O. Box 15400, FI-00076 Aalto, Finland
firstname.lastname@aalto.fi

Abstract. Ma and Lombardi (2009) introduce and study the Pattern self-Assembly Tile set Synthesis (PATS) problem. In particular they show that the optimization version of the PATS problem is NP-hard. However, their NP-hardness proof turns out to be incorrect. Our main result is to give a correct NP-hardness proof via a reduction from the 3SAT. By definition, the PATS problem assumes that the assembly of a pattern starts always from an "L"-shaped seed structure, fixing the borders of the pattern. In this context, we study the assembly complexity of various pattern families and we show how to construct families of patterns which require a non-constant number of tiles to be assembled.

1 Introduction

Algorithmic self-assembly of nucleic acids (DNA and RNA) has advanced extensively in the past 30 years, from a seminal idea to the current designs and implementations of complex nano-structures and devices [3, 5, 8, 13]. Currently, DNA self-assembly is one of the most promising methodologies for computationally directed bottom-up manufacturing. A key technique here is the use of a self-assembling system of DNA tiles to build a scaffold structure on which functional units (ions, proteins, other molecules) are deposited [5, 12, 20]. Such an approach became even more attractive with the introduction of Rothemund's [15] DNA origami technique, as the "origami-tiles" obtained in this way are highly addressable. Recently, several laboratory techniques for synthesising the requisite 2D DNA template lattices have been demonstrated by several groups [7, 14, 21].

As a pre-requisite of the manufacture of anisotropic structures, such as electronic circuit designs, the DNA structures grounding these constructions need to be addressable. When the template is construed as a tiling from a family of DNA tiles, one can view the base tiles as being coloured according to their different functionalities, and the completed template implementing a desired colour pattern. In [2] for example, the authors presented a scheme whereby the synthesis of arbitrary logical circuits can be achieved by the self-assembly of a collection of 14 tiles, each being differentially functionalised by appropriate carbon nanotube deposits.

The Pattern self-Assembly Tile set Synthesis (PATS) problem is first introduced by Ma and Lombardi [9, 10], and placed inside the general Tile Assembly Model (TAM) framework, introduced by Winfree [16]. Given a coloured pattern of a rectangular shape, the PATS problem asks to construct a Tile Assembly System (TAS) assembling the pattern which obeys the following constrains: the TAS uses only strength-one glues (while the total temperature of the system is two), and the seed structure from which the assembly commence consists of the complete lower and left-most border of the pattern[1]. From a practical perspective, the optimization version of the PATS problem which asks to minimize the number of tile types needed for assembling a given pattern, is of particular interest. This problem was considered in several publications, and a number of algorithms were proposed. Ma and Lombardi provide two greedy heuristics [9, 10]; subsequently Göös and Orponen [4] present an exhaustive partition-search branch-and-bound algorithm. Recently, Lempiäinen et al. [6] consider a randomized search algorithm which, coupled with a parallel implementation, achieves almost optimal results for large patterns. Finally, in [11], the authors consider a version of the PATS problem, the assembly of coloured 1D patterns, applicable this time in the framework of staged self-assembly.

Ma and Lombardi [10] consider the computational complexity of the optimization PATS problem and claim that the problem is NP-hard. However the proof of this result is incorrect: the authors show a reduction from the PATS problem to an optimization problem on a tile and a bond graph and show that the latter is NP-hard. Since they don't show a reduction in the reverse direction, their result does not prove the NP-hardness of PATS.

In the present work we provide a correct proof of the NP-hardness of the (optimization version of) PATS problem via a reduction from the 3SAT problem. Another problem that we consider is the design of complex patterns that need a large number of tile types for their assembly. The problem appears as a natural consequence of the testing effort for several PATS approximation algorithms [6]. In this framework, in order to test the efficiency of the algorithm, one needs a family of patterns which require a non-constant number of tiles to be assembled. In here we provide such families of patterns of unbounded assembly complexity.

[1] In the literature, the seed assembly of a TAS is often taken to be a single seed tile whereas in this context we consider an L-shaped seed assembly. These boundaries can always be self-assembled using $m + n + 1$ different tiles with strength-2 glues, but for practical purposes we allow for the possibility of using, for example, DNA origami techniques to construct these boundary conditions, see e.g. [18, 19].

2 The Tile Assembly Model and the PATS Problem

2.1 The Tile Assembly Model

Let $\mathcal{D} = \{N, E, S, W\}$ be the set of directions. To each of them we associate a corresponding function $N, E, S, W : \mathbb{Z}^2 \to \mathbb{Z}^2$, so that $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S = N^{-1}$ and $W = E^{-1}$. If Σ is a finite set of *glue types*, let $s : \Sigma \times \Sigma \to \mathbb{N}$ be a *glue strength* function, i.e., $s(\sigma_1, \sigma_2) = s(\sigma_2, \sigma_1)$ for all $\sigma_1, \sigma_2 \in \Sigma$. In this paper, we restrict to strength-one self-sticking glues, that is, $s(\sigma, \sigma) = 1$ for all $\sigma \in \Sigma$, and zero otherwise. A *tile type* $t \in \Sigma^4$ is a quadruple $(\sigma_N(t), \sigma_E(t), \sigma_S(t), \sigma_W(t))$ of glue types associated with the four sides of a unit square (we assume the tiles cannot be rotated or reflected). An *assembly* $\mathcal{A}$ is a partial mapping from $\mathbb{Z}^2$ to Σ^4. A *tile assembly system* (TAS) $\mathcal{T} = (T, \mathcal{S}, s, \tau)$ consists of a finite set T of tile types, an assembly $\mathcal{S}$ called the *seed assembly*, a glue strength function s and a *temperature* $\tau \in \mathbb{Z}^+$ (we restrict here to $\tau = 2$). By definition, we assume that the seed assembly S is stable and can not be disassembled[2].

For a TAS $\mathcal{T} = (T, \mathcal{S}, s, \tau)$ and an assembly $\mathcal{A}$, a new tile can be adjoined to $\mathcal{A}$ if it shares a common boundary with tiles that bind it into place with total strength at least τ. For two assemblies $\mathcal{A}$ and $\mathcal{A}'$, $\mathcal{A}' = \mathcal{A} \cup \{((x, y), t)\}$, we write $\mathcal{A} \to_{\mathcal{T}} \mathcal{A}'$ to denote such a tile addition event. By denoting as $\to_{\mathcal{T}}^*$ the reflexive transitive closure of $\to_{\mathcal{T}}$ we say that a TAS $\mathcal{T}$ *produces* an assembly $\mathcal{A}$ if $\mathcal{S} \to_{\mathcal{T}}^* \mathcal{A}$. That is, $\mathcal{A}$ is an *extension* of the seed assembly $\mathcal{S}$. We denote by $\mathrm{Prod}\,\mathcal{T}$ the set of all assemblies produced by $\mathcal{T}$. An assembly $\mathcal{A}$ that cannot be further extended is called a *terminal assembly*. We denote by $\mathrm{Term}\,\mathcal{T}$ the set of terminal assemblies of $\mathcal{T}$.

We say that a TAS $\mathcal{T}$ is *deterministic* if for any assembly $\mathcal{A} \in \mathrm{Prod}\,\mathcal{T}$ and for every $(x, y) \in \mathbb{Z}^2$ there exists at most one $t \in T$ such that $\mathcal{A}$ can be extended with t at position (x, y). In case of finite assemblies, an equivalent definition of determinism is that all *assembly sequences* $\mathcal{S} \to_{\mathcal{T}} \mathcal{A}_1 \to_{\mathcal{T}} \mathcal{A}_2 \to_{\mathcal{T}} \cdots$ terminate and $\mathrm{Term}\,\mathcal{T} = \{\mathcal{P}\}$ for some assembly $\mathcal{P}$. In this case we say that $\mathcal{T}$ *uniquely produces* $\mathcal{P}$.

2.2 The PATS Problem

Let $[n]$ be the ordered set $\{1, 2, ...n\}$. For some $m, n \geq 1$, a mapping $c : [m] \times [n] \to [k]$ defines a *k-colouring* or a *k-coloured pattern* (or simply pattern). The PATS problem (first considered in [9]) asks to construct a TAS that produces the pattern. However, when doing so, we impose a strict condition on the starting point of this process and on the glue strength of the tiles. Namely, we impose that the seed assembly consists of all boundary tiles placed on the West and South borders of the m by n rectangle and that all the tiles used for extending the seed assembly have strength-one glues.

[2] On some experimental implementations of the TAM, the seed assembly is implemented using e.g., DNA origami [18, 19].

Definition 1 (Pattern self-Assembly Tile set Synthesis (PATS) [9])

Given: *A k-colouring $c : [m] \times [n] \to [k]$.*
Find: *A tile assembly system $\mathcal{T} = (T, \mathcal{S}, s, 2)$ such that*

P1. The tiles in T have bonding strength 1.
P2. The domain of $\mathcal{S}$ is $[0, m] \times \{0\} \cup \{0\} \times [0, n]$ and all the terminal assemblies have the domain $[0, m] \times [0, n]$.
P3. There exists a colouring $d : T \to [k]$ such that for each terminal assembly $\mathcal{A} \in \mathrm{Term}\,\mathcal{T}$ we have $d(\mathcal{A}(x, y)) = c(x, y)$ for all $(x, y) \in [m] \times [n]$.

In [4] it was proved that any minimal solution to the PATS problem has to be deterministic. Thus, in the following we restrict ourselves to the case of deterministic TAS. In this case, the self-assembly process proceeds in a uniform manner directed from South-West to North-East. Hence we can make the following observation regarding deterministic TAS in the context of the PATS problem:

Proposition 1. [4] *Solutions $\mathcal{T} = (T, \mathcal{S}, s, 2)$ of the PATS problem are deterministic precisely when for each pair of glue types $(\sigma_1, \sigma_2) \in \Sigma^2$ there is at most one tile type $t \in T$ so that $\sigma_S(t) = \sigma_1$ and $\sigma_W(t) = \sigma_2$.*

Throughout the paper we also use the following notations. We define an l-colour *sub-pattern* p (or sub-pattern) as a partial function $p : [w] \times [h] \to [l]$. We say that the sub-pattern p is *complete*[3] if it is defined on all positions inside $[w] \times [h]$. For two complete sub-patterns p_1 and p_2, we denote by $p_1 \cdot p_2$ the pattern formed by horizontally adjoining the two patterns. Also, for a collection of complete sub-patterns p_i, $1 \le i \le n$ we denote by $\prod_{1 \le i \le n} p_i$ the pattern $p_1 \cdot p_2 \cdot \ldots \cdot p_n$. Given a sub-pattern $p : [w] \times [h] \to [l]$, we say that a pattern $c : [m] \times [n] \to [k]$ contains (or includes) p if $m \ge w$, $n \ge h$, $k \ge l$, and there exists $(x, y) \in \mathbb{N}^2$ such that $p(q, r) = c(q + x, r + y)$ for all positions $(q, r) \in [w] \times [h]$ where p is defined.

Given a TAS $\mathcal{T} = (T, \mathcal{S}, s, 2)$ and a k-colouring tile function $d : T \to [k]$, we say that the pair $(\mathcal{T}, d)$ is a *k-coloured TAS* (or simply TAS if no confusion can arise). Then, given a pattern $c : [m] \times [n] \to [k]$, we say that the TAS $(\mathcal{T}, d)$ *assembles* the pattern c (starting from a full-border seed) if the domain of $\mathcal{S}$ is $[0, m] \times \{0\} \cup \{0\} \times [0, n]$, the domain of $\mathcal{T}$'s terminal assembly $\mathcal{A}$ is $[0, m] \times [0, n]$, and for all $(x, y) \in [m] \times [n]$, $d(\mathcal{A}(x, y)) = c(x, y)$.

3 The Algorithmic Complexity of the PATS Minimization Problem

In this section we prove that the PATS minimization problem is NP-hard. In particular, we consider the decision version of the PATS problem, which we term

[3] Clearly, the notions of pattern and complete sub-pattern overlap each other. In general, we are going to use the term complete sub-pattern when we want to indicate that the pattern in case will be later included into some bigger structure.

Fixed-N PATS problem, and prove the NP-completeness of this decision problem via a reduction from the 3SAT. The Fixed-N PATS problem asks if there exists a tiling for the given pattern that uses precisely N tiles. Before giving the proof of the NP-completeness result we need to give a few intermediary results. Due to space limitations, most of the proofs are left to the full version.

Proposition 2. *For any given colour "1" and any value $n \geq 1$ there exists a family of sub-patterns p_m, $m \geq 1$ such that for any pattern c containing all p_m, any deterministic TAS assembling c contains at least n tile types coloured "1".*

The above result can be applied whenever we want to enforce the existence of some n number of tile types which are similarly coloured inside a TAS. In particular, we use it for the case $n = 3$ as described in the following corollary.

Corollary 1. *For any given colour "1" we can design a sub-pattern $p : [10] \times [2] \to \{1, C^1, ..., C^5\}$ such that for any TAS $(\mathcal{T}, d)$ assembling a pattern c containing p we have that $|\{t \in T \mid d(t) = 1\}| \geq 3$. That is, the TAS must contain at least three different tile types which are coloured "1".*

Indeed, the sub-pattern p defined as in Figure 1 satisfies the condition.

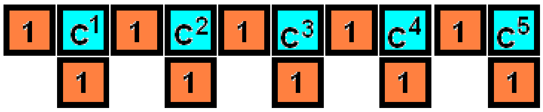


Fig. 1. A sub-pattern p enforcing the existence of at least three different tile types coloured "1"

Observation 3. *Note that for the previously constructed sub-pattern p and a pattern c containing it, any TAS $(\mathcal{T}, d)$ assembling c needs at least 8 different tile types: 3 coloured "1", and one for each of the colours $C^i, 1 \leq i \leq 5$.*

In some of our constructions, we need not only to enforce the existence of two different tile types with the same colour, but also to enforce that the East glues of these tiles are different. This is achieved by the following construction.

Proposition 4. *For any given colour "1" we can design a sub-pattern $p : [8] \times [2] \to \{1, A, B, C\}$ such that for any TAS $(\mathcal{T}, d)$ assembling a pattern c containing p we have $|\{t \in T \mid d(t) = 1\}| \geq 2$. Moreover, if $\mathcal{T}$ is such that $|\{t \in T \mid d(t) = 1\}| = 2$ and $|\{t \in T \mid d(t) = A\}| = |\{t \in T \mid d(t) = B\}| = |\{t \in T \mid d(t) = C\}| = 1$, then the two tiles $t_1, t_2 \in T$ for which $d(t_1) = d(t_2) = 1$ satisfy that $\sigma_E(t_1) \neq \sigma_E(t_2)$.*

The pattern enforcing the above constraint is described in Figure 2.

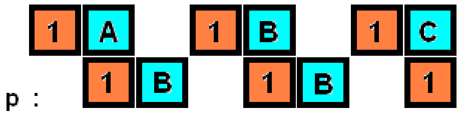


Fig. 2. A pattern p for which we need at least two tile types coloured "1", and for which if we have exactly two tile types coloured "1" then these tiles must differ on their East glue

Observation 5. *Note that for the previously constructed sub-pattern p and a pattern c containing it, any TAS $(\mathcal{T}, d)$ assembling c needs at least 5 different tile types: two coloured "1", and one for each of the colours A, B, and C, respectively.*

The following result provides a method of enforcing the existence of n different tile types which are identically coloured by designing a pattern of size polynomial in n.

Proposition 6. *For any given colour "1" and any values $n, k \geq 1$ we can design a sub-pattern $p : [2nk] \times [2] \to \{1, C_i, F^j\}$, with $1 \leq i \leq n$ and $1 \leq j \leq k$ with the following property. For any coloured TAS $(\mathcal{T}, d)$ assembling a pattern c containing the sub-pattern p we have that either $|\{t \in T \mid d(t) = 1\}| \geq n$, in which case $|T| \geq 2n + k$ or if $|\{t \in T \mid d(t) = 1\}| = m$ with $m < n$ then $|T| \geq m + n + k\frac{n}{m}$.*

The pattern enforcing the above constraint is obtained by horizontally adjoining the patterns described in Figure 3.

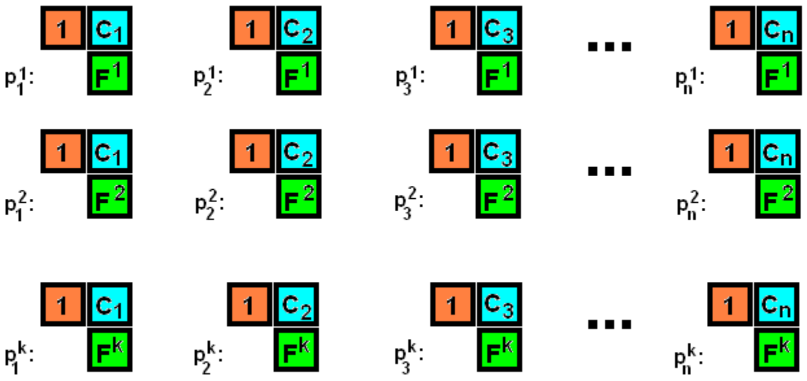


Fig. 3. The $(p_n^k)_{n,k}$ family of sub-patterns used in Proposition 6; all $C_1, \ldots, C_n$, and $F^1, \ldots, F^k$ represent different colours

By considering the parameter k large enough, it becomes more and more un-economical to use some $m < n$ tile types coloured "1", instead of using n different ones. In particular, the case $k = n$ represents the threshold value for this phenomenon.

Corollary 2. *For any given colour "1" and any $n \geq 1$ we can design a complete sub-pattern $\mathit{pol}(n) : [2n^2] \times [2] \to \{1, C_i, F^j, D_i^j\}$, $1 \leq i, j \leq n$, with the following property. There exists a TAS $(\mathcal{T}, d)$ assembling the complete sub-pattern $\mathit{pol}(n)$ such that $|\{t \in T \mid d(t) = 1\}| = n$ and $|T| = 3n + n^2$. Moreover, for any other TAS $(\mathcal{T}', d)$ assembling $\mathit{pol}(n)$ such that $|\{t \in T' \mid d(t) = 1\}| = m$ with $m < n$, we have that $|T'| - |T| > 0$.*

Indeed, by replacing k with n in the inequality of Proposition 6 we obtain: (Note that a number of n^2 new colours, and thus tile types, are needed to complete the pattern.)

$$|T'| - |T| \geq \left(m + n + n\frac{n}{m} + n^2\right) - \left(3n + n^2\right) = m - 2n + \frac{n^2}{m} = \frac{(n-m)^2}{m} > 0.$$

We are now ready to construct the 3SAT reduction. Let $C = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be an arbitrary instance of a 3SAT problem involving a set of n variables $V = \{x_1, x_2, \ldots, x_n\}$. For each $1 \leq j \leq m$, let $C_j = (X_j^1 \vee X_j^2 \vee X_j^3)$, where $X_j^1, X_j^2, X_j^3 \in \{x_1, x_2, \ldots x_n\} \bigcup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}$.

First, we associate to the pair (V, C) a complete sub-pattern $\mathit{patt}(c)$, and ascertain the minimal TAS assembling it. Then, we define two more complete sub-patterns $\mathit{assgn}(c)$ and $\mathit{val}(c)$: the first one implements a truth assignment of the variables in V, while the second is used to verify whether the previous assignment validates the formula C. In particular, we show that if all of these complete sub-patterns are adjoined, the minimal TAS assembling them has the following properties:

– While assembling the pattern $\mathit{assgn}(c)$, no new tile types are needed (except for covering new colours appearing in $\mathit{assgn}(c)$ but not in $\mathit{patt}(c)$);
– While assembling the pattern $\mathit{val}(c)$, no new tile types are needed (except for covering new colours not in $\mathit{patt}(c) \cdot \mathit{assgn}(c)$), if and only if the formula C is satisfiable.

Let $Rainbow_C$ (or simply Row) denote the following set of all different colours: $Row = \{\alpha_{x_i}, A^{x_i}, B^{x_i}, C^{x_i}, D_l^{x_i}, x_i, \neg x_i, LC, Cl_j, E_j^g, G_j^e, 1, C_i, F^h, D_i^h\}$, where $1 \leq i, h \leq n$, $\quad 1 \leq j \leq m$, $\quad 1 \leq l \leq 7$, $\quad 1 \leq g \leq 5$, $\quad 1 \leq e \leq 8$.
For each 3SAT instance C with n variables and m clauses we associate a unique complete sub-pattern $\mathit{patt}(c) : [12n + 16m + 2n^2] \times [2] \to Row$, consisting of the following families of complete sub-patterns:

– for all $1 \leq i \leq n$ define $\mathit{def}(x_i) : [12] \times [2] \to \{\alpha_{x_i}, A^{x_i}, B^{x_i}, C^{x_i}, D_l^{x_i}, x_i, \neg x_i, LC\}$, $1 \leq l \leq 7$, as in Figure 4;
– for all $1 \leq j \leq m$ let $\mathit{def}(C_j) : [16] \times [2] \to \{Cl_j, E_j^g, [X_j^1], [X_j^2], [X_j^3], G_j^e, LC\}$, where $1 \leq e \leq 8$, $1 \leq g \leq 5$ and $[X_j^1], [X_j^2]$, and $[X_j^3]$ stand for the colours associated with the signed-variables in clause C_j. (That is for example, if $C_j = (x_2 \vee \neg x_5 \vee \neg x_6)$, then $\{[X_j^1], [X_j^2], [X_j^3]\} = \{x_2, \neg x_5, \neg x_6\}$.) The pattern $\mathit{def}(C_j)$ is defined as in Figure 5;
– $\mathit{pol}(n) : [2n^2] \times [2] \to \{1, C_i, F^h, D_i^h\}$, with $1 \leq i, h \leq n$, as defined in Corollary 2.

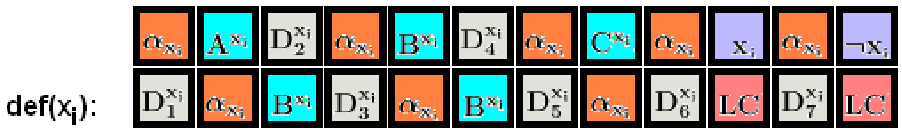

def(x_i):
α_{x_i} A^{x_i} D_2^{x_i} α_{x_i} B^{x_i} D_4^{x_i} α_{x_i} C^{x_i} α_{x_i} x_i α_{x_i} ¬x_i
D_1^{x_i} α_{x_i} B^{x_i} D_3^{x_i} α_{x_i} B^{x_i} D_5^{x_i} α_{x_i} D_6^{x_i} LC D_7^{x_i} LC

Fig. 4. The complete sub-pattern $\mathit{def}(x_i)$ associated to variable x_i; all $\alpha_{x_i}, A^{x_i}, B^{x_i}, C^{x_i}, D_l^{x_i}, x_i, \neg x_i, LC$ represent different colours, where $1 \leq i \leq n$ and $1 \leq l \leq 7$.

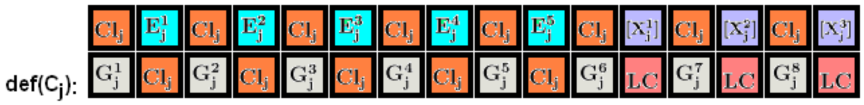

def(C_j):
Cl_j E_j^1 Cl_j E_j^2 Cl_j E_j^3 Cl_j E_j^4 Cl_j E_j^5 Cl_j [X_j^1] Cl_j [X_j^2] Cl_j [X_j^3]
G_j^1 Cl_j G_j^2 Cl_j G_j^3 Cl_j G_j^4 Cl_j G_j^5 Cl_j G_j^6 LC G_j^7 LC G_j^8 LC

Fig. 5. The complete sub-pattern $\mathit{def}(C_j)$ associated to clause C_j; all Cl_j, E_j^g, G_j^e represent different colours, where $1 \leq j \leq m$, $1 \leq g \leq 5$ and $1 \leq e \leq 8$. The colours $[X_j^1]$, $[X_j^2]$, and $[X_j^3]$ stand for the colours associated with the signed-variables in clause C_j and are the same as in Figure 4; similarly, the colour LC is the same as in $\mathit{def}(x_i)$.

Proposition 7. *A minimal TAS $(\mathcal{T}, d)$ assembling the complete sub-pattern $\mathit{patt}(c)$ (where $\mathit{patt}(c) = \prod_{1 \leq i \leq n} \mathit{def}(x_i) \cdot \prod_{1 \leq j \leq m} \mathit{def}(C_j) \cdot \mathit{pol}(n)$), contains $1 + 17n + 16m + n^2$ different tile types. Moreover, for any variable $x_i \in V$, $|\{t \in T \mid d(t) = \alpha_{x_i}\}| = 2$, and no two tiles coloured by some α_{x_i} and α_{x_j} colours (possibly with $i = j$) have the same glue on their East edge.*

Proof. Intuitively, by previous results we have that at least two tiles are needed for each α_{x_i} colour, three tiles for each Cl_j colour, n tiles for the "1" colour, and one tile for each of the remaining $1 + 14n + 13m + n^2$ different colours. Moreover, such a construction using this minimal number of tiles is reachable. Also, since in such a construction only one tile is coloured LC, all the α_x coloured tiles become uniquely identifiable by their East glue.

We introduce now the patterns $\mathit{assgn}(c) : [3n] \times [2] \to \{\alpha_{x_i}, 1, LB, H_i, O_i^1, O_i^2\}$, $1 \leq i \leq n$ and $\mathit{val}(c) : [2m] \times [2] \to \{Cl_j, 1, LB, K_j, O_j'^1, O_j'^2\}$, $1 \leq j \leq m$, presented in Figure 6 and Figure 7, respectively. Intuitively, in the first pattern we introduce a truth assignment for all the variables, while in the second one we are checking whether these truth values validate all the clauses.

Proposition 8. *Let $(\mathcal{T}, d)$ be a minimal coloured TAS assembling the complete sub-pattern $\mathit{patt}(c) \cdot \mathit{assgn}(c)$. Then, in this tile system there exist exactly n tile types coloured "1".*

Proof (Idea of the proof). Indeed, if less than n tiles are coloured "1", say p, by Proposition 2 the number of tiles for assembling $\mathit{pol}(n)$ increases with more than $n - p$. Thus, the advantage obtained by using less than n tiles coloured "1" becomes completely lost.

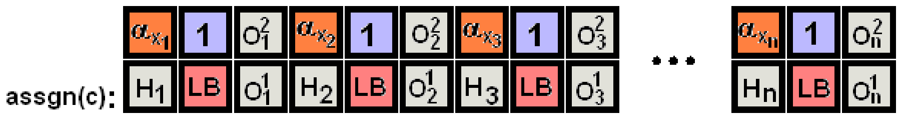


Fig. 6. The complete sub-pattern $\mathit{assgn}(c)$; all $\alpha_{x_i}, 1, LB, H_i, O_i^1, O_i^2$ represent different colours, where $1 \leq i \leq n$.

Corollary 3. *The minimal coloured TAS assembling the complete sub-pattern* $\mathit{patt}(c) \cdot \mathit{assgn}(c)$ *contains* $2 + 20n + 16m + n^2$ *tile types as follows:*

- *two different tile types coloured* α_{x_i} *for each* $1 \leq i \leq n$*;*
- *three different tile types coloured* Cl_j *for each* $1 \leq j \leq m$*;*
- n *different tile types coloured "1";*
- *one tile type for each of the remaining* $2 + 17n + 13m + n^2$ *different colours.*

Consider a minimal coloured TAS as above. Since each tile coloured "1" is placed on the right hand side of a tile coloured α_{x_i}, its West glue must be equal to the East glue of the tile coloured α_{x_i}; that is, it must be equal with the West glue of either the tile coloured x_i, or the tile coloured $\neg x_i$. Thus, that tile type coloured "1" is chosen to non-deterministically represent either the "true" or the "false" value of the variable x_i.

Consider now the complete sub-pattern, $\mathit{val}(c)$, in which we are performing a checking whether the truth assignment implemented inside $\mathit{assgn}(c)$ validates all the clauses. This is achieved in the following way. For each clause C_j, inside the pattern $\mathit{def}(C_j)$ (namely on positions $\mathit{def}(C_j)(11,2)$ to $\mathit{def}(C_j)(16,2)$), we placed a tile coloured Cl_j to the left of exactly three signed variable colours (that is either colours of the form x or $\neg x$). These signed variable colours correspond to exactly those three variable truth assignments which validate the clause C_j. Thus, if as a result of assembling the pattern $\mathit{assgn}(c)$ we introduced a tile type coloured "1" which was assigned either of the above three truth-values validating the clause C_j, let us denote this tile by t_1^x, we proceed as follows. Inside the pattern $\mathit{val}(c)$, on the position coloured Cl_j, we place the tile type t_{Cl_j} (out of the three possible choices) for which $\sigma_E(t_{Cl_j}) = \sigma_W(t_1^x)$, and on its right hand side we place t_1^x. Otherwise, if none of the tile types coloured "1" are corresponding to a truth-value validating the clause C_j, then it means that the West glue of all the n previously introduced tiles coloured "1" is different than the East glue of all the

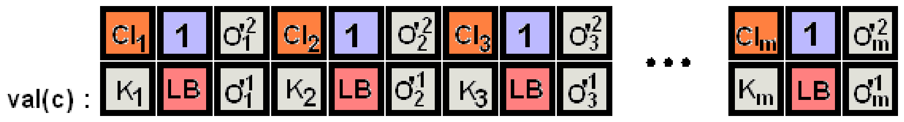


Fig. 7. The complete sub-pattern $\mathit{val}(c)$; all $Cl_j, 1, LB, K_i, O_i'^1, O_i'^2$ represent different colours, where $1 \leq i \leq n$.

tile types coloured Cl_j. Thus, a new tile type (either coloured Cl_j or coloured "1") must be introduced.

Theorem 1. *Let $C = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a an instance of a 3SAT problem with n variables, $V = \{x_1, x_2, \ldots, x_n\}$. Let $(\mathcal{T}, d)$ be a minimal TAS assembling the pattern $\mathit{patt}(c) \cdot \mathit{assgn}(c) \cdot \mathit{val}(c)$. Then $|T| = 2 + 20n + 19m + n^2$ if and only if the formula C is satisfiable.*

Proof. Assume first that the formula C is satisfiable; that is, there exists a truth-value assignment for the variables $x_1, \ldots, x_n$ such that all the m clauses $C_1, \ldots, C_m$ are satisfied. We construct the minimal TAS assembling first the horizontally adjoined patterns $\mathit{patt}(c)$ and $\mathit{assgn}(c)$, such that the tiles coloured "1" (and by Corollary 3 we have exactly n of them) are associated with the above truth-value assignments of the variable. By Corollary 3, such a minimal TAS contains exactly $2 + 20n + 16m + n^2$ different tile types. By the above considerations, we can adjoin also the pattern $\mathit{val}(c)$ and tile the entire structure using an extra of $3m$ tile types, one for each of the new colour used in this pattern: $K_j, O_j'^1$, and $O_j'^2$. Thus, the total number of different tile types reaches the value $2 + 20n + 19m + n^2$.

Assume now that although the formula is not satisfiable, we can still assemble the three adjoined sub-patterns by a TAS using the above mentioned number of tile types. Because inside the sub-pattern $\mathit{val}(c)$ we use $3m$ colours which do not appear in any of the other two sub-patterns, we can conclude that the TAS $\mathcal{T}$ assembles the sub-pattern $\mathit{patt}(c) \cdot \mathit{assgn}(c)$ using at most $2 + 20n + 16m + n^2$ different tile types. Furthermore, by Corollary 3, we have that at least this number of tile types is needed and moreover, the restriction of $\mathcal{T}$ to these tile types represents a minimal coloured TAS assembling the sub-pattern $\mathit{patt}(c) \cdot \mathit{assgn}(c)$. Also, by the same result, we can partition the tile types as follows:

– two different tile types coloured α_{x_i} for each $1 \leq i \leq n$; denote them as $t^1_{\alpha_{x_i}}$ and $t^2_{\alpha_{x_i}}$
– three different tile types coloured Cl_j for each $1 \leq j \leq m$;
– n different tile types coloured "1";
– one tile type for each of the remaining $2 + 17n + 13m + n^2$ different colours.

Proposition 4 implies that for each variable x_i, $\sigma_E(t^1_{\alpha_{x_i}}) \neq \sigma_E(t^2_{\alpha_{x_i}})$. Moreover, since we use a unique tile coloured LC we have that all the tiles α_x with $x \in V$, are uniquely identifiable by the glue placed on their East side. By Corollary 3, we also know that there are exactly n tile types coloured "1" which are used for assembling the pattern $\mathit{patt}(c) \cdot \mathit{assgn}(c)$. By the above observation regarding the uniqueness of the East glue of any of the tiles coloured α_x, and since there exist a unique tile type coloured LB, we obtain that all n tile types coloured "1" differ on their West glue. Moreover, we also have that for each $x \in V$, there exists exactly one tile type coloured "1", denote it as 1^x for which $\sigma_W(1^x)$ is either equal with $\sigma_W(t^x)$ or with $\sigma_W(t^{\neg x})$, where t^x and $t^{\neg x}$ denote the unique tile types coloured x and $\neg x$. Thus, the n tiles coloured "1" (inside the sub-pattern

$\mathit{assgn}(c)$) determine (uniquely) a truth assignment for the variables $x_i, \ldots, x_n$. Since the coloured TAS contains a total of $2+20n+19m+n^2$ different tile types, out of which $2+20n+16m+n^2$ are used for assembling $\mathit{patt}(c) \cdot \mathit{assgn}(c)$ and $3m$ are coloured K_j, $O_j'^1$, and $O_j'^2$, $1 \leq j \leq m$, we conclude that for assembling the sub-pattern $\mathit{val}(c)$ no new tile types coloured Cl_j, "1", or LB are defined. Thus, it means that the previous truth assignment of the variables $x_1, \ldots, x_n$ (uniquely identified by the West glues of the n tiles coloured "1") validates the formula C (contradiction). □

Corollary 4. *The Fixed-*`N` *PATS problem is* NP*-complete.*

Indeed, let us assume that we can construct a polynomial algorithm $\mathcal{P}$ solving the Fixed-`N` PATS problem. Consider now an instance of a 3SAT problem using n variables and m clauses. We generate the patterns $\mathit{patt}(c)$, $\mathit{assgn}(c)$ and $\mathit{val}(c)$ and use the algorithm $\mathcal{P}$ to verify whether the pattern $\mathit{patt}(c) \cdot \mathit{assgn}(c) \cdot \mathit{val}(c)$ can be assembled using a coloured TAS with exactly $\mathtt{N} = 2+20n+19m+n^2$ different tile types. By the above theorem, we can thus decide using $\mathcal{P}$ whether the formula C is satisfiable or not.

Corollary 5. *The optimization problem for PATS is* NP*-hard.*

Note: In the above NP-hardness proof of the PATS problem we have employed a reduction from 3SAT.

In our reduction from 3SAT proving the NP-hardness of PATS we employ a construction which utilizes a number of colours which is linear in the size of the logic formula; more precise it is linear in the number of variables and of clauses of the CNF formula. In a recent breakthrough, Seki [17] proposes an improvement of the result, showing that a 3SAT reduction is possible in which the constructed tile pattern uses only a constant number of colours, independent of the size of the CNF formula.

4 Families of Patterns of Unbounded Assembly Complexities

In the following we study the assembly complexity of various patterns within the PATS framework, that is, when the assembly proceeds from an "L"-shaped seed structure, which fixes the borders of the pattern. In particular, we are pursuing the task of finding some families of patterns with an unbounded assembly complexity. Note that because we assume the assembly starts from an "L"-shaped seed, one can construct a variety of complex patterns (and families of patterns) using only a constant number of tiles. For example, let us consider the 'black" $n \times n$-square pattern, for any $n \geq 1$. If starting from a one-tile seed structure, it was shown in [16] (and proved to be asymptotically tight in [1]) that the assembly complexity of this pattern is in $\Omega(\frac{\log n}{\log \log n})$. However, if we are allowed to start from an "L"-shaped seed structure which fixes the border of the pattern, the assembly complexity becomes trivially equal to 1. Other patterns which are assembled with constant number of tiles are the Sierpinski pattern and the binary-counter pattern, see Figure 8. Thus, the problem of finding a family of patterns of unbounded assembly complexity (and using only a bounded number of colours) is not immediate.

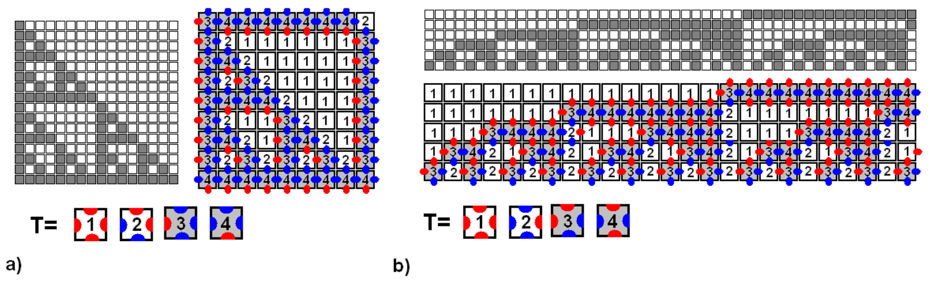


Fig. 8. a) The Sierpinski pattern and a tile set assembling it; b) The binary-counter pattern and a tile set assembling it. In both cases the assembly starts from an appropriate L-seed structure.

Given a pattern $\mathcal{A}$, we say that it has *assembly complexity*[4] η if the minimal TAS assembling $\mathcal{A}$ and obeying the constraints $P1, P2, P3$[5] from Definition 1 (of the PATS problem), contains η different tile types; we denote by $\eta_{\mathcal{A}}$ the tile assembly complexity of the pattern $\mathcal{A}$.

Given two increasing functions $F_w, F_l : \mathbb{N} \to \mathbb{N}$, we define a *family of patterns* $(\mathcal{A}_n)_{n\geq 1}$ as a sequence where for all $n \geq 1$ there exist numbers $(k_n)_{n\geq 1}$, with $k_n \geq 1$ (for all $n \geq 1$) and $k_n \leq k_{n+1}$, such that $\mathcal{A}_n : [F_w(n)] \times [F_l(n)] \to [k_n]$ is a pattern. If $k_n = k$ for all $n \geq 1$, we say that $(\mathcal{A}_n)_{n\geq 1}$ is a *family of $k-$coloured patterns*. Moreover, if for all $n \geq 1$, $\mathcal{A}_n(w, l) = \mathcal{A}_{n+1}(w, l)$ for all $(w, l) \in [F_w(n)] \times [F_l(n)]$ (i.e., the pattern $\mathcal{A}_n$ is embedded into the down-left corner of the pattern $\mathcal{A}_{n+1}$), we say that $(\mathcal{A}_n)_{n\geq 1}$ is an *embedded family of patterns*. Given a family of patterns $(\mathcal{A}_n)_{n\geq 1}$ we say that it has *unbounded assembly complexity* if for every $n \geq 1$ there exists some $p \geq 1$ such that $\eta_{\mathcal{A}_n} < \eta_{\mathcal{A}_{n+p}}$. Otherwise, we say that $(\mathcal{A}_n)_{n\geq 1}$ has *bounded assembly complexity*. In the last case, we denote by $\eta_{(\mathcal{A}_n)_{n\geq 1}} = \max\{\eta_{\mathcal{A}_n}, n \geq 1\}$ the maximal assembly complexity of all the patterns in the family $(\mathcal{A}_n)_{n\geq 1}$.

The problem of finding a family of patterns of unbounded assembly complexity can be solved immediately if one does not bound the number of colours. Indeed, we can define an embedded family of patterns in which between any two consecutive element of the family we add at least one position which is covered by a new colour. Thus, from now on, we consider only the case of families of patterns over a bounded number of colours.

[4] We differentiate from the classical notion of *tile complexity* in order to underline the difference between the pre-requirements of the two definitions.

[5] One can summarize these constrains as: P1. Glues strength one for all glues; P2. Seed-assembly of the form $[0, m] \times \{0\} \cup \{0\} \times [0, n]$; P3. The TAS together with a colouring function assembles the pattern.

Family of patterns of fixed height. For any value $k \geq 2$, define the strip of height k as the subspace $S^k = \mathbb{N} \times [k]$. We restrict in the following to families of patterns included into some strip S^k, and we consider first the case of S^2. We call a pattern *binary*, if it consists of only two colours.

Theorem 2. *Any family of binary patterns included into the S^2 strip has bounded assembly complexity. In particular, the assembly complexity is at most 4.*

On the other hand, assume now that for some strip-2 binary pattern $\mathcal{A} : [m] \times [2] \to \{0, 1\}$ we can find included within all the following four sub-patterns $p_{(i,j)} : [2] \times [2] \to \{0, 1\}$, with $0 \leq i, j \leq 1$ and $p_{(i,j)}(2, 1) = p_{(i,j)}(1, 2) = i$ and $p_{(i,j)}(2, 2) = j$. Then, by Proposition 2, any tile system $\mathcal{T}$ which assembles $\mathcal{A}$ deterministically must contain at least two tile types coloured "0" and two tile types coloured "1". Thus, $\eta_{\mathcal{A}} \geq 4$ and by Theorem 2, we get $\eta_{\mathcal{A}} = 4$.

Theorem 2 can be generalized both in terms of the number of colours and in terms of the height of the strip.

Proposition 9. ***(i)*** *For every $k \geq 2$ and any S^2 pattern $\mathcal{A} : [m] \times [2] \to [k]$, we have that $\eta_{\mathcal{A}} \leq k^2$. Moreover, there exists a S^2 pattern $\mathcal{A}_k^2$ such that $\eta_{\mathcal{A}_k^2} = k^2$.*
(ii) *For every $k, p \geq 2$ and any S^p pattern $\mathcal{A} : [m] \times [p] \to [k]$, we have $\eta_{\mathcal{A}} \leq k^p$.*

Based on the previous result, we conclude that if we restrict to some S^k domain, as long as we bound the number of possible colours appearing into the pattern, we cannot construct a family of patterns of unbounded assembly complexity.

Theorem 3. *For every $k, p \geq 2$, there exists no family of k-colour patterns $(\mathcal{A}_n)_{n\geq 1}$ included into the S^p strip which is of unbounded assembly complexity.*

The result of Proposition 9 upper bounds the assembly complexity of any S^p k-coloured pattern. However, it does not say whether this bound is tight, as is the situation in the $p = 2$ case. In the following we show that there exist patterns whose assembly complexity gets close to the described upper bound.

Proposition 10. ***(i)*** *For every $p \geq 2$, there exists a S^p binary pattern $park_p^2$ such that $\eta_{park_p^2} \geq 2\lceil 2^{(p-1)/2} \rceil$.*

(ii) *For every $k, p \geq 2$ there exists a S^p k-colour pattern $park_p^k$ such that $\eta_{park_p^k} \geq k\lceil k^{(p-1)/2} \rceil$.*

Families of Patterns of Unbounded Assembly Complexity. Based on the patterns $park_p^k$ we can now describe two families of patterns (the second being an embedded family) which have an unbounded assembly complexity.

Example 1. For any $n \geq 1$ define $\mathcal{A}_n$ as $park_n^k$ above. By Proposition 10, $(\mathcal{A}_n)_{n\geq 1}$ is a k-colour family of patterns of unbounded assembly complexity.

For defining an embedded family of unbounded complexity, we need first to define a stacking operation for patterns. Given two patterns $\mathcal{A}_1 : [l_1] \times [h_1] \to K_1$ and

$\mathcal{A}_2 : [l_2] \times [h_2] \to K_2$, and some background (arbitrary) colour "0", we define the stacking of the two patterns $\mathcal{A}_2 \diagup \mathcal{A}_1$ (with background "0") as the pattern $\mathcal{A}_2 \diagup \mathcal{A}_1 : [l] \times [w_1 + w_2] \to K_1 \cup K_2 \cup \{0\}$, where $l = \max(l_1, l_2)$, and

- $\mathcal{A}_2 \diagup \mathcal{A}_1(i, j) = \mathcal{A}_1(i, j)$ for all $1 \le i \le l_1$, $1 \le j \le h_1$,
- $\mathcal{A}_2 \diagup \mathcal{A}_1(i, h_1 + j) = \mathcal{A}_2(i, j)$ for all $1 \le i \le l_2$, $1 \le j \le h_2$,
- $\mathcal{A}_2 \diagup \mathcal{A}_1(i, j) = 0$ for all remaining positions.

Example 2. For $n \ge 2$ we define the family of patterns $(\mathcal{A}_n)_{n \ge 1}$ inductively as follows: $\mathcal{A}_2 = \mathit{park}_2^2$; $\mathcal{A}_{i+1} = \mathit{park}_{i+1}^2 \diagup \mathcal{A}_i$. By its definition, $(\mathcal{A}_n)_{n \ge 1}$ is indeed an embedded family of (binary) patterns. Moreover, by Proposition 10 and the previous example, we have that $(\mathcal{A}_n)_{n \ge 1}$ is an embedded family of binary patterns of unbounded assembly complexity.

Acknowledgments. We thank Prof. Pekka Orponen and Dr. Shinnosuke Seki for their useful discussions and valuable comments in the development process of our results. Also, we thank the anonymous referee for their very helpful suggestions.

References

[1] Adleman, L., Cheng, Q., Goel, A., Huang, M.: Running time and program size for self-assembled squares. In: STOC 2001: Proc. 32nd Annual ACM Symp. on Theory of Computing (2001)
[2] Czeizler, E., Lempiäinen, T., Orponen, P.: A design framework for carbon nanotube circuits affixed on DNA origami tiles. In: Proc. FNANO 2011 (2011) (Poster)
[3] Douglas, S., Bachelet, I., Church, G.: A Logic-Gated Nanorobot for Targeted Transport of Molecular Payloads. Science 335(6070) (2012)
[4] Göös, M., Orponen, P.: Synthesizing Minimal Tile Sets for Patterned DNA Self-assembly. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 71–82. Springer, Heidelberg (2011)
[5] Kuzyk, A., Laitinen, K., Törmä, P.: DNA origami as a nanoscale template for protein assembly. Nanotechnology 20(23) (2009)
[6] Lempiäinen, T., Czeizler, E., Orponen, P.: Synthesizing Small and Reliable Tile Sets for Patterned DNA Self-assembly. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 145–159. Springer, Heidelberg (2011)
[7] Liu, W., Zhong, H., Wang, R., Seeman, N.C.: Crystalline two-dimensional DNAorigami arrays. Angew. Chem. Int. Ed. 50(1) (2011)
[8] Lund, K., et al.: Molecular robots guided by prescriptive landscapes. Nature 465 (2010)
[9] Ma, X., Lombardi, F.: Synthesis of tile sets for DNA self-assembly. IEEE Trans. on Computer-Aided Design of Integrated Circuits 27 (2008)
[10] Ma, X., Lombardi, F.: On the computational complexity of tile set synthesis for DNA self-assembly. IEEE Trans. Circuits and Systems II: Express Briefs 56 (2009)
[11] Demaine, E.D., Eisenstat, S., Ishaque, M., Winslow, A.: One-Dimensional Staged Self-assembly. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 100–114. Springer, Heidelberg (2011)

[12] Maune, H.T., Han, S., Barish, R.D., Bockrath, M., Rothemund, P.W.K., Winfree, E.: Self-assembly of carbon nanotubes into two-dimensional geometries using DNA origami templates. Nature Nanotechnology 5 (2010)

[13] Modi, S., Bhatia, D., Simmel, F.C., Krishnan, Y.: Structural DNA Nanotechnology: From bases to bricks, from structure to function. J. Phys. Chem. Lett. 1 (2010)

[14] Rajendran, A., Endo, M., Katsuda, Y., Hidaka, K., Sugiyama, H.: Programmed twodimensional self-assembly of multiple DNA origami jigsaw pieces. ACS Nano 5(1) (2011)

[15] Rothemund, P.W.K.: Folding DNA to create nanoscale shapes and patterns. Nature 440 (2006)

[16] Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In: STOC 2000: Proc. 32nd Annual ACM Symp. on Theory of Computing (2000)

[17] Seki, S.: Combinatorial optimizations in pattern assembly (manuscript)

[18] Schulman, R., Winfree, E.: Synthesis of Crystals with a Programmable Barrier to Nucleation. Proc. Nat. Ac. Sci. 104 (2007)

[19] Schulman, R., Yurke, B., Winfree, E.: Robust self-replication of combinatorial information via crystal growth and scission. Proc. Nat. Ac. Sci. 109 (2012)

[20] Yan, H., Park, S.H., Finkelstein, G., Reif, J.H., LaBean, T.H.: DNA-templated self-assembly of protein arrays and highly conducive nanowires. Science 301 (2003)

[21] Zhao, Z., Yan, H., Liu, Y.: A route to scale up DNA origami using DNA tiles as folding staples. Angw. Chem. Int. Ed. 49(8) (2010)

A Geometric Approach to Gibbs Energy Landscapes and Optimal DNA Codeword Design

Max H. Garzon and Kiran C. Bobba

Computer Science, the University of Memphis, TN 38152, U.S.A.
{mgarzon,kbobba}@memphis.edu

Abstract. Finding a large set of single DNA strands that do not crosshybridize to themselves or to their complements (so-called *domains* in the language of chemical reaction networks (CRNs)) is an important problem in DNA computing, self-assembly, DNA memories and phylogenetic analyses because of their error correction and prevention properties. In prior work, we have provided a theoretical framework to analyze this problem and showed that Codeword Design is **NP**-complete using any single reasonable *metric* that approximates the Gibbs energy, thus practically excluding the possibility of finding any procedure to find maximal sets exactly and efficiently. In this framework, codeword design is reduced to finding large sets of strands maximally separated in DNA spaces and, therefore, the size of such sets depends on the geometry of these spaces. Here, we introduce a new general technique to embed them in Euclidean spaces in such a way that oligos with high/low hybridization affinity are mapped to neighboring/remote points in a geometric lattice, respectively. This embedding materializes long-held mataphors about codeword design in terms of sphere packing and leads to designs that are in some cases known to be provable nearly optimal for some oligo sizes. It also leads to upper and lower bounds on estimates of the size of optimal codes of size up to 32−mers, as well as to infinite families of DNA strand lengths, based on estimates of the kissing (or contact) number for sphere packings in Euclidean spaces. Conversely, we show how solutions to DNA codeword design obtained by experimental or other means can also provide solutions to difficult spherical packing geometric problems via this embedding. Finally, the reduction suggests an analytical tool to arrange the dynamics of strand displacement cascades in CRNs to effect the transformation through bounded Gibbs energy changes, and thus is potentially useful in compilers for wet tube implementation of biomolecular programs.

Keywords: Gibbs energy, DNA space, molecular design, spherical codes, noncrosshybridizing oligonucleotide bases, molecular compiler.

1 Introduction

DNA computing has brought forth the important theoretical problem of deep understanding of the thermodynamics hybridization for a variety of applications, such as self-assembly (Qian & Winfree, 2011; Seeman, 2006), natural language processing (Garzon et al., 2009; Neel at al., 2006; Bobba et al., 2006) and DNA-based memories (Neel & Garzon, 2008) and, more recently, biological phylogenies based purely on whole-genomic DNA (Garzon & Wong, 2011). Of primary importance in all these applications is the appropriate identification of DNA molecular ensembles that encode inputs to computational problems or serve as building blocks for the appropriate nanostructures in order to guarantee that the desirable reactions take place as intended, amidst the tendency of DNA molecules to form other structures due to the uncertainty and variability inherent in hybridization affinity. This so-called *Codeword Design* problem has seen some progress in the last decade in at least two subareas. First, in searching and/or building such DNA code sets (Garzon et al., 2009; Deaton et al., 2006; Tulpan et al., 2005; Chen et al., 2006), in which the size of feasible computational problems or self-assembled nanostructures is usually directly related to the largest ensemble of DNA molecules that satisfy a given set of cross-hybridization and noncrosshybrization constraints. The second area deals with developing the appropriate theoretical framework to understand this type of problems, find solutions, and organize the knowledge about the subject in a systematic manner.

Solutions to finding large sets of short oligonucleotides with noncrosshybridizing (nxh) properties have been produced by several groups, particularly the PCR Selection (PCRS) protocol used below (Garzon et al., 2009; Deaton et al., 2006; Tulpan et al., 2005; Chen et al., 2006). In prior work (Phan & Garzon, 2009), we have also provided a theoretical framework to analyze this problem. In this framework, codeword design is reduced to finding large sets of strands maximally separated in the DNA space of all oligos of a given size. The size of such sets thus depends on the geometry of these DNA spaces. However, we showed therein that Codeword Design is **NP**-complete using any single reasonable measure that approximates the Gibbs energy by any reasonable metric that would provide such a nice geometry to analyze DNA spaces, thus practically excluding the possibility of finding any procedure to find truly maximal sets exactly and efficiently through such an approach. In this paper, we introduce a new general technique to bring the problem into the more geometric context of ordinary Euclidean spaces, where ordinary intuition may provide better insights into the nature and characteristics of DNA spaces. Specifically, we describe in Section 3 a way to embed DNA spaces in Euclidean spaces and thus, among others, reduce the word design problem to the well known sphere packing problem in ordinary geometry. The embedding sheds some insights into the geometry of DNA spaces via the best known metric approximation of Gibbs energy landscapes, namely the $h-$distance , which is briefly described in Section 2.1. In Section 2.2, we show that this metric can be extended to give a provably close approximation of the Gibbs energies of duplex formation, while preserving the metric property (i.e., isometrically.) From this embedding, we develop a technique to obtain nearly optimal codeword sets for short oligomer spaces (up to 12-mers or so) in Section 3. The quality of these sets is evaluated against the only general purpose method to produce maximal codes, the PCR Selection protocol, using a well accepted model of the Gibbs energy of duplex formation, namely the nearest neighbor model. In the conclusion in Section 4, other potential applications of the embedding are discussed to gain further understanding of the Gibbs energy landscapes for DNA spaces.

2 Optimal DNA Codes, Gibbs Energies and DNA Spaces

The cardinality of maximal codes for given parameters has been previously established only for simple models (King, 2003; Marathe, 1999) but were largely unknown under more realistic measures of hybridization. Attempts to construct non-crosshybridizing sets *in silico* using various methods such as templates (Arita et al. 2002) or heuristic search (Shortreed et al., 2005; Tulpan et al., 2005) tended to produce relatively small sets. Other methods to create non-crosshybridizing sets (Chen et al., 2004) in vitro are attractive for being able to produce a maximal physical set, but suffer from the challenge of identifying precisely its size and, more importantly, the composition of the sequences. More recently, nearly optimal codes have explicitly obtained by simulations of the PCR Selection (PCRS) protocol (Garzon et al., 2005; Garzon et al., 2004) for short length oligonculetoides (up to 13−mers) and for longer lengths by a so-called shuffle operation to generate longer codes from smaller ones (Garzon et al., 2009). No general, scalable, analytic and efficient method exists that is capable of producing the explicit sequences in DNA codeword sets, to the best of our knowledge. One such method is described in the following sections using a model equivalent to the one described in prior work (Phan & Garzon, 2009; Garzon et al., 1997).

2.1 Gibbs Energies

DNA duplex formation of two single strands is determined by the familiar Gibbs energy in thermodynamics and physics, which generally depends on physical parameters such as the internal energy (U), pressure (p), volume (V), temperature (T), and entropy (S) of the environment in which the duplex is formed. The Gibbs energy is the chemical equivalent of the potential energy in physics. The more negative the Gibbs energy, the more stable the duplex formed. Unfortunately, models of biochemistry are inherently approximations and so no gold standard exists to assess Gibbs energies other than accepted empirical approximations (Wetmur, 1997), although it is known that the energy depends not only on the nucleotide composition of the strands but also on their arrangement, in addition to other thermodynamical factors (temperature, pressure, acidity of the solvent, etc.).

On the other hand, knowledge of the corresponding Gibbs energy landscapes would appear critical in an analysis of the codeword design problem. The most popular method to approximate the Gibbs energy is the so-called nearest-neighbor (NN hereafter) method, which postulates that the free energy for duplex formation basically depends on three factors: the initial free energy given by the unfavorable entropy due to the loss of translational freedom, the sum of the complementary pair-wise terms between the nucleotides sequences being stacked to form the duplex, and the entropic penalty of the maintenance of the symmetry in a double strand. To date, several different experimental thermodynamic parameters for the 10 'nearest-neighbor' di-nucleotide sequences, namely dAA/dTT, dAT/dTA, dTA/dAT, dCA/dGT, dGT/dCA, dCT/dGA, dGA/dCT, dCG/dGC, dGC/dCG, dGG/dCC, have been measured experimentally (Huguet et al., 2010; SantaLucia, 1998). A full use of even the NN model to analyze the codeword design problem forces the examination of exponentially many configurations in the minimization of the energy and thus it is computationally intractable, where we need to deal in principle with 2^{4^n} possible subsets of the entire DNA space to search for the code set of the largest size.

2.2 Combinatorial Approximations to the Gibbs Energy

In this paper we will therefore use a different method, using the $h-$distance introduced in (Garzon et al., 1997) as a rougher but also more tractable and close approximation of the Gibbs energy. Briefly, the relevant chemistry is abstracted as two operations of interest for our purpose, a unary operation of Watson-Crick complementation and a binary operation of hybridization. The Watson-Crick complement y' of strand y is obtained by reversing it and swapping nucleotides within the pairs a, t and c, g. Hybridization is modeled as a binary operation between two single strands x, y that forms a double-stranded molecule in which pairs of nucleotides are joined in a duplex molecule by two or three hydrogen bonds, for an $a-t$ or a $c-g$ pair, respectively. The distance is defined in terms of the h-measure between two single strands x, y as follows.

$$h(x, y) := \min_{-n<k<n} \left\{|k| + H(x, \sigma^k(y'))\right\} \tag{1}$$

where $\sigma^k(y)$ is the shift of y by k positions (right-shift if $k > 0$; left-shift if $k < 0$) with respect to x, y' is the Watson-Crick complement of y, and the Hamming distance H measures the number of different bases in the overlap of x and y' in the specified frameshift $\sigma^k(y')$. In words, the h-measure finds *the optimal alignment in which x and y have the maximum number of complementary pairs, thus forming the most stable duplex.* A measure of $h(x, y) = 0$ means $x = y'$. A large distance indicates that even when x finds itself in the proximity of y, they contain few complementary base pairs regardless of the position they find themselves in, and thus are unlikely to form a stable duplex, i.e. no hybridization will take place. For example, if $x = agc$, $y = tgg$ (and $y' = cca$), at shift $k = -2$, there are 2 base pair mismatches and one identity match, so $2 + H(a, a) = 2$; at shift $k = -1$, there are 3 pair mismatches $1 + H(ag, ca) = 3$; At shift 0 (perfect alignment) there are 3 mismatches $H(agc, cca) = 3$; at shift 1, the number of mismatches is $1 + H(gc, cc) = 2$; and at shift 2, the distance is $2 + H(c, c) = 2$. Therefore, $h(agc, ttg) = 2$. Note that h is invariant under Watson-Crick complementation, as well as under both reversal and pointwise complementation of each nucleotide, i.e., for every pair of strands $x, y \in \mathbf{D}_n$,

$$h(x, y) = h(x', y') = h(x^r, y^r) = h(x^c, y^c)$$

From an analytic point of view, it is desirable to have a measure of hybridization that can be thought of as some sort of distance, intuitively analogous to the ordinary distance in Euclidean spaces. Unfortunately, the h-measure does not possess the requisite properties to make $\mathbf{D}_n$ a so-called *metric space.* Even though the h-measure possesses the properties of non-negativity, it misses the triangle inequality (since x can be chosen so that $0 < h(x, x)$ but $0 = h(x, x') + h(x', x)$.) This measure becomes a true distance function, however, if we bundle together complementary strands into a so-called *poligos* $X = \{x, x'\}$ and measure the distance between two poligos as

$$h(X, Y) = \min_{x \in X, y \in Y} h(x, y)\,. \tag{2}$$

It has been established in (Phan & Garzon, 2009) that this function truly defines a metric in $\mathbf{D}_n$. This metric model reduces codeword design to a well known problem that has been researched for over half a century, namely the design of communication codes in information theory (Roman, 1995). Although the analogy has proved useful in using error-correcting codes in information theory to produce DNA codes (Arita et al., 2003, Garzon et al., 1997), the fact that the $h-$distance is very different in nature from the Hamming distance (as can be seen by perusing eq. (1) and the consequent properties below) makes the analogy only a metaphor. Therefore, the h-distance quantifies hybridization afinity more closely while preserving the advantages of a metric space structure in $\mathbf{D}_n$, the space of all $n-$mers of given length n (Phan & Garzon, 2009).

For the purpose of this paper, we carry this reduction a step further by building a modified version of the NN model described above, using the values in Table 1 as multipliers for the corresponding counts of mismatches in the optimal alignment that produced the standard $h-$distance $h(x, y)$ between strands x and y. The weights were normalized by the standard partition function (divide every weight by the total sum of all weights involved) to obtain weights ranging from 0 to 1 for all the possible nucleotide mismatches, as shown in the first column of Table 1. Now, the proof in (Phan & Garzon, 2009) is easily seen to carry over to this modification of the $h-$distance so that the normalized $h-$distance remains a metric. (We will continue to refer to this normalized version as the $h-$distance in the reminder of this paper.)

That **this normalized $h-$distance comes close enough for an approximation of the Gibbs energy** can be seen as follows. A genetic algorithm was employed to generate the weights with two fitness functions: accuracy and RMSE (statistical root mean square error) over all $n-$mers for code sets in the column headers in Table 1. The *accuracy* is defined as the ratio of the weighted $h-$distance s that are within a certain range of their Gibbs energy (as given by the NN model) to the total number of possible Gibbs energies for that set. The range that was used for deciding whether the weighted $h-$distance was close to Gibbs energy or not was calculated with a parabolic function that will make sure that difference should be minimal in the critical region about -6.0 Kcal/mole. A function of $\epsilon + (\Delta G - 0.5)^2$ was the threshold for the difference between the normalized weighted $h-$distance and Gibbs energy ΔG. The $RMSE$ fitness is defined as $1 - RMSE$ of the differences over the same distribution. The results are uniformly very good since they are nearly identical for all weight sets and are summarized in Fig 1.

Table 1. Weights for the corresponding counts of mismatches in the optimal alignment that normalizes the $h-$distance into the interval $[0, 1]$ so that hybridization between strands occurs when $h(x, y) < 0.5$ (equivalent to $\Delta G < -6.0$ Kcal/mole.) The weights were obtained by search through a genetic algorithm that minimizes the RMSE difference between a linear combination (with the weights) of the number of occurrences of the di-nucleotides mismatches in optimal alignments for the $h-$distance and the (normalized) NN Gibbs energies (at 20^oC) across a representative sample of all strands. Naturally, the weights depend on the temperature at which the NN Gibbs energy is computed, but analogous weights can be similarly obtained for other temperatures.

Mismatches	6−mers (100%)	6−mers (40%)	7−mers (1%)	8−mers (1%)
AA	2.651	2.620	2.616	2.443
AC/CA	2.631	2.605	2.485	2.417
AG/GA	2.602	2.631	2.560	2.516
A-/-A	1.328	1.307	1.257	1.011
CC	2.636	2.562	2.532	2.347
CT/TC	2.647	2.629	2.558	2.476
C-/-C	1.334	1.317	1.280	1.040
TT	2.642	2.636	2.586	2.672
TG/GT	2.639	2.633	2.545	2.458
T-/-T	1.258	1.335	1.274	1.226
GG	2.560	2.632	2.535	2.321
G-/-G	1.315	1.316	1.268	1.437

To facilitate the comparison of the $h-$distance to the NN Gibbs energy, both values are rescaled to the unit interval as follows. Normalization of the $h-$distance was calculated as a ratio of the sum of the weights for all the mismatches in the best alignment in computing the $h-$distance to the sum over the maximum possible number of mismacthes,. i.e., $2n - 1$, where n is the size of the words in the code set. On the oher hand, Gibbs energies of a DNA code set were normalized by a piecewise linear function so that the minimum Gibbs energy of the set maps to 0, -6.0 Kcal/mole maps to 0.5 and the maximum Gibbs energy of the set maps to 1. As shown in Fig 1, these two independent fitnesses show that over 80% of the two agree across all oligo sets tested for the purpose of deciding whether or not a hybridization event occurs (Gibbs energies were computed at 20^oC.) Further, the uniformity of the results shown in Table 1 show that this result is likely to scale for larger oligos as far as the NN model holds, albeit with descresing values and slightly lower performance. Further evidence of the apropriateness of the $h-$distance as an approximation can be obtained by comparison of results in optimal code sets obtained independently by both methods. These results can be constrated in columns 3 and 4 of Table 2, although a more complete set of results up to 20−mers is not yet available. A more principled argument can be given from the definitions of the Gibbs energy in the NN model and the $h-$distance , since both are minimizing over all possible frameshifts, under the hypothesis of stiffness mentioned below, but the argument is too long for the space allowed and will be reported elsewhere.

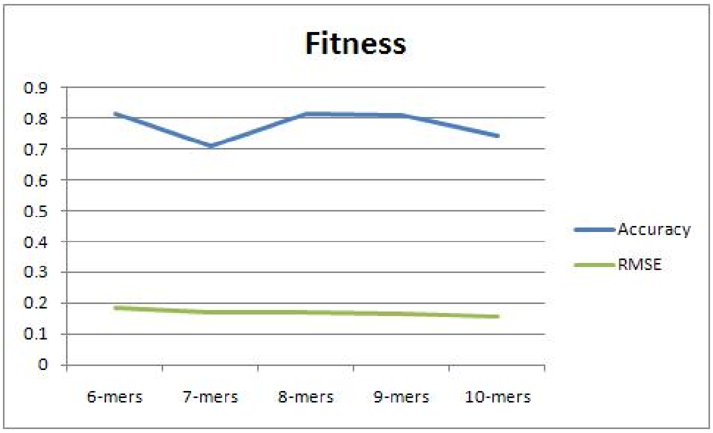


Fig. 1. Accuracy of the normalized $h-$distance in predicting hybridization events, as compared to a Gibbs energy model of duplex formation (nearest neighbor model.) The accuracy is measured by the percentage of hits ($h(x, y) < 0.5$ exactly when $\Delta G(x, y) < -6.0$ Kcal/mole) and the Root Mean Square Error (RMSE) of the difference of the two, in both cases over the entire DNA space.

The $h-$distance thus provides a more computationally efficient approximation of the Gibbs energy based solely on composition and sequence. Although the $h-$distance makes the assumption that DNA oligos are stiff and do not form bulges or hairpins, this is a mild assumption for short oligos up to 16$-$mers or so because pre-processing can filter out in polynomial time strands not satisfying this condition (such as Watson-Crick palindromes.) On the other hand, the $h-$distance allows a fine control of hybridization stringency, considers hybridization in all possible frameshifts, and is therefore on some sense more realistic than the simpler models mentioned above. As such, the $h-$distance may afford good insights into the structure of DNA spaces and reasonable avenues into the codeword design problem, as further evidenced by the results below.

With this degree of soundness of the $h-$distance approximation for the Gibbs energy in place, we can proceed to the discussion of the algorithm for nearly optimal code design using the $h-$distance .

3 Optimal Codes from Geometric Packing

The second phase in the reduction of the codeword design problem to a sphere packing problem proceeds in three stages. First, we show that it suffices to solve the problem for the $h-$distance approximation, as discussed in Section 2. Second, we will show how to map DNA spaces into ordinary Euclidean spaces so that their Gibbs energy landscapes are basically converted into more intuitive and analyzable geometric objects. As a result, the codeword design problem is reduced to an age-old problem in geometry, originated in Newton's problem of the 12 spheres (conways & Sloane, 1999, p. 21). Third, a number of solutions to this problem are shown to provide answers that are nearly optimal given the state of the art of the geometric problem. For space reasons, in this section we will be illustrate with a particular case, the so-called h_0-distance defined as above but without considering frameshifts, i.e.,

$$h_0(x,y) := \min\{H(x,y), H(x,y')\} \tag{3}$$

The technique scales to the full normalized $h-$distance (Phan & Garzon, 2009), but the technical details are more involved to explain than the space allowed here -see (Phan & Garzon, 2009) for more details.

For the first stage in the reduction, we find a representation of DNA strands $x \in \mathbf{D}_n$ as vectors in the Euclidean space $\mathbf{R}^{4n}$ of dimension $4n$ by coarse coding them as follows. Given a basis $b \in \{a, c, g, t\}$, let $\Psi_b(x) \in \mathbf{R}^{4n}$ be the vector given by $\Psi_b(x)_i = \psi(b, x_i) = 1$ if $b = x_i$ and 0 otherwise, i.e., $\Psi_b(x)_i$ indicates the presence or absence of nucleotide b in position i in x. The representation Ψ of a strand x is given by the concatenation of these vectors in the following order

$$\Psi(x) = \Psi_a(x)\Psi_c(x)\Psi_g(x)\Psi_t(x)\,. \tag{4}$$

The following properties of Ψ are easy to verify for every pair of strands x, y, where $||\Psi(x)||$ is the standard Euclidean norm of a vector, "$\cdot$" denotes the standard dot (scalar) product between two vectors in $\mathbf{R}^{4n}$, and $|x|$ denotes the length of strand x:

1. Strands of length n are represented as points on a sphere of radius $\sqrt{n}$, i.e.
$$||\Psi(x)||^2 = |x| = n \text{ and } ||\Psi(x)|| = \sqrt{n}\,;$$
2. The number of Watson-Crick complementary matches between x and y in perfect alignment is $m(x,y) = \Psi(x)\cdot\Psi(y)$. Therefore
$$H(x,y) = n - \Psi(x)\cdot\Psi(y)\,;$$
3. The Hamming distance can be normalized as
$$\frac{1}{n}H(x,y) = 1 - \frac{1}{\sqrt{n}}\Psi(x)\cdot\frac{1}{\sqrt{n}}\Psi(y) \tag{5}$$
$$= 1 - \cos\theta(\Psi(x),\Psi(y)) = 2\sin^2(\theta/2)\,. \tag{6}$$
where θ is the angle between the two $4n$D vectors $\Psi(x), \Psi(y)$. This metric essentially is equivalent to H and has the advantage that it represents DNA strands as points on the unit sphere in the Euclidean space of dimension $4n$. From now on, we will identify a DNA oligo x with its representation $\Psi(x)$.
4. Complementation is reduced to a reversal operation, i.e.,
$$\Psi(y') = \Psi(y)^r = R(\Psi(y))\,,$$

where R is a linear transformation reversing the coordinates of a vector in $\mathbf{R}^{4n}$. This operation can in fact be shown to be a *reflection* about a subspace of dimension $2n$. For example, when $n = 1$, the bases $a \equiv 1000, c \equiv 0100$ can be obtained by reversal of the vectors for the bases $t \equiv 0001, g \equiv 0010$, respectively, in 4D-space. This linear transformation has two eigenvalues ± 1, each of multiplicity 2, with eigenvectors $1001, 0110$ (left invariant for being palindromes) and $\bar{1}001, 0\bar{1}10$, respectively ($\bar{1} \equiv -1$.) The complement of strand x is thus obtained by image reflection about the fixed 2D plane generated by the first two eigenvalues (hereforth called the *mirror*) and amounts to a change in sign of the other two components in the orthogonal coordinate system consisting of these eigenvalues. Longer strands can then be represented by adding sets of 4-tuples for the additional bases without changing these procedures to find the complementary strands by reflection (about a higher dimensional subspace of dimension $2n$, of course.)

5. The normalized h_0-distance is then computed by just the simple normalized Hamming distance *provided* we ensure that the two vectors x, y lie on the "same side" of the mirror (the invariant subspace), i.e.,

$$\frac{1}{n} h_0(x, y) = 2 \sin^2(\theta/2)\,. \tag{7}$$

Therefore, maximal DNA codeword sets of $n-$mers with a noncrosshybridizing quality given by a minimum separating h_0-distance of τ (now normalized after division by $\sqrt{n}$) is reduced to an age-old packing problem in ordinary geometry going back to Newton and Gauss in the 1600-1800s. This is the so-called **spherical packing problem**, analogous to but essentially different from the ordinary sphere packing problem in geometry and classical coding theory (Conway & Sloane, 1999, p.24). This problem is challenging and far from resolved, but a wealth of knowledge has been gathered over the centuries, which can now be used to help answer critical questions about the DNA codeword design problem. For example, the well-known *kissing number* problem in nD-Euclidean space asks for the maximum number of congruent spheres that can be arranged to touch a given one. We are actually concerned here with the analogous kissing number on the surface of the unit sphere Ω_{4n} in the same $4n$D space, denoted $A^*(n, \phi)$, where $2\sin^2(\phi/2) \geq \tau$ and $\tau > 0$ is the parameter coding for quality in the codeword set (fault-tolerance to reaction conditions, etc.). Resolving this inequality gives a minimum angle $\phi_0 = 2\arcsin(\sqrt{\tau/2})$ for the minimum separation between DNA points on Ω_{4n} for $\tau < 1$. As a second example, known solutions to the spherical code problem (usually called *spherical codes*, i.e., a set of points $\mathcal{C} \subseteq \Omega_{4n}$ satisfying a minimal separation distance as measured by the angle $\phi = x \cdot y$ between two points/vectors x, y) can be used to obtain sets of noncrosshybridizing DNA codeword of nearly the same quality as the centers of the kissing spheres in Ω_{4n} by selecting vectors representing $n-$mers that come close enough to points in $\mathcal{C}$.

The optimality and/or quality of the solutions to the kissing spheres problem is usually inversely related to the simplicity in generating them. The best known solutions are afforded by so-called *lattice packings*, obtained by diophantine (integer coefficients) combinations of n linearly independent vectors (usually assumed to be diophantine as well.) We illustrate the third stage in our method to obtain a DNA codeword set with the optimal solution for lattice packings in 24D-space (i.e., for 6−mers), the *Leech lattice*, which can be generated from Golay's [24,12] 3-error correcting code in 24D Hamming space and is known to have a separation angle of arccos(1/3) $\approx 70°$ (Conway & Sloane, 1999, p. 27). The basic ideas is to decompose the lattice point in 4D-space into nearly concentric layers (usually referred to as *shells*) around one of its points located as origin (perhaps after a translation.) The points in this shell are then projected onto Ω_{4n} by dividing by their norm. They provide an approximate solution to the sphere kissing problem on Ω_{4n} because their angle separation does not change due to the projection. Finally, the DNA codeword sets to be guaranteed to be at a minimum distance τ will be obtained by filtering from this kissing set those $4n$-dimensional points coding for DNA strands that are within a suitable angle ϵ/k (usually $k = 2$ or 3 will do) of a (normalized) shell point, for a suitable value of k. Here $\epsilon = |\phi - Arccos(1/3)|$ is the difference between the angle separation of two lattice points and the desired angle separation between DNA codewords. Table 2 shows the results for the corresponding set of 6−mers obtained from the Leech lattice. It is clear from this construction that the codeword sets thus obtained are nearly optimal in the DNA space with respect to the h_0−distance IF the spherical code is (nearly) optimal.

Thus, well established solutions to sphere packing problems can be used to obtain DNA codeword designs that are in some cases known to be nearly optimal (through difficult and elaborate proofs.) They include upper and lower bounds on estimates of the size of optimal codes for dimensions up to 128−mers as well as infinite families of DNA strand lengths, based on estimates of the kissing (or contact) number for sphere packings in Euclidean spaces, as shown in Table 2.

Table 2. Size of DNA (nearly optimal) codeword sets obtained from (nearly) optimal spherical codes and packings in comparison to best known sets obtained by exhaustive computational searches (Garzon et al., 2009) or other means. The second/last column shows the best (or best known) *proven* theoretical lower/upper bounds, respectively, on $A^*(n, \phi)$, the kissing numbers for h−distance spheres in the DNA space of all n−mers, as derived from the respective bounds for $A(4n, \phi)$ for $4n$D-spaces (Conway & Sloane, 1999, p. 24-27). Footnotes † indicate a result for metric h_0 only (hence the <), and ⋆ a result from PCRS *in vitro*; c is a constant independent of n and $\phi < \phi_0 \approx 70°$. (Blank cells either follow the general bounds or the precise values are still being computed.)

$\mathbf{D}_n$, n−mers	$? \le A^*(n,\phi)$	Spherical codes†	Best Known	$A^*(n,\phi) \le ?$
6		$< 1,268$	620	$196,560$
8		$< 4,096$	4,549	$261,120$
10			15,600	(N/A)
16			>> 58,500	$9,694,080$
20		< 100,000⋆		(N/A)
n	$2^{-4n\log_2(csin\phi)}$	(N/A)	(N/A)	$2^{7608(1-cos\phi)^{-4n}}$

4 Conclusions and Future Work

DNA Codeword design addresses the issue of *capacity* of fixed size olignucleotides for fault-tolerant self-assembly and computing and therefore is an important and challenging problem for molecular programming. We have used the theoretical framework described in (Phan & Garzon, 2009; Garzon et al., 1997) to show how the important Codeword Design problem of selecting noncrosshybrizing DNA oligonucleotides for DNA computing can be approximated through a geometric approach, despite its established **NP**-complete status. In this framework, Codeword Design is reduced to finding large sets of strands maximally separated in DNA spaces endowed with a mathematical metric directly related to the Gibbs energy of duplex formation (or rather its nearest neighbor model.) The size and composition of such sets is very difficult to establish because of the lack of intuition and tools as to the structure of the Gibbs energy landscapes. We have used a metric approximation of the Gibbs energy, the combinatorial $h-$distance , to introduce a new general technique to embed DNA spaces in the familiar Euclidean spaces by an isometric transformation that maps the $h-$distance to the ordinary Euclidean distance. To illustrate the type of practical applications of this embedding, we have shown a linear time reduction of the codeword design problem in DNA spaces to well researched but difficult sphere packing problems in Euclidean spaces (e.g., the so-called spherical code design problem (Conway & Sloan, 1999, p. 24).) Thus, upper and lower bounds on the size of optimal codeword designs for dimensions up to $128-$mers as well as infinite families of DNA strand lengths have been obtained. A full optimal solution to the problem of codeword design via this reduction, however, remains difficult despite their familiarity, because sphere packings are age-old difficult geometric problems. For example, their computational complexity and even specific solutions for a fixed dimension (including 3) are still unknown. Nevertheless, the consideration of sphere packings is necessary in this approach, despite the fact that not every point in Euclidean space represents a DNA strand, because the geometric intuitions and results afforded by the model are entirely absent in the string representation of DNA strands.

Conversely, this connection between codeword design and geometric problems has other applications. Solutions to codeword design obtained by experimental or other means also enable solutions to spherical packing problems via our embedding. Prior results for optimal codes by exhaustive search (Garzon et al., 2009) can be used to obtain packing lattices of reasonable quality for various applications (such as microarray design, crystallography, error-correcting codes in communication theory, and numeric integration.) As another example, the **NP**-completeness of finding optimal codeword sets for the $h-$distance implies the **NP-completeness of computing the kissing number** $A(n, \phi)$ **for** Ω_n, a problem that to the best of our knowledge has been open for some time.

There remains the question of whether these codes will be appropriate for use *in vitro*. This paper also presented a validation of the $h-$distance using the nearest neighbor model of the Gibbs energy, which is a standard model in experimental applications. Thus there is a good degree of confidence that these codes will increase the error-preventing/-correcting capability of codeword designs close to the extent possible. Moreover, the visualization of Gibbs energy landscapes for a given size of $n-$mers enabled by this embedding affords a more analytic and accessible tool to understanding biomolecular programming. For example, it appears plausible that this new metric approximation of Gibbs energies may allow other applications in shedding light on the structure of Gibbs energy landscapes for DNA spaces, particularly for designing long sequences of strands to form cascades descending towards stable equilibria along trajectories in Gibbs energy landscapes. For example, the embedding of DNA space of $n-$mers onto a hypersphere maps oligos with high (low) hybridization affinity (a *domain* in the language of chemical reaction networks) to neighboring (remote, respectively) points in a geometric lattice. Given an initial strand x in a domain and a target strand y in another, a shortest path connecting x to y in the spherical embedding (which can be easily found using Dijkstra's algorithm), suggests an optimal cascade of displacement reactions through intermediate strands to effect the transformation through minimal energy changes in going from one neighbor to the next. Moreover, this cascade would be fully fault-tolerant in the sense that nowhere along the cascade will the strands ever have a chance to "switch domains", i.e., to myshybridize with other strands in the mixture. (A more detailed description can be found in the analogy with planetary motion described in the Appendix on "Planetary Perturbations" in (Conway & Sloan, 1999, p. 29).) This application will be fully explored elsewhere.

References

1. Bobba, K.C., Neel, A.J., Phan, V., Garzon, M.H.: "Reasoning" and "Talking" DNA: Can DNA Understand English? In: Mao, C., Yokomori, T. (eds.) DNA12. LNCS, vol. 4287, pp. 337–349. Springer, Heidelberg (2006)
2. Chen, J., Deaton, R., Garzon, M., Wood, D.H., Bi, H., Carpenter, D., Wang, Y.Z.: Characterization of Non-Crosshybridizing DNA Oligonucleotides Manufactured in vitro. J. of Natural Computing 5(2), 165–181 (2006)
3. Conway, J.H., Sloane, N.J.: Sphere packings, lattices and groups. Comprehensive Studies in Mathematics, vol. 290. Springer (1999)
4. Deaton, J., Chen, J., Garzon, M., Wood, D.H.: Test Tube Selection of Large Independent Sets of DNA Oligonucleotides, pp. 152–166. World Publishing Co., Singapore (2006) (Volume dedicated to Ned Seeman on occasion of his 60th birthday)
5. Garzon, M.H., Wong, T.Y.: DNA Chips for Species identification and Biological Phylogenies. J. Natural Computing 10, 375–389 (2011)
6. Garzon, M.H., Phan, V., Neel, A.: Optimal Codes for Computing and Self-Assembly. Int. J. of Nanotechnology and Molecular Computing 1, 1–17 (2009)
7. Garzon, M.H., Yan, H. (eds.): DNA 2007. LNCS, vol. 4848. Springer, Heidelberg (2008)
8. Garzon, M.H., Phan, V., Bobba, K.C., Kontham, R.: Sensitivity and Capacity of Microarray Encodings. In: Carbone, A., Pierce, N.A. (eds.) DNA 11. LNCS, vol. 3892, pp. 81–95. Springer, Heidelberg (2006)
9. Garzon, M.H., Blain, D., Neel, A.J.: Virtual Test Tubes for Biomolecular Computing. J. of Natural Computing 3(4), 461–477 (2004)

10. Garzon, M., Neathery, P.I., Deaton, R., Murphy, R.C., Franceschetti, D.R., Stevens Jr., S.E.: A New Metric for DNA Computing. In: Koza, J.R., et al. (eds.) Proc. 2nd Annual Genetic Programming Conference, pp. 230–237. Morgan Kaufmann (1997)
11. Huget, J.M., Bizarro, C.V., Forns, N., Smith, S.B., Bustamante, C., Ritort, F.: Single-molecule derivation of salt-dependent base-pair free energies in DNA. PNAS 107(35), 15431–15436 (2010)
12. Neel, A., Garzon, M.: Semantic Retrieval in DNA-Based Memories with Gibbs Energy Models. Biotechnology Progress 22(1), 86–90 (2006)
13. Phan, V., Garzon, M.H.: On Codeword Design in Metric DNA Spaces. J. Natural Computing 8(3), 571–588 (2009)
14. Roman, J.: The Theory of Error-Correcting Codes. Springer, Berlin (1995)
15. SantaLucia, J.: A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proc. Natl. Acad. Sci. 95(4), 1460–1465 (1998)
16. Seeman, N.: DNA in a material world. Nature 421, 427–431 (2003)
17. Tulpan, D., Andronescu, M., Chang, S.B., Shortreed, M.R., Condon, A., Hoos, H.H., Smith, L.M.: Thermodynamically based DNA strand design. Nucleic Acids Res. 33(15), 4951–4964 (2005)
18. Qian, L., Winfree, E.: Scaling Up Digital Circuit Computation with DNA Strand Displacement Cascades. Science 332, 1196–1201 (2011)

A DNA Based Molecular Logic Gate Capable of a Variety of Logical Operations

Anton Kan, Koh-ichiroh Shohda, and Akira Suyama

Department of Life Sciences
and Institute of Physics Graduate School of Arts and Sciences,
The University of Tokyo, 3-8-1 Komaba, Meguro-ku, Tokyo 153-8902, Japan
{akan,shohda}@genta.c.u-tokyo.ac.jp, suyama@dna.c.u-tokyo.ac.jp

Abstract. In this paper we report on a module in the RTRACS molecular computing system, constructed with DNA, RNA and enzymes. The module is a 2-input logic gate that receives input and produces output in the form of RNA molecules. Each of the two input molecules is chosen from a set of two, and the logic gate produces an output molecule for each of the four possible input combinations. Two output RNA molecules can be produced by this module, one for only one combination of inputs, whilst the remaining three combinations lead to the production of the other output. Since the RNA strands can be arbitrarily assigned logical values, this module is capable performing multiple logical operations, including AND, NAND, OR and NOR, given the appropriate mapping of RNA molecules to logical values. We performed numerical simulations of the logic gate reaction scheme, revealing the details of the kinetics of the production of output molecules and the theoretical input-output characteristics. We also experimentally demonstrated the proper functioning of the logic gate, showing the correct formation of intermediate steps, the real time production of output molecules as well as the input-output characteristics of the module. We believe this versatile logic gate has significant advantages as a basic module of RTRACS due to the wide variety of possible logical operations.

Keywords: Biomolecular computing, DNA computing, logic gate.

1 Introduction

RTRACS (Reverse-transcription and TRanscription-based Autonomous Computing System) is a modular biomolecular computing system composed of DNA, RNA and enzymes, whose mechanism is based on retroviral replication. RTRACS modules are comprised of DNA molecules that communicate with each other through RNA strands, and the modules are capable of logical operations performed by enzymatically catalyzed reactions. The module functions autonomously as once all the component reactants and enzymes are in place, no further action and the gate will begin operation once the input RNA strands are added. The logical operations of such modules are analogous to electronic logic gates, although here inputs and outputs are in the form of RNA molecules rather than voltage levels. The modularity of RTRACS allows for many simple logic gates to be combined together by using output RNA as input for successive gates to create systems capable of sophisticated computational operations. The mechanism of retroviral replication has been naturally adapted to integrate with living cells, allowing RTRACS to be a potentially powerful tool for genetic diagnosis and therapy [1].

In this work we have built on the reaction scheme first suggested by Sakai *et al.* [2], demonstrating that the logic gate's reaction scheme can be used to perform a wide variety of logical operations, as well as showing the input-output relations using numerical simulations and real time experimental data.

2 Reaction Scheme

The logic gate module receives input and produces output in the form of 52 base single stranded RNA molecules. There are four possible input molecules, grouped into two pairs labeled $\mathbf{X} = \{X, X'\}$ and $\mathbf{Y} = \{Y, Y'\}$, and the outputs are a pair labeled $\mathbf{Z} = \{Z, Z'\}$. Each input and output RNA strand is comprised of two connected orthonormal sequences. Orthonormal sequences have been developed for reliable DNA computing [3], and our case are 23 bases long, have a uniform melting temperature, and no potential for mishybridization or stable self-folded structures. The labels x, x', y, ay, ay', z, z', and yp refer to such orthonormal segments, and sequences labelled cA are also orthonormal and complementary to A. All sequences are given in the 5'→3' direction. The RNA strand sequences are y-x for X, ay'-x' for X', ay-y for Y, ay'-y for Y', z-yp for Z and z'-yp for Z'. In this case, the $\mathbf{Z}$ output of the gate is designed to be a $\mathbf{Y}$ input strand for another similar gate, so that the sequence of the $\mathbf{Z}$ outputs follows the same format as for the $\mathbf{Y}$ strands.

The logic gate itself consists of two ssDNA (single stranded DNA) primer strands, two converter ssDNA molecules, and three enzymes to catalyze the necessary reactions. The primer strands consist of one orthonormal sequence, cx and cx'. The converter molecules Q and Q' have sequences $cyp - cz - ct7 - ay$ and $cyp - cz' - ct7 - ay'$ respectively, where $t7$ is the T7 promoter sequence. The converter molecules are capped by an NH_2 group at the 3' end to prevent wasteful elongation of the converter strands. The required enzymes are: AMV RTase (Reverse Transcriptase), which performs reverse transcription reactions, and is also capable of DNA polymerization and RNase H (Ribonuclease H) activity; *Thermus thermophilus* T7 RNAP (T7 RNA polymerase), which specifically binds to a double stranded T7 promoter sequence and produces RNA strands based on the DNA template; *Thermus Themophilus* RNase H, which specifically binds to RNA-DNA hybrids and digests the RNA.

The reaction proceeds with steps of RNA-DNA hybridization, reverse transcription, and RNA digestion, and follows the scheme of Fig. 1. The reaction begins similarly for both of the $\mathbf{X}$ input RNA strands, initially hybridizing with a complementary 23 base DNA primer strand, after which the RTase extends the primer strand to 52 bases, creating a sequence complementary to the RNA

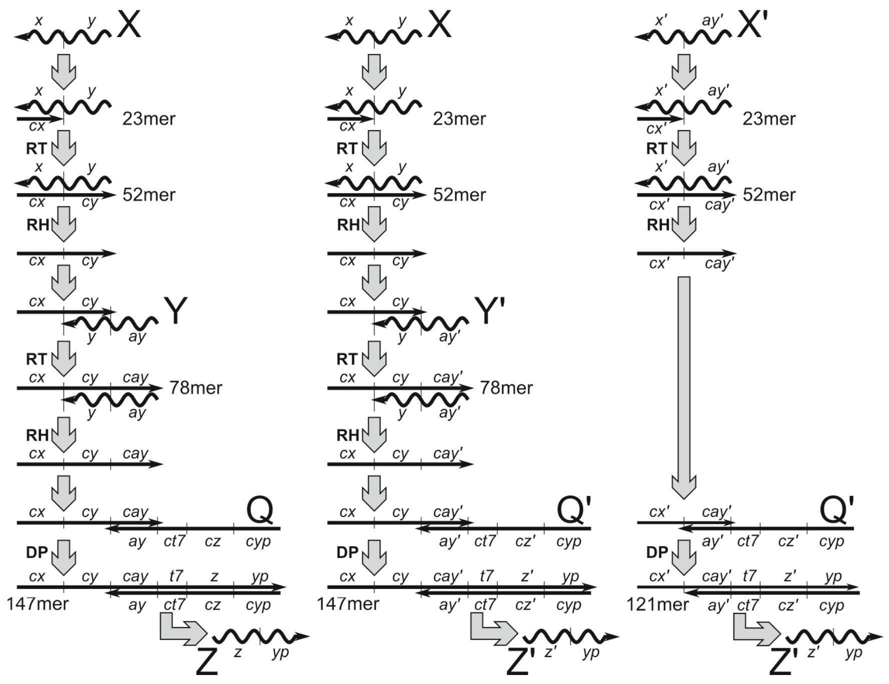


Fig. 1. The reaction scheme of the logic gate module for each distinct input combination. The curved lines denote RNA strands, whilst straight lines denote DNA strands. The polymer lengths in each case refer to the length of the DNA strands. RT denotes a reverse transcription reaction, RH to a ribonuclease H reaction and DP to a DNA polymerization reaction.

template. RNase H then cleaves the RNA strand from the RNA-DNA hybrid, leaving a 52 base ssDNA strand. Following this reaction, the scheme diverges for X and X'. With X input, the **Y** strands hybridize with the extended primer, undergoing further reverse transcription, elongation and RNA digestion to create a 78 base DNA strand. This completes the logical operation, with the newly extended segment of the 78 base DNA strand complementary to a segment of a converter DNA molecule, Q with the Y input or Q' with the Y' input.

With X' input, the logical operation is complete after a single reverse transcription, elongation and RNA digestion step, when the primer is 52 bases long. In this case, the Y and Y' molecules are not necessary as the newly extended segment is already complementary to the Q' converter molecule.

Once the extended primers hybridize to the complementary converter ssDNA, the DNA polymerase activity of the RTase elongates the extended primer to 147 bases in the case of the X input and 121 bases with X' input. This elongation creates a double stranded T7 promoter and output sequence, allowing the T7 RNA polymerase to bind and begin the production of **Z** output RNA molecules.

Table 1. Possible logical operations of the logic gate showing the required mapping of logical assignments (1 or 0) to RNA molecules.

RNA Molecule	AND Mapping	NAND Mapping	OR Mapping	NOR Mapping	INH Mapping		NINH Mapping	
X	1	1	0	0	0	1	0	1
X'	0	0	1	1	1	0	1	0
Y	1	1	0	0	1	0	1	0
Y'	0	0	1	1	0	1	0	1
Z	1	0	0	1	1	1	0	0
Z'	0	1	1	0	0	0	1	1

3 Logical Operations

Each pair of RNA inputs **X** and **Y** can be arbitrarily assigned to a pair of logical values {0, 1}. Unlike previous similar logic gates, such as the simpler AND gate [4], the reaction scheme of the logic gate presented here produces an output molecule for all correct combinations of inputs, and the output strands **Z** can also be arbitrarily assigned to logical 1 or 0. As well as allowing us to distinguish logical 0 from a malfunctioning of the gate, this has the added benefit of extending the functionality of this logic gate to multiple logical operations.

The gate is capable of performing any logical operation which combines two inputs, and outputs one molecule (Z) for only one specific combination, and another molecule (Z') for any of the other three combinations, as shown on the truth table in Table 2. Since the logical values can be assigned to both input and output strands in a situation dependent manner, any logical operation in this class can be performed. This class contains the AND, NAND, OR, NOR, INH (inhibitory, where the logical 1 output is produced only with a 10 or 01 input), and NINH (NOT inhibitory) operations, and the relevant mapping of molecule to logical value for each operation is shown in Table 1. Since this module is capable of both NOR and NAND operations, it is functionally complete and any logical operation can be realized with a combination of solely this logic gate module.

Table 2. Truth table for the reaction scheme

X Input	**Y** Input	**Z** Output
X	Y	Z
X'	Y	Z'
X	Y'	Z'
X'	Y'	Z'

4 Numerical Simulation

To demonstrate the functioning of this logic gate module, we first conducted numerical simulations following the methodology outlined by Takinoue *et al.* [4]. The logic gate was modeled using coupled ordinary differential equations, with a rate equation for each of the intermediate species and complexes. The reaction scheme in Fig. 1 does not take into account several back hybridizations and backward RNase H reactions that are possible under the reaction scheme, and these led to 6 new intermediate complexes not accounted for by the initial scheme. For example, the bound $X' - cx'$ complex is capable of undergoing reverse transcription to elongate the DNA, as well as an RH reaction to digest the X' input RNA. Taking all the possible species and complexes into account, as well as DNA molecule number conservation, we were ultimately left with 28 ordinary differential equations (not shown for brevity).

Several basic assumptions were made to simplify the model. Firstly, all enzymatic reactions were modeled with Michaelis Menten kinetics. Also, hybridization reactions were taken to be unidirectional since the bound products are stable, and the rates were taken to be proportional only to strand length.

$$\frac{d[J]}{dt} = kL[I_i][I_j] \tag{1}$$

$$\frac{d[J]}{dt} = \frac{vE_T[I]}{K_M}\left(1 + \sum_i \frac{[I_i]}{K_{Mi}}\right)^{-1} \tag{2}$$

$$\frac{d[J]}{dt} = \sum_j \frac{v_j E_{Tj}[I]}{[I] + K_{Mj}} \tag{3}$$

For each species or complex in the reaction, the rate equation is composed of appropriate hybridization and enzymatic terms. Hybridization terms are in the form of Eq. (1) (square brackets denote concentrations), for reactions of the form $I_i + I_j \rightarrow J$, where L is the base length of the binding section, and k an experimentally determined binding rate. The enzymatic reactions are of the general form $I + E \underset{k_-}{\overset{k_+}{\rightleftharpoons}} IE \xrightarrow{v} J + E$, although since enzymes can catalyze multiple reactions, and several substrates are capable of binding to multiple enzymes, it was necessary to use 2 different terms for each case. When substrates are competing for enzymes, Eq. (2) is used, with the parameter v denoting the rate of the enzymatic reaction, E_T the total enzyme concentration, $K_M = \frac{k_- + v}{k_+}$ and is the Michaelis-Menten constant for the reaction, and with the sum over i being the sum over the other possible reactions capable of being catalyzed by the enzyme. When enzymes compete for substrates, Eq. (3) is used, with the sum denoting the sum over all enzyme species capable of performing the reaction. To further simplify the equations, the results were normalized such that $kL = 1$ for $L = 26$ bases, and all RNA and DNA concentrations were normalized to $[Q] = [Q'] = 1$. Full derivations of equations (1)–(3) are given in reference [4].

The model was parametrized using results reported in other biochemical research. For RTase the Michaelis-Menten parameters are taken as $K_{M,RT} = 0.01\ uM$ for all reactions, and the maximum enzymatic catalysis rates for RTase are taken as $v_{RT} = 2\ s^{-1}$ for reverse transcription, $v_{RT,DP} = \frac{v_{RT}}{3}$ for DNA polymerization and $v_{RT,RH} = \frac{v_{RT}}{1200}$ for the RTase RNase H activity [5,6].

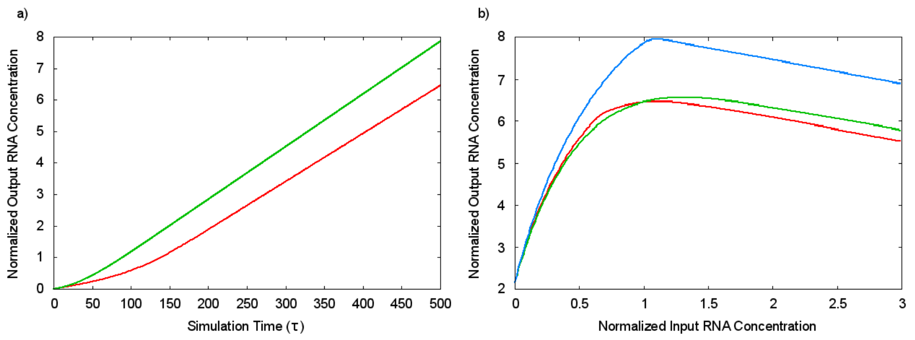


Fig. 2. Numerical simulation results: (a) Output RNA concentration as a function of time, with the red line corresponding to Z output RNA concentration for the (X, Y) input, and the green line corresponding to the Z' output RNA concentration for the (X', Y') input. In both cases the input RNA concentrations were kept at 1 normalized units ($= 0.1\ \mu M$) (b) Input-output characteristics for the logic gate module showing output RNA concentration after $\tau = 500$ as a function of initial input RNA, with the red and green lines showing Z output RNA concentration as a function of X and Y respectively for the (X, Y) input, and the blue line showing the Z' output as a function of X' for the (X', Y') input.

T7 RNAP has $K_{M,RP} = 0.02\ \mu M$ and $v_{RP} = 0.9\ s^{-1}$ for dsDNA [7], and basal transcription levels from ssDNA set to $K_{M,RP_B} = 0.1\ \mu M$ and $v_{RP_B} = \frac{v_{RP}}{2}$, based on our experiments. RNase H parameters were $K_{M,RH} = 7.8\ \mu M$ (defined for a 4-mer RNA sequence), $v_{RH} = 0.082\ s^{-1}$ [8]. The hybridization rate k was set at $10\ \mu M^{-1}s^{-1}$, based on our experimental measurements. Whilst many of the parameters in the literature are defined for slightly different temperatures than actually occurred in our reactions, we believe that the model captures the qualitative features of the reaction well. The total enzyme concentrations in the simulations were taken set to $0.8\ M$ for RTase, $0.15\ \mu M$ for T7 RNAP and $5.1 \times 10^{-5}\ M$ for RNase H, and these were the enzyme concentrations used in later experiments.

Figure 2(a) shows the results of the numerical simulations. This figure displays the reaction results for only the (X, Y) and (X', Y') input states, since the kinetics for the (X, Y') and (X', Y) reactions are identical to the (X, Y) and (X', Y') cases respectively.

The simulations revealed a lag on the production in the reactions containing the X input as compared to the X' input arising from the extra steps in the reaction. Several factors contribute to this lag, firstly increased competition for the RTase enzyme, slowing many enzymatic steps. In particular, the ribunuclease H steps caused the greatest delay in the reaction, due to the low concentration of RNaseH itself and the competition for RTase. The lag was shortest at lower concentrations of input, and increased with input RNA concentration, due to higher concentrations of intermediate steps delaying output production due to increased competition for enzymes and back reactions. The maximum production rate of output RNA increased with input concentration, and approached a maximum soon after the concentration of the input surpassed the primer concentration, as this led to all of the relevant converter molecules becoming double stranded. Given enough input RNA, the output production rate is therefore primarily limited by the concentration of T7 RNAP, as the concentration of double stranded converter molecules is well above the Michaelis-Menten constant for T7 RNAP.

Fig. 2(b) shows the theoretical input-output characteristics of the logic gate, giving the concentration of output RNA after 500 simulation time units (τ) as a function of input RNA concentration. The logic gate was sensitive to the input molecules, giving a sharp increase in response as the concentration of input molecules is raised from zero. A maximum is reached at the point when there was enough input RNA to lead to the full conversion of the converter molecules from ssDNA to dsDNA, and this is slightly higher than the concentration of the primers due to the unwanted back reactions. After the peak, increasing input RNA concentration did not further increase the gradient of output RNA concentration, but did linearly increase the lag time, thus there was a shallow linear decrease in the RNA output concentration as input concentration is increased. For the (X, Y) (and (X, Y')) input the production was slightly more sensitive to the X input RNA than to the Y (or Y') input RNA.

5 Experimental Data

We experimentally demonstrated the proper operation of the logic gate in two separate ways. Firstly, the production of the correct intermediate states of the logic gate were demonstrated by showing the elongation of DNA primer strands. Secondly, we observed the real time production kinetics of the correct output RNA strands.

5.1 Correct Elongation

To demonstrate correct formation of intermediate states, the full logic gate reaction was carried out with ssDNA primer strands tagged with fluorescent dye at the 5' end, Cy5-*cx* and Cy3-cx'. The dyes emit fluorescence at different wavelengths, allowing the products to be easily identified.

The full reaction for each of the four input states $(X, Y), (X, Y'), (X', Y)$ and (X', Y') was carried out at 50 °C for 2 hours. The results of each reaction were mixed with a denaturing 8M Urea and TBE electrophoresis buffer, and added to different lanes of a denaturing gel containing 12% polyacrylamide (acrylamide/bisacrylamide, 29:1). The electrophoresis was then performed at 250 V and 65 °C. Following the separation of the reaction products, the gel was imaged at the appropriate wavelengths to observe Cy5 and Cy3 fluorescence using a fluoroimage analyzer, FLA-5100 (Fuji Film, Japan).

Figure 3 shows the results of the electrophoresis, clearly showing the correctly elongated fluorescent primers. In both images, the bottom-most band corresponds to the unelongated primers. As expected from the reaction scheme, the Cy5-*cx* primer elongation only occurred in the reactions containing the X input RNA, whereas the Cy3-cx' primer was elongated only in the presence

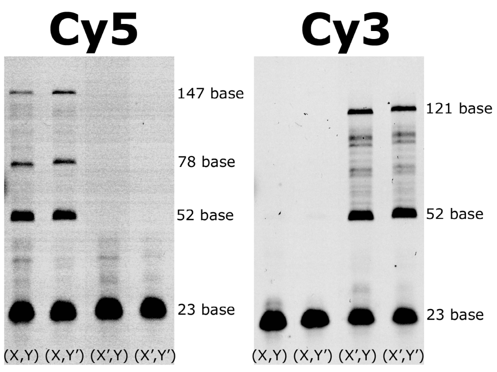


Fig. 3. Images of Cy5 and Cy3 fluorescence intensity in the same gel after denaturing gel electrophoresis

of the X' molecule. The Cy5-*cx* primer was elongated 3 times, firstly to 52 bases, then to 78 bases, by successive steps of RNA hybridization, reverse transcription, and RNA digestion, the third 147 base product was created by the DNA polymerization step after hybridization with the converter molecule. In contrast, the Cy3-cx' primer was elongated twice, firstly to 52 bases and then to 121 bases following binding to the converter molecule Q' and DNA polymerization, as expected from the reaction scheme.

5.2 Real Time Operation

To experimentally observe the real time production of output RNA molecules, the full reaction was carried out using molecular beacons to report on the output of **Z** RNA molecules. The molecular beacons are RNA/DNA molecules capable of forming stem and loop structures, with the loop containing a segment complementary to the RNA sequences of interest. The ends of the molecular beacons strands are attached to a fluorophore and a quencher such that the hairpin state does not emit fluorescence due to FRET (fluorescence resonance energy transfer) quenching the fluorophore. Two molecular beacon types were added to the reaction, *cz*-FAM and cz'-Cy5, to report on Z and Z' respectively, and the beacons are spectrally distinct. The molecular beacons were annealed before the reaction to ensure that they were in the hairpin state, and consequently emit only low fluorescence in the absence of the appropriate **Z** molecule.

The full reaction for each of the 4 input states was conducted at 50 °C using the same buffer condition as for the electrophoresis reaction, with the exception of the addition of 1.5 μM of *cz*-FAM and cz'-Cy5 molecular beacons. The FAM and Cy5 fluorescence intensity was then measured every 30 seconds in real time PCR apparatus, the LightCycler 480 II (Roche), and the results are given in Fig. 4.

The results clearly show that the logic gate reaction produced the expected outputs. The significant gains in fluorescence occurred only for the input states in which we expect the appropriate **Z** output molecule to be present. The fluorescence intensity in reactions containing the correct inputs increased over 3 fold compared to those without in a period of 2 hours, allowing for robust result determination. The behaviour of the fluorescence intensity for reactions without the appropriate inputs were indistinguishable from no input at all, and the slight increase in intensity corresponds to basal levels of spurious transcription arising from T7 RNAP acting on ssDNA converter strands.

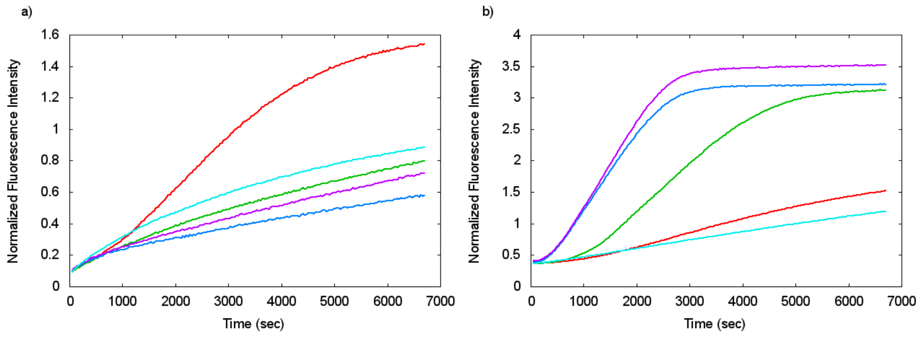


Fig. 4. Real time PCR data showing the normalized fluorescence intensity of spectrally distinct molecular beacons during the same reaction. (a) *cz*-FAM molecular beacon fluorescence intensity reporting on Z output RNA molecules, (b) cz'-Cy5 molecular beacon fluorescence intensity reporting on Z' RNA. In both charts, the red line corresponds to the (X, Y) input, the green line to the (X, Y') input, the dark blue line to the (X, Y') input, the purple line to the (X', Y') input, and the light blue line to no input RNA molecules present. The fluorescence intensity is normalized to the intensity of the molecular beacon at 95 °C.

To show the input-output characteristics of the gate, the fluorescence intensity of the cz'-Cy5 molecular beacon is shown as a function of input RNA concentration in Fig. 5. Since the reactions for the (X, Y) and (X, Y') have the same kinetics, differing only in the sequences of the DNAs and RNAs, only the results for the (X, Y') state are shown in comparison to (X', Y') (itself being identical to (X', Y)) so that Cy5 fluorescence can be compared.

Figure 5 demonstrates the experimentally determined input-output characteristics of the logic gate module, with the fluorescence intensity of the cz'-Cy5 molecular beacon shown as a function of input RNA concentration for two combinations of inputs that produce the Z' output RNA molecule. The logic gate was sensitive to low concentrations of input RNA, with the normalized fluorescence intensity of the molecular beacon increasing to over 2 fold the basal levels with even 0.025 μM of input RNA. The shape of the output response matches theoretical predictions, peaking close to 0.1 μM after which there was a shallow decline due to increased lag time. Also as predicted by simulation, the output was more sensitive to the X input RNA molecules than the Y'.

The experimentally determined results in Figs. 4 and 5 matched those of the numerical simulations results in Fig. 2 well. The kinetics of the X input states displayed the production lag predicted by the models, which was on the order of 10 minutes. The gradients of Z and Z' production were similar, also matching simulation results. The difference in the gradients of the output of the X and X' reactions was more pronounced than the models suggested, although this can be understood to have arisen from RNA degradation occurring during the experiments that the model did not take into account.

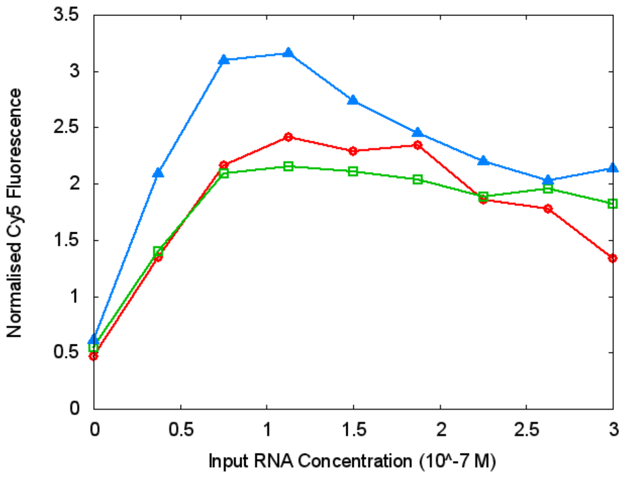


Fig. 5. Cy5 fluorescence intensity reporting on the Z' output RNA as a function of input RNA concentration after 3000 seconds of the reaction. Blue line and triangle markers correspond to varying X' for the (X', Y') input, the red line with circular markers corresponds to varying X for the (X, Y') input, and the green line with square markers corresponds to varying Y' for the (X, Y') input.

6 Discussion

We have successfully demonstrated numerically and experimentally the functioning of the of this RTRACS logic gate module. The production of correct RNA output molecules is significant enough to allow for robust determination of the result of the logical operation on hour timescales. The lags in production arising from more reaction steps for certain input combinations were only on the order of 10 minutes, and are thus unlikely to slow downstream kinetics significantly if this module is used in a complex reaction network. The logic gate was also sensitive to input RNA concentrations, and gives a significant increase in output RNA at all input RNA concentrations above 0.5 μM.

The versatility of the logical operations possible by this gate is very promising, allowing complex logical operations to be performed with combinations of this logic gate module in series. When embedded in an RTRACS network comprised of several modules, this logic gate is capable of performing different logical operations in different situations without needing to alter the network, adding to the power and versatility of RTRACS. To demonstrate that the gates can be chained together, simulations were performed with a linear growth of input RNA molecules rather than at a fixed initial concentration, as would be the case if one module followed another in series. The input RNA molecule concentrations were initially at 0, and increased with a maximal gradient of output RNA concentration found in Fig. 2(a). The results in Fig. 6 demonstrate that although the lag on the output of the second gate us larger, there was still a significant

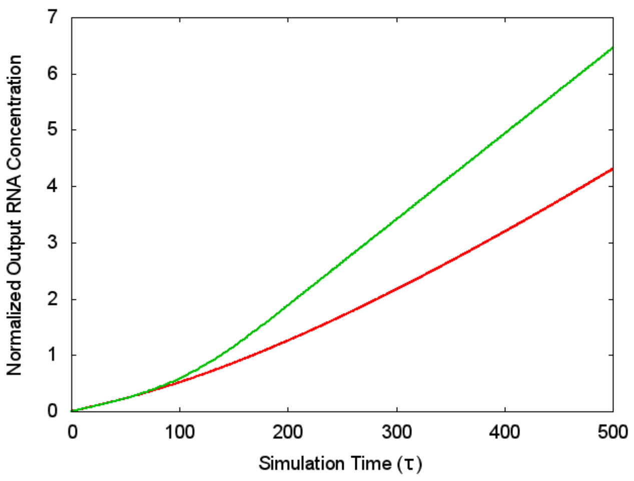


Fig. 6. Chaining gates: simulation results for the operation of the gate varying initial conditions. The green line corresponds to the Z output of the gate with an initial fixed amount of X and Y input RNA molecules (1 normalized unit = 0.1 μM), the red line is the Z output for a linear increase in X and Y input RNA molecules from 0 with the gradient being the maximal gradient of the green line.

increase in output RNA for the second gate, and the output production rate did eventually reach the maximal rate. In this way, gates connected in series could perform computation through successive logical operations.

Although the gate is not dynamic since output production continues indefinitely once the logical operation is complete, given enough monomers and enzyme activity, components of the logic gate remain unused, which raises intriguing possibilities for their reuse in subsequent reactions. For example, after an (X, Y) input creates the Z producing duplex, the cx' and Q' molecules are still available for subsequent operations, so that in certain situations the gate could be used more than once. Such free components of the module could also provide efficiencies when scaling up RTRACS to contain several such modules.

The numerical modeling of the gate has been shown as a useful and accurate tool to model the operation of the gate, and such simulations could be extended to model several modules in concert. This would be a valuable tool in the design of more complex RTRACS circuits, and to optimize designs for particular tasks.

7 Conclusion

We have successfully demonstrated the proper functioning of the logic gate module, and have shown that our numerical models can accurately portray the behaviour of the module. The sensitivity and high versatility of this basic module shows great promise in giving RTRACS powerful computing capabilities.

References

1. Nitta, N.: Retrovirus-based computer. Natural Computing 4, 127–139 (2005)
2. Sakai, Y., Mawatari, Y., Yamasaki, K., Shohda, K.-I., Suyama, A.: Construction of AND Gate for RTRACS with the Capacity of Extension to NAND Gate. In: Deaton, R., Suyama, A. (eds.) DNA 15. LNCS, vol. 5877, pp. 137–143. Springer, Heidelberg (2009)
3. Kitajima, T., Takinoue, M., Shohda, K.-I., Suyama, A.: Design of Code Words for DNA Computers and Nanostructures with Consideration of Hybridization Kinetics. In: Garzon, M.H., Yan, H. (eds.) DNA 2007. LNCS, vol. 4848, pp. 119–129. Springer, Heidelberg (2008)
4. Takinoue, M., Kiga, D., Shohda, K.-I., Suyama, A.: Experiments and simulation models of a basic computation element of an autonomous molecular computing system. Phys. Rev. E 78, 041921 (2008)
5. Krug, M.S., Berger, S.L.: Reverse transcriptase from human immunodeficiency virus: a single template-primer binding site serves two physically separable catalytic functions. Biochemistry 30(44), 10614–10623 (1991)
6. Yu, H., Goodman, M.F.: Comparison of HIV-1 and avian myeloblastosis virus reverse transcriptase fidelity on RNA and DNA templates. Journal of Biological Chemistry 267(15), 10888–10896 (1992)
7. Martin, C.T., Coleman, J.E.: Kinetic analysis of T7 RNA polymerase-promoter interactions with small synthetic promoters. Biochemistry 26(10), 2690–2696 (1987)
8. Hirano, N., Haruki, M., Morikawa, M., Kanaya, S.: Enhancement of the enzymatic activity of ribonuclease HI from thermus thermophilus HB8 with a suppressor mutation method. Biochemistry 39(43), 13285–13294 (2000)

Deciding Whether a Regular Language Is Generated by a Splicing System[*]

Lila Kari and Steffen Kopecki

Department of Computer Science,
The University of Western Ontario,
London ON N6A 5B7 Canada
{lila,steffen}@csd.uwo.ca

Abstract. Splicing as a binary word/language operation is inspired by the DNA recombination under the action of restriction enzymes and ligases, and was first introduced by Tom Head in 1987. Shortly thereafter, it was proven that the languages generated by (finite) splicing systems form a proper subclass of the class of regular languages. However, the question of whether or not one can decide if a given regular language is generated by a splicing system remained open. In this paper we give a positive answer to this question. Namely, we prove that, if a language is generated by a splicing system, then it is also generated by a splicing system whose size is a function of the size of the syntactic monoid of the input language, and which can be effectively constructed.

1 Introduction

In [10] Head described an operation on formal languages, called *splicing*, which models DNA recombination, a cut-and-paste operation on DNA strands under the action of restriction enzymes and ligases. A splicing system consists of a set of *axioms* or *initial words* and a set of *(splicing) rules*. The most commonly used definition for a splicing rule is a quadruple of words $r = (u_1, v_1; u_2, v_2)$. This rule splices two words $x_1u_1v_1y_1$ and $x_2u_2v_2y_2$: the words are cut between the factors u_1, v_1, respectively u_2, v_2, and the prefix (the left segment) of the first word is recombined by catenation with the suffix (the right segment) of the second word, see Figure 1 and also [17].

Splicing as a language-theoretic word operation is meant to abstract the action of two compatible restriction enzymes and the ligase enzyme on two DNA strands. The first enzyme recognizes the subword u_1v_1, called its *restriction site*, in any DNA string and cuts the string containing this subword between u_1 and v_1. The second restriction enzyme, with restriction site u_2v_2, acts similarly. Assuming that the "sticky ends" obtained after these cuts are in some sense "compatible", the enzyme ligase aids then the recombination (catenation) of the first segment of one cut string with the second segment of another cut string.

[*] This research was supported by the Natural Sciences and Engineering Research Council of Canada Discovery Grant R2824A01 and UWO Faculty of Science grant to L. K.

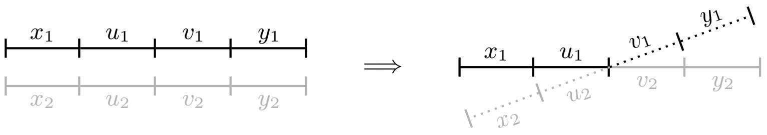


Fig. 1. Splicing of the words $x_1u_1v_1y_1$ and $x_2u_2v_2y_2$ by the rule $r = (u_1, v_1; u_2, v_2)$. The splicing result is the word $x_1u_1v_2y_2$.

A splicing system generates a language which contains every word that can be obtained by successively applying rules to axioms and the intermediately produced words. The most natural variant of splicing systems, often referred to as finite splicing systems, is to consider a finite set of axioms and a finite set of rules. In this paper, by a splicing system we always mean a finite splicing system. Shortly after the introduction of splicing in formal language theory, Culik II and Harju [6] proved that splicing systems generate regular languages, only; see also [12,16]. Gatterdam [7] gave $(aa)^*$ as an example of a regular language which cannot be generated by a splicing system; thus, the class of languages generated by splicing systems is strictly included in the class of regular languages. However, for a regular language L over an alphabet Σ, adding a marker $b \notin \Sigma$ to the left side of every word in L results in the language bL which can be generated by a splicing system [11]; e. g., the language $b(aa)^*$ is generated by the axioms $\{b, baa\}$ and the rule $(baa, \varepsilon; b, \varepsilon)$, where ε is the empty word.

This led to the question of whether or not one of the known subclasses of the regular languages corresponds to the class $\mathcal{S}$ of languages which can be generated by a splicing system. All investigations to date indicate that the class $\mathcal{S}$ does not coincide with another naturally defined language class. A characterization of *reflexive* splicing systems using *Schützenberger constants* has been given by Bonizzoni, de Felice, and Zizza [1–3]. A splicing system is reflexive if for all rules $(u_1, v_1; u_2, v_2)$ in the system we have that $(u_1, v_1; u_1, v_1)$ and $(u_2, v_2; u_2, v_2)$ are rules in the system, too. A word v is a Schützenberger constant of a language L if $x_1vy_1 \in L$ and $x_2vy_2 \in L$ imply $x_1vy_2 \in L$ [18]. Recently, it was proven by Bonizzoni and Jonoska that every splicing language has a constant [5]. However, not all languages which have a constant are generated by splicing systems, e. g., in the language $L = (aa)^* + b^*$ every word b^i is a constant, but L is not generated by a splicing system.

Another approach was to find an algorithm which decides whether a given regular language is generated by a splicing system. This problem has been investigated by Goode, Head, and Pixton [8,9,13] but it has only been partially solved: It is decidable whether a regular language is generated by a reflexive splicing system. It is worth mentioning that a splicing system by the original definition in [10] is always reflexive.

In this paper we settle the problem by proving that for a given regular language L, it is indeed decidable whether L is generated by a splicing system (which is not necessarily reflexive), Corollary 1. More precisely, if the language L is generated by a splicing system, then it is generated by one particular splicing system whose size is a function of the size of the syntactic monoid of L, Theorem [1]. If m is the size of the syntactic monoid of L, then all axioms and all components of rules have a length in $\mathcal{O}(m^2)$. By results from [12, 13], we can construct a finite automaton which accepts the language generated by this splicing system, compare it with a finite automaton which accepts L, and, thus, decide whether L is generated by a splicing system.

Due to page limitations the proofs of most of the lemmas have been omitted in this version of the paper. The missing proofs can be found in the arXiv version [15].

2 Notation and Definitions

We assume the reader to be familiar with the fundamental concepts of language theory, see [14]. Let Σ be an *alphabet*, Σ^* be the set of all words over Σ, and ε denote the *empty word*. A subset L of Σ^* is a *language* over Σ. Throughout this paper, we consider languages over the alphabet Σ, only. We consider the letters of Σ to be ordered and for words $u, v \in \Sigma^*$ we denote the *(strict) length-lexicographical order* by $u \leq_{\ell\ell} v$ (resp., $u <_{\ell\ell} v$); i. e., $u \leq_{\ell\ell} v$ if either $|u| \leq |v|$, or $|u| = |v|$ and u is equal or less than v in lexicographic order. For a length bound $m \in \mathbb{N}$ we let $\Sigma^{\leq m}$ denote the set of words whose length is at most m, i. e., $\Sigma^{\leq m} = \bigcup_{i \leq m} \Sigma^i$. Analogously, we define $\Sigma^{<m} = \bigcup_{i<m} \Sigma^i$. Let $w \in \Sigma^*$ be a word. If $w = xyz$ for some $x, y, z \in \Sigma^*$, then x, y, and z are called *prefix*, *factor*, and *suffix* of w, respectively. If a prefix or suffix of w is distinct from w, it is said to be *proper*.

Every language L induces a *syntactic congruence* $\sim_L$ over words such that $u \sim_L v$ if and only if for all words x, y we have $xuy \in L \iff xvy \in L$. The *syntactic class* (with respect to L) of a word u is $[u]_L = \{v \mid u \sim_L v\}$. The *syntactic monoid* of L is the quotient monoid $M_L = \Sigma^* / {\sim_L} = \{[u]_L \mid u \in \Sigma^*\}$. It is well-known that a language L is regular if and only if its syntactic monoid M_L is finite. We will use two basic facts about syntactic monoids of regular languages.

Lemma 1. *Let L be a regular language and let w be a word with $|w| \geq |M_L|^2$. We can factorize $w = \alpha\beta\gamma$ with $\beta \neq \varepsilon$ such that $\alpha \sim_L \alpha\beta$ and $\gamma \sim_L \beta\gamma$.*

Lemma 2. *Let L be a regular language. Every element $X \in M_L$ contains a word $x \in X$ with $|x| < |M_L|$.*

3 Splicing Systems and Regular Languages

We consider the splicing operation as defined in [17]. This is the most commonly used definition for splicing in formal language theory. The notation we use has been employed in previous papers, see e. g., [2, 9]. A quadruple of words $r = (u_1, v_1; u_2, v_2) \in (\Sigma^*)^4$ is called a *(splicing) rule*. The words u_1v_1 and u_2v_2 are called *left* and *right side* of r, respectively. This splicing rule can be applied to two words $w_1 = x_1u_1v_1y_1$ and $w_2 = x_2u_2v_2y_2$, that each contain one of the sides, in order to create the new word $z = x_1u_1v_2y_2$, see again Figure 1. This operation is called *splicing* and it is denoted by $(w_1, w_2) \vdash_r z$. The *splicing position* of this splicing is the position between the factors x_1u_1 and v_2y_2 in z.

For a rule r we define the *splicing operator* σ_r such that for a language L

$$\sigma_r(L) = \{z \in \Sigma^* \mid \exists w_1, w_2 \in L\colon (w_1, w_2) \vdash_r z\}$$

and for a set of splicing rules R, we let $\sigma_R(L) = \bigcup_{r\in R} \sigma_r(L)$. The reflexive and transitive closure of the splicing operator σ_R^* is given by

$$\sigma_R^0(L) = L, \qquad \sigma_R^{i+1}(L) = \sigma_R^i(L) \cup \sigma_R(\sigma_R^i(L)), \qquad \sigma_R^*(L) = \bigcup_{i\geq 0} \sigma_R^i(L).$$

A finite set of axioms $I \subseteq \Sigma^*$ and a finite set of splicing rules $R \subseteq (\Sigma^*)^4$ form a *splicing system* (I, R). Every splicing system (I, R) generates a language $L(I, R) = \sigma_R^*(I)$. Note that $L(I, R)$ is the smallest language which is closed under the splicing operator σ_R and includes I. It is known that the language generated by a splicing system is regular, see [6,16]. A (regular) language L is called a *splicing language* if a splicing system (I, R) exists such that $L = L(I, R)$.

A rule r is said to *respect* a language L if $\sigma_r(L) \subseteq L$. It is easy to see that for any splicing system (I, R), every rule $r \in R$ respects the generated language $L(I, R)$ and a rule $r \notin R$ respects $L(I, R)$ if and only if $L(I, R \cup \{r\}) = L(I, R)$. Furthermore, we say a splicing step $(w_1, w_2) \vdash_r z$ *respects* a language L if $w_1, w_2 \in L$ and r respects L; obviously, this implies $z \in L$, too.

The purpose of this section is to prove that if a regular language L is a splicing language, then it is created by a splicing system (I, R) which only depends on the syntactic monoid of L.

Theorem 1. *Let L be a splicing language and $m = |M_L|$. The splicing system (I, R) with $I = \Sigma^{<m^2+6m} \cap L$ and*

$$R = \left\{r \in \Sigma^{<m^2+10m} \times \Sigma^{<2m} \times \Sigma^{<2m} \times \Sigma^{<m^2+10m} \;\middle|\; r \text{ respects } L\right\}$$

generates the language $L = L(I, R)$.

The structure of this section is the following. In Section 3.1 we will present techniques to obtain rules that respect a regular language L from other rules respecting L and we show how we can modify a single splicing step, such that the words used for splicing are not significantly longer than the splicing result; similar results can be found in [8,9]. In Section 3.2 we use these techniques to show that a long word $z \in L$ can be obtained by a series of splicings from a set shorter words from L and by using rules which satisfy certain length restrictions. Finally, in Section 3.3 we prove Theorem 1.

3.1 Rule Modifications

Our first lemma tells us that we can extend the sides of a rule r such that the extended rule respects all languages that are respected by r.

Lemma 3. *Let $r = (u_1, v_1; u_2, v_2)$ be a rule which respects a language L. For every word x, the rules $(xu_1, v_1; u_2, v_2)$, $(u_1, v_1x; u_2, v_2)$, $(u_1, v_1; xu_2, v_2)$, and $(u_1, v_1; u_2, v_2x)$ respect L as well.*

Henceforth, we will refer to the rules $(xu_1, v_1; u_2, v_2)$, $(u_1, v_1x; u_2, v_2)$ as extensions of the left side and to $(u_1, v_1; xu_2, v_2)$, $(u_1, v_1; u_2, v_2x)$ as extensions of the right side.

Next, for a language L, let us investigate the syntactic class of a rule $r = (u_1, v_1; u_2, v_2)$. The *syntactic class* (with respect to L) of r is the set of rules $[r]_L = [u_1]_L \times [v_1]_L \times [u_2]_L \times [v_2]_L$ and two rules r and s are *syntactically congruent* (with respect to L), denoted by $r \sim_L s$, if $s \in [r]_L$.

Lemma 4. *Let r be a rule which respects a language L. Every rule $s \in [r]_L$ respects L.*

Consider a splicing $(x_1u_1v_1y_1, x_2u_2v_2y_2) \vdash_r x_1u_1v_2y_2$ which respects a regular language L, as shown in Figure 2 on the left side. The factors v_1y_1 and x_2u_2 may be relatively long but they do not occur as factors in the resulting word $x_1u_1v_2y_2$. In particular, it is possible that two long words are spliced and the outcome is a relatively short word. Using the Lemmas 3 and 4, we can find shorter words in L and a modified splicing rule which can be used to obtain $x_1u_1v_2y_2$.

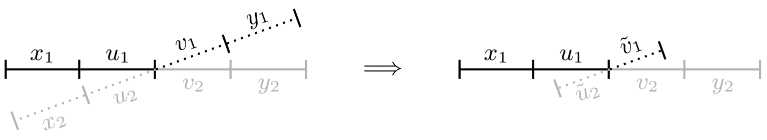


Fig. 2. Replacing v_1y_1 and x_2u_2 by *short* words

Lemma 5. *Let $r = (u_1, v_1; u_2, v_2)$ be a rule which respects a regular language L and $w_1 = x_1u_1v_1y_1 \in L$, $w_2 = x_2u_2v_2y_2 \in L$. There is a rule $s = (u_1, \tilde{v}_1; \tilde{u}_2, v_2)$ which respects L and words $\tilde{w}_1 = x_1u_1\tilde{v}_1 \in L$, $\tilde{w}_2 = \tilde{u}_2v_2y_2 \in L$ such that $|\tilde{v}_1|, |\tilde{u}_2| < |M_L|$. More precisely, $\tilde{v}_1 \in [v_1y_1]_L$ and $\tilde{u}_2 \in [x_2u_2]_L$.*

In particular, whenever $(w_1, w_2) \vdash_r x_1u_1v_2y_2 = z$, then there is a splicing $(\tilde{w}_1, \tilde{w}_2) \vdash_s z$ which respects L where $\tilde{w}_1$, $\tilde{w}_2$, and s have the properties described above.

3.2 Series of Splicings

Let us consider the creation of words by a series of splicings. We begin with a simple observation. In case when a word is created by two (or more) successive splicings, but the sides of the splicings do not cover the splicing position of the other splicing, then the order of these splicings is irrelevant. Recall that the splicing position of a splicing $(w_1, w_2) \vdash_r z$ with $r = (u_1, v_1; u_2, v_2)$ is the position between the factors u_1 and v_2 in z. The notation in Remark 1 is the same as in the Figure 3.

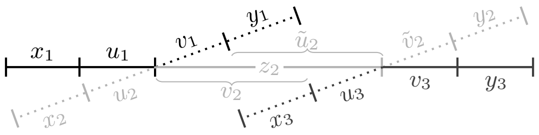


Fig. 3. The word $x_1u_1z_2v_3y_3$ can be created by either using the right splicing first or by using the left splicing first

Remark 1. Let $w_1 = x_1u_1v_1y_1$, $w_2 = x_2u_2z_2\tilde{v}_2y_2$, where v_2 is a prefix of z_2 and $\tilde{u}_2$ is a suffix of z_2, $w_3 = x_3u_3v_3y_3$ be words and $r_1 = (u_1, v_1; u_2, v_2)$, $r_2 = (\tilde{u}_2, \tilde{v}_2; u_3, v_3)$ be rules. In order to create the word $z = x_1u_1z_2v_3y_3$ by splicing, we may use splicings

$$(w_1, w_2) \vdash_{r_1} x_1u_1z_2\tilde{v}_2y_2 = z', \qquad (z', w_3) \vdash_{r_2} z \qquad \text{or}$$
$$(w_2, w_3) \vdash_{r_2} x_2u_2z_2v_3y_3 = z'', \qquad (w_1, z'') \vdash_{r_1} z.$$

Consider a splicing system (J, S) and the generated language $L = L(J, S)$. Let n be the length of the longest word in J and let μ be the length-lexicographically largest word that is a component of a rule in S. Define $W_\mu = \{w \in \Sigma^* \mid w \leq_{\ell\ell} \mu\}$ as the set of words which are at most as large as μ, in length-lexicographic order. Furthermore, let $I = \Sigma^{\leq n} \cap L$ be a set of axioms and let

$$R = \left\{r \in W_\mu^4 \;\middle|\; r \text{ respects } L\right\}$$

be a set of rules. It is not difficult to see that $J \subseteq I$, $S \subseteq R$, and $L = L(I, R)$. Whenever convenient, we will assume that a splicing language L is generated by a splicing system which is of the form of (I, R).

Let $xzy \in L$ be a word where the length of the middle factor z is at least $|\mu|$. The creation of xzy by splicing in (I, R) can be traced back to a word $x_1zy_1 = z_1$ where either $z_1 \in I$ or where z_1 is created by a splicing that affects the factor z, i. e., the splicing position lies in the factor z. The next lemma describes this creation of $xzy = z_{k+1}$ by k splicings in (I, R), and shows that we can choose the rules and words which are used to create z_{k+1} from z_1 such that the words and bridges of rules are not significantly longer than $\ell = \max\{|x|, |y|\}$.

Lemma 6. *Let L be a splicing language, let $\ell, n \in \mathbb{N}$, let $m = |M_L|$, and let μ be a word with $|\mu| \geq \ell + 2m$ such that for $I = \Sigma^{\leq n} \cap L$ and $R = \left\{r \in W_\mu^4 \;\middle|\; r \text{ respects } L\right\}$ we have $L = L(I, R)$.*

Let $z_{k+1} = x_{k+1}zy_{k+1}$ with $|z| \geq |\mu|$ and $|x_{k+1}|, |y_{k+1}| \leq \ell$ be a word that is created by k splicings from a word $z_1 = x_1zy_1$ where either $z_1 \in I$ or z_1 is created by a splicing $(\tilde{w}_1, \tilde{w}_2) \vdash_s z_1$ where $\tilde{w}_1, \tilde{w}_2 \in L$, s respects L, and the splicing position lies in the factor z. Furthermore, for $i = 1, \ldots, k$ the intermediate splicings are either

(i) $(w_i, z_i) \vdash_{r_i} x_{i+1}zy_{i+1} = z_{i+1}$, $w_i \in L$, $r_i \in R$, $y_{i+1} = y_i$, and the splicing position lies on the left of the factor z or
(ii) $(z_i, w_i) \vdash_{r_i} x_{i+1}zy_{i+1} = z_{i+1}$, $w_i \in L$, $r_i \in R$, $x_{i+1} = x_i$, and the splicing position lies on the right of the factor z.

There are rules and words creating z_{k+1}, as above, satisfying in addition:

1. *There is $k' \leq k$ such that for $i = 1, \ldots, k'$ all splicings are of the form (i) and for $i = k' + 1, \ldots, k$ all splicings are of the form (ii).*
2. *For $i = 1, \ldots, k'$ the following bounds apply: $|x_i| < \ell + 2m$, $|w_i| < \ell + 2m$, $r_i \in \Sigma^{<\ell+m} \times \Sigma^{<2m} \times \Sigma^{<2m} \times W_\mu$, and $x_{k'+1} = x_{k'+2} = \cdots = x_{k+1}$.*
3. *For $i = k'+1, \ldots, k$ the following bounds apply: $|y_i| < \ell+2m$, $|w_i| < \ell+2m$, $r_i \in W_\mu \times \Sigma^{<2m} \times \Sigma^{<2m} \times \Sigma^{<\ell+m}$, and $y_1 = y_2 = \cdots = y_{k'+1}$.*

In particular, if $n \geq \ell + 2m$, then $w_1, \ldots, w_k \in I$.

The first statement follows by the fact that $|z| \geq |\mu|$ and by Remark 1. The proof of the other two statements requires a much more complicated analysis of the creation of the word z_{k+1} by splicing which is omitted in this version of the paper.

3.3 Proof of Theorem 1

Let L be a splicing language and $m = |M_L|$. Throughout this section, by $\sim$ we denote the equivalence relation $\sim_L$ and by $[\,\cdot\,]$ we denote the corresponding equivalence classes $[\,\cdot\,]_L$.

Recall that Theorem 1 claims that the splicing system (I, R) with $I = \Sigma^{<m^2+6m} \cap L$ and

$$R = \left\{ r \in \Sigma^{<m^2+10m} \times \Sigma^{<2m} \times \Sigma^{<2m} \times \Sigma^{<m^2+10m} \;\middle|\; r \text{ respects } L \right\}$$

generates L. The proof is divided in two parts. In the first part, Lemma 7, we proof that the set of rules can be chosen as $\left\{ r \in (\Sigma^{<m^2+10m})^4 \;\middle|\; r \text{ respects } L \right\}$ for some finite set of axioms. The second part concludes the proof of Theorem 1, by employing the length bound $2m$ for the second and third component of rules and by proving that the set of axioms can be chosen as $I = \Sigma^{<m^2+6m} \cap L$.

Lemma 7. *Let L and m as above. There exists $n \in \mathbb{N}$ such that the splicing system (I, R) with $I = \Sigma^{\leq n} \cap L$ and*

$$R = \left\{ r \in (\Sigma^{<m^2+10m})^4 \;\middle|\; r \text{ respects } L \right\}$$

generates the same language $L = L(I, R)$.

Proof. As every word in I belongs to L and every rule in R respects L, the inclusion $L(I, R) \subseteq L$ holds (for any n).

Let (I', R') be a splicing system that generates $L = L(I', R')$ and let n such that $n - 6m$ is larger than any word in I' and larger than any component of a rule in R'. As in the claim, let $I = \Sigma^{\leq n} \cap L$.

For a word μ we let $W_\mu = \{ w \in \Sigma^* \mid w \leq_{\ell\ell} \mu \}$, as we did before. Define the set of rules where every component is length-lexicographically bounded by μ

$$R_\mu = \left\{ r \in W_\mu^4 \;\middle|\; r \text{ respects } L \right\}$$

and the language $L_\mu = L(I, R_\mu)$; clearly, $L_\mu \subseteq L$. For two words $\mu \leq_{\ell\ell} v$ we see that $R_\mu \subseteq R_v$, and hence, $L_\mu \subseteq L_v$. Thus, if $L_\mu = L$ for some word μ, then for all words v with $\mu \leq_{\ell\ell} v$, we have $L_v = L$. As $L = L(I', R')$, there exists a word μ such that $L_\mu = L$ and $|\mu| + 6m \leq n$. Let b be the lexicographically largest letter in Σ. For $\nu = b^{m^2+10m-1}$ the set R_ν contains exactly the rules that respect L and where every component has a length of less than $m^2 + 10m$; therefore, $R_\nu = R$ and if $L_\nu = L$, the claim holds. For the sake of contradiction assume $L_\nu \neq L$ and let μ be the smallest word, in the length-lexicographic order, such that $L_\mu = L$; hence, $|\mu| \geq m^2 + 10m$. Let μ' be the next-smaller word than μ, in the length-lexicographic order, and let $S = R_{\mu'}$. Note that $L(I, S) \subsetneq L$ and $R_\mu \setminus S$ contains only rules which have a component that is equal to μ.

Choose w from $L \setminus L(I, S)$ as a shortest word, i. e., for all $\tilde{w} \in L$ with $|\tilde{w}| < |w|$, we have $\tilde{w} \in L(I, S)$. Factorize $w = xzy$ with $|x| = |y| = 3m$, n. b., $|z| \geq |\mu|$, otherwise $w \in I$. Factorize $\mu = \delta_1\alpha\beta\gamma\delta_2$ with $|\delta_1|, |\delta_2| \geq 5m$, $|\alpha\beta\gamma| = m^2$, $\beta \neq \varepsilon$, $\alpha \sim \alpha\beta$, and $\gamma \sim \beta\gamma$ (Lemma 1).

We will show that there is a series of splicings which creates w from a set of shorter words and by using splicing rules from S. This yields a contradiction to the choice of w. In order to find this series of splicings we investigate the creation of a word $x\tilde{z}y$ where $\tilde{z}$ is derived by using a pumping argument on all factors $\alpha\beta\gamma$ in z.

Let j be a sufficiently large even number ($j > 4|\mu| + |z|$ will suffice). Let $\tilde{z}$ be the word that we obtain by replacing all factors $\alpha\beta\gamma$ by $\alpha\beta^j\gamma$ in z by the following pumping algorithm:

```
1. $\tilde{z} := z$;
2. if there is a factor $\alpha\beta\gamma$ of $\tilde{z}$ such that neither
   (a) the factor $\alpha\beta\gamma$ is a prefix of a factor $\alpha\beta^{j/2}$ in $\tilde{z}$ nor
   (b) the factor $\alpha\beta\gamma$ is a suffix of a factor $\beta^{j/2}\gamma$ in $\tilde{z}$,
   then replace this factor by $\alpha\beta^j\gamma$;
3. repeat step 2 until there is no such factor $\alpha\beta\gamma$ left.
```

A proof that the algorithm will terminate, hence $\tilde{z}$ is well defined, can be found in the arXiv version [15]. The new word $\tilde{z}$ may still contain the factor $\alpha\beta\gamma$, but if it does, then (a) or (b) holds. By induction and as $\alpha\beta\gamma \sim \alpha\beta^j\gamma$, it is easy to see that $\tilde{z} \sim z$.

Let us trace back the creation of $x\tilde{z}y \in L$ by splicing in (I, R_μ) to a word $x_1\tilde{z}y_1$ where either $x_1\tilde{z}y_1 \in I$ or where $x_1\tilde{z}y_1$ is created by a splicing that affects $\tilde{z}$, i. e., the splicing position lies within the factor $\tilde{z}$. Let $z_{k+1} = x_{k+1}\tilde{z}y_{k+1}$, where $x_{k+1} = x$ and $y_{k+1} = y$, be created by k splicings from a word $z_1 = x_1\tilde{z}y_1$ where either $x_1\tilde{z}y_1 \in I$ or $x_1\tilde{z}y_1$ is created by a splicing $(\tilde{w}_1, \tilde{w}_2) \vdash_s z_1$ with $\tilde{w}_1, \tilde{w}_2 \in L$, $s \in R_\mu$, and the splicing position lies in the factor $\tilde{z}$. Furthermore, for $i = 1, \ldots, k$ the intermediate splicings are either

(i) $(w_i, z_i) \vdash_{r_i} x_{i+1}\tilde{z}y_{i+1} = z_{i+1}$, $w_i \in L$, $r_i \in R_\mu$, $y_{i+1} = y_i$, and the splicing position lies on the left of the factor $\tilde{z}$ or

(ii) $(z_i, w_i) \vdash_{r_i} x_{i+1}\tilde{z}y_{i+1} = z_{i+1}$, $w_i \in L$, $r_i \in R_\mu$, $x_{i+1} = x_i$, and the splicing position lies on the right of the factor $\tilde{z}$.

Note that $|\tilde{z}| \geq |z| \geq |\mu|$ and, therefore, we can apply Lemma 6 (with $\ell = 3m$). Thus, $w_i \in I$ and $|x_i|, |y_i| < 5m$ for $i = 1, \ldots, k$.

Consider a rule r_i in a splicing of the form (i). By Lemma 6, $r_i \in \Sigma^{<4m} \times \Sigma^{<2m} \times \Sigma^{<2m} \times W_\mu$. Suppose the fourth component of r_i covers a prefix of the left-most factor $\alpha\beta^{j/2}$ in $\tilde{z}$ which is longer than α (as j is very large, it cannot fully cover $\alpha\beta^{j/2}$). By extension (Lemma 3), we may write $r_i = (u_1, v_1; u_2, \tilde{v}\alpha\beta^e)$ for some $e \geq 1$. By Lemma 4 and as $\alpha \sim \alpha\beta$, we may replace this rule by $(u_1, v_1; u_2, \tilde{v}\alpha)$. Note that, as the fourth component got shorter, now $r_i \in S$.

After we symmetrically treated rules of form (ii), these new rules $r_1, \ldots, r_k$ and the words $w_1, \ldots, w_k$ can be used in order to create $w = x_{k+1}zy_{k+1}$ from x_1zy_1 by splicing. In order to see this, observe that, even though the factors $\alpha\beta\gamma$ in z, which we pumped up before, may overlap with each other, the left-most (and right-most) position where we replaced β by β^j is preceded by the factor α (resp., succeeded by the factor γ) in $\tilde{z}$.

Furthermore, the rules $r_1, \ldots, r_k$ all belong to S. By contradiction, suppose $r_i \notin S$ for some i and, by symmetry, suppose the i-th splicing is of the form (i). Thus, the fourth component of r_i has to be $\mu = \delta_1\alpha\beta\gamma\delta_2$. As $|\delta_1| \geq 5m > |x_i|$, $\alpha\beta\gamma$ is a factor of $\tilde{z}$. The pumping algorithm ensured that (a) the prefix α is succeeded by $\beta^{j/2}$ or (b) the suffix γ is preceded by $\beta^{j/2}$. As $j/2$ is very large and the splicing position is too close to the left end of z_i, case (b) is not possible. Thus, the fourth component of r_i overlaps in more than $|\alpha|$ letters with the left-most factor $\alpha\beta^{j/2}$ in $\tilde{z}$ and we used the replacement above, which ensured $r_i \in S$ — the contradiction.

Let us summarize: If x_1zy_1 was in $L(I, S)$, then $w \in L(I, S)$ as well, which would contradict the choice of w. If $z_1 = x_1\tilde{z}y_1 \in I$, then x_1zy_1, which is at most as long as z_1, would belong to I and we are done. We only have to consider the case when $(\tilde{w}_1, \tilde{w}_2) \vdash_s z_1 = x_1\tilde{z}y_1$ and the splicing position lies within the factor $\tilde{z}$. We will show that, from this splicing, we derive another splicing $(\hat{w}_1, \hat{w}_2) \vdash_t x_1zy_1$ which respects $L(I, S)$ and, therefore, yields the contradiction.

Let $s = (u, v_1; u_2, v)$, $\tilde{w}_1 = xuv_1$ and $\tilde{w}_2 = u_2vy$ where $|v_1|, |u_2| < m$, by Lemma 5 (here, x and y are newly chosen words). We have

$$z_1 = x_1\tilde{z}y_1 = xuvy$$

where xu is a proper prefix of $x_1\tilde{z}$ and vy is a proper suffix of $\tilde{z}y_1$.

We will see next that if $s \notin S$, then we can use a rule $\tilde{s} \in S$ and maybe slightly modified words in order to obtain z_1 by splicing. If $s \notin S$, then $u = \mu$ or $v = \mu$. Suppose $u = \mu = \delta_1\alpha\beta\gamma\delta_2$. Thus, $\alpha\beta\gamma$ is a factor of $\tilde{z}$, as $|\delta_1| \geq 5m > |x_1|$, and, as such, either (a) α is succeeded by $\beta^{j/2}$ or (b) γ is preceded by $\beta^{j/2}$. If (b) holds, $\delta_1\alpha$ is a suffix of a word in β^+. We may write $\delta_1\alpha = \beta_2\beta^\ell$ where $\ell \geq 0$ and β_2 is a suffix of β. Replace u by $\beta_2\gamma\delta_1$ and use this new rule $\tilde{s}$ in order to splice $(\tilde{w}_1, \tilde{w}_2) \vdash_{\tilde{s}} z_1$. Note that the first component is now shorter than μ. Otherwise, (a) holds and $\gamma\delta_2v$ is a prefix of a word in β^+. As j is very large and γ is a prefix of a word in β^+, we may extend v (Lemma 3) such that we can write $\beta\gamma\delta_2 = \beta^{\ell_1}\beta_1$ and $v = \beta_2\beta^{\ell_2}\gamma$ where $\ell_1 \geq 1$, $\ell_2 \geq 0$ and $\beta_1\beta_2 = \beta$. Now, we pump down one of the β in the first component and β^{ℓ_2} in the fourth component and we let $\tilde{s} = (\delta_1\alpha\beta^{\ell_1-1}\beta_1, v_1; u_2, \beta_2\gamma) \sim s$. As both components are shorter than μ, we see that $\tilde{s} \in S$ and

$$(x\delta_1\alpha\beta^{\ell_1-1}\beta_1 v_1, u_2\beta_2\beta^{\ell_2+1}\gamma y) \vdash_{\tilde{s}} z_1,$$

i. e., we have shifted one of the occurrences of β from $\tilde{w}_1$ to $\tilde{w}_2$. Note that $\beta_2\gamma$ is a prefix of $\beta_2\beta^{\ell_2+1}\gamma$. Treating the fourth component analogously justifies the assumption that $s \in S$.

Next, we will pump down the factors $\alpha\beta^j\gamma$ to $\alpha\beta\gamma$ in $\tilde{z}$ again. At every position where we pumped up before, we are now pumping down (in reverse order) in order to obtain the words $\hat{x}, \hat{u}, \hat{v}, \hat{y}$ from the words x, u, v, y, respectively. For each pumping step:

If u is covered by the factor $\alpha\beta^j\gamma$ (which we pump down in this step), extend u to the left such that it becomes a prefix of $\alpha\beta^j\gamma$. Symmetrically, if v is covered by the factor $\alpha\beta^j\gamma$, extend v to the right such that it becomes a suffix of $\alpha\beta^j\gamma$ (Lemma 3). Observe that extension ensures that the factor $\alpha\beta^j\gamma$ is covered by either xu, uv, or vy. If $\alpha\beta^j$ or $\beta^j\gamma$ is fully covered by one of x, u, v, or y, then replace this factor by $\alpha\beta$ or $\beta\gamma$, respectively. Otherwise, let us show how to pump when $\alpha\beta^j\gamma$ is covered by xu. The cases when $\alpha\beta^j\gamma$ is covered by uv or vy can be treated analogously. We can factorize

$$x = \tilde{x}\alpha\beta^{j_1}\beta_1, \qquad\qquad u = \beta_2\beta^{j_2}\gamma\tilde{u}$$

where $\beta_1\beta_2 = \beta$ and $j_1 + j_2 + 1 = j$. The pumping result are the words $\tilde{x}\alpha\beta_1$ and $\beta_2\gamma\tilde{u}$, respectively.

Observe that, after reversing all pumping steps, $\hat{x}\hat{u} \sim xu$, $\hat{v}\hat{y} \sim vy$, $\hat{x}\hat{u}\hat{v}\hat{y} = x_1zy_1$, and the rule $t = (\hat{u}, v_1; u_2, \hat{v})$ respects L. Furthermore, if we used extension for u (or v) in one of the steps, then $|\hat{u}| \le m^2$ (resp., $|\hat{v}| \le m^2$); in any case $t \in S$. Recall that w was chosen as the shortest word from $L \setminus L(I, S)$. As $|\hat{x}\hat{u}v_1|, |u_2\hat{v}\hat{y}| < |z| + 6m = |w|$, the words $\hat{x}\hat{u}v_1$ and $u_2\hat{v}\hat{y}$ belong to $L(I, S)$, and as $(\hat{x}\hat{u}v_1, u_2\hat{v}\hat{y}) \vdash_t x_1zy_1$, we conclude that x_1zy_1 as well as w belong to $L(I, S)$ — the desired contradiction. □

Now, let us outline how the proof of Theorem 1 can be concluded. The full proof can be found in the arXiv version [15].

For a splicing language L with $m = |M_L|$ we intend to prove that the splicing system (I, R) with $I = \Sigma^{<m^2+6m} \cap L$ and

$$R = \left\{ r \in \Sigma^{<m^2+10m} \times \Sigma^{<2m} \times \Sigma^{<2m} \times \Sigma^{<m^2+10m} \;\middle|\; r \text{ respects } L \right\}$$

generates the language $L = L(I, R)$. By Lemma 7, we may assume that L is generated by a splicing system (J, S) where

$$S = \left\{ r \in (\Sigma^{<m^2+10m})^4 \;\middle|\; r \text{ respects } L \right\}.$$

In order to prove $L \subseteq L(I, R)$, we use induction on the length of words in L. For $w \in L$ with $|w| < m^2 + 6m$, by definition, $w \in I \subseteq L(I, R)$.

For $w \in L$ with $|w| \ge m^2 + 6m$, the induction hypothesis states that every word $\tilde{w} \in L$ with $|\tilde{w}| < |w|$ belongs to $L(I, R)$. Factorize $w = x\alpha\beta\gamma\delta y$ such that $|x| = |y| = 3m$, $|\alpha\beta\gamma| = m^2$, $\beta \ne \varepsilon$, $\alpha \sim \alpha\beta$, and $\gamma \sim \beta\gamma$.

The proof idea is similar to the idea in the proof of Lemma 7, but this time we are using induction instead of a proof-by-contradiction. We use a pumping argument on β in order to obtain a very long word $\hat{w}$. This word has to be created by a series of splicings in (J, S). Due to Lemma 6 these splicings can be modified in order to create $\hat{w}$ by splicing from a set of strictly shorter words and with rules from R. Just like in the proof of Lemma 7, almost the same words and rules can be used in order to create w from a set of strictly shorter words and with rules from R. Then, the induction hypothesis yields $w \in L(I, R)$.

4 Conclusion and Final Remarks

The main question we intended to answer when starting our investigation was, if it is decidable whether a given regular language L is a splicing language. If we can decide whether a splicing rule respects a regular language and if we can construct a (non-deterministic) finite automaton accepting the language generated by a given splicing system, then we can decide whether L is a splicing language as follows. We compute the splicing system (I, R) as given in Theorem 1, we compute a finite automaton accepting the splicing language $L(I, R)$, and we test whether $L(I, R)$ equals to L. Recall that Theorem 1 implies that L is a splicing language if and only if $L = L(I, R)$ and that equivalence of regular languages is decidable [14]. It is known from [8, 13] that it is decidable whether a classic splicing rule respects a regular language. Furthermore, there is an effective construction of a finite automaton which accepts the language generated by a splicing system [12,16]. These observations lead to the following decidability result.

Corollary 1. *For a given regular language L, it is decidable whether or not L is a splicing language. Moreover, if L is a splicing language, a splicing system (I, R) generating L can be effectively constructed.*

Another variant of splicing has been defined by Pixton in [16]. Pixton's variant of splicing can be seen as more general than the classical splicing, which we investigated in this paper, because every classical splicing rule can easily be translated into a Pixton splicing rule, but not the other way around. Actually, the class of classical splicing languages is strictly included in the class of Pixton splicing languages [4]. In the online version of our paper [15] we also prove that if a regular language L is a Pixton splicing language, then it is generated by one particular Pixton splicing system whose size is a function of the size of the syntactic monoid of L. A decidability result, analogous to Corollary 1, follows immediately.

As final remarks, note that it has been known since 1991 that the class $\mathcal{S}$ of languages that can be generated by a splicing system is a proper subclass of the class of regular languages. However, to date, no other natural characterization for the class $\mathcal{S}$ exists. The problem of deciding whether a regular language is generated by a splicing system is a fundamental problem in this context and has remained unsolved. To the best of our knowledge, the problem was first stated in the literature in 1998 [11]. In this paper we solved this long standing open problem.

References

1. Bonizzoni, P.: Constants and label-equivalence: A decision procedure for reflexive regular splicing languages. TCS 411(6), 865–877 (2010)
2. Bonizzoni, P., de Felice, C., Zizza, R.: The structure of reflexive regular splicing languages via Schützenberger constants. TCS 334(1-3), 71–98 (2005)
3. Bonizzoni, P., de Felice, C., Zizza, R.: A characterization of (regular) circular languages generated by monotone complete splicing systems. TCS 411(48), 4149–4161 (2010)
4. Bonizzoni, P., Ferretti, C., Mauri, G., Zizza, R.: Separating some splicing models. Inf. Process. Lett. 79(6), 255–259 (2001)
5. Bonizzoni, P., Jonoska, N.: Regular Splicing Languages Must Have a Constant. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 82–92. Springer, Heidelberg (2011)
6. Culik II, K., Harju, T.: Splicing semigroups of dominoes and DNA. Discrete Applied Math. 31(3), 261–277 (1991)
7. Gatterdam, R.W.: Splicing systems and regularity. International Journal of Computer Mathematics 31(1-2), 63–67 (1989)
8. Goode, E.: Constants and Splicing Systems. PhD thesis, Binghamton University (1999)
9. Goode, E., Pixton, D.: Recognizing splicing languages: Syntactic monoids and simultaneous pumping. Discrete Applied Math. 155(8), 989–1006 (2007)
10. Head, T.: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. Bull. of Math. Bio. 49(6), 737–759 (1987)
11. Head, T.: Splicing languages generated with one sided context. In: Păun, G. (ed.) Computing With Bio-molecules: Theory and Experiments, pp. 269–282. Springer (1998)
12. Head, T., Pixton, D.: Splicing and Regularity. In: Ésik, Z., Martín-Vide, C., Mitrana, V. (eds.) Recent Advances in Formal Languages and Applications. SCI, pp. 119–147. Springer, Heidelberg (2006)
13. Head, T., Pixton, D., Goode, E.: Splicing Systems: Regularity and *Below*. In: Hagiya, M., Ohuchi, A. (eds.) DNA 2002. LNCS, vol. 2568, pp. 262–268. Springer, Heidelberg (2003)
14. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
15. Kari, L., Kopecki, S.: Deciding whether a regular language is generated by a splicing system. CoRR, abs/1112.4897 (2011)
16. Pixton, D.: Regularity of splicing languages. Discrete Applied Math. 69(1-2), 101–124 (1996)
17. Păun, G.: On the splicing operation. Discrete Applied Math. 70(1), 57–79 (1996)
18. Schützenberger, M.P.: Sur certaines opérations de fermeture dans le langages rationnels. Symposia Mathematica 15, 245–253 (1975)

Probabilistic Reasoning with a Bayesian DNA Device Based on Strand Displacement

Iñaki Sainz de Murieta and Alfonso Rodríguez-Patón

Departamento de Inteligencia Artificial,
Universidad Politécnica de Madrid (UPM),
Campus de Montegancedo s/n, Boadilla del Monte 28660 Madrid, Spain
inaki.sainzdemurieta@upm.es, arpaton@fi.upm.es

Abstract. We present a computing model based on the DNA strand displacement technique which performs Bayesian inference. The model will take single stranded DNA as input data, representing the presence or absence of a specific molecular signal (evidence). The program logic encodes the prior probability of a disease and the conditional probability of a signal given the disease playing with a set of different DNA complexes and their ratios. When the input and program molecules interact, they release a different pair of single stranded DNA species whose relative proportion represents the application of Bayes' Law: the conditional probability of the disease given the signal. The models presented in this paper can empower the application of probabilistic reasoning in genetic diagnosis in vitro.

1 Introduction

Since the birth of biomolecular computation in Leonard Adleman's seminal work [2], different applications have been proposed in the literature. The trend of resolving NP-complete problems during the early years of the discipline [11] progressively evolved towards nanotechnology and biomedicine oriented applications, such as genetic diagnosis and drug delivery automata [5,3,1,4].

An important research line emerged taking advantage of the DNA *strand displacement* phenomenon, which in short can be described as follows: a strand A displaces another strand B from a complex $A'B$, due to the higher affinity between A and A' and the greater stability of the duplex AA'. We cite only a few contributions to this extensive topic introduced by Yurke et al. [23], like for example the design of logic gates [19,21,9], DNA automata [22] and theoretical models [6].

The interest in molecular logical inference was reawakened in 2009 with the work presented by the group of Prof. Shapiro [15], where the authors developed an enzyme driven system able to perform autonomously simple logical deductions with DNA molecules. Since then, Rodríguez-Patón et al. [16,17,13] have been working on the design of enzyme free logical inference models that only exploit the DNA strand displacement operation.

With the exception of the work done by the group of Prof. Benenson about stochastic enzymatic reactions [1,4], all the logical models cited above share a common property: they only implement Boolean logic, and thus their output always represent an absolute truth value (true / false, active / inactive, presence / absence, 1 / 0, etc.). None of these deterministic models is able to deal with uncertain knowledge. Other enzyme free models (but not autonomous) have been presented implementing stochastic paradigms [10,24].

Probabilistic reasoning can be used when we want to consider diagnostic accuracy or uncertainty of tests in our clinical decisions (i.e. classic systems like Mycin [20]). With the motivation of designing a model that can process this uncertainty, this article presents a Bayesian biosensor that makes probabilistic reasoning and whose output represents the probability (value between 0 and 1) of a disease. Such type of device can be used to estimate and update the probability of a certain diagnose based in the light of new evidence, i.e., based on the presence or absence of a new specific signal (or set of signals). The DNA sensor device would encode two different probabilities as program data: the conditional probability of the signal given the disease ($P(signal|disease)$) and the *prior* probability of the disease ($P(disease)$). Then, when the sensor interacts with an input representing the evidence of a signal (its presence or absence), *Bayes' Law* would be autonomously computed by means of strand displacement cascades, releasing a set of DNA species whose ratio encodes the *posterior* probability of the disease given the input ($P(disease|signal)$).

The rest of the paper is structured as follows: Section 2 includes a brief review of the main concepts in probability theory and Bayesian inference. Section 3 describes how the model encodes the prior and conditional probabilities, as well as the input evidences. Section 4 shows an inference process example that updates the knowledge of a disease applying the Bayes' rule, and how it is implemented by our model. Section 5 discusses in detail the scalability and the mapping of the biological evidences as inputs to the system. Finally, Section 6 summarizes the conclusions and future work.

2 Principles of the Model

Basic concepts of probability theory and Bayesian inference [14,18] used throughout the article are summarized first:

Random variable. A function whose possible values are numerical outcomes of a random phenomenon. It can take different value domains, so that we can talk about continuous, discrete, or Boolean random variables. This paper will focus on boolean random variables, that can take the value true or false with a certain probability. For example, we can talk about the random variable D representing a given disease, which can be present (true) or absent (false).

Logical proposition. A logical formula expressing an assignment between a random variable and one of its potential domain values. Hence, the propositions $D = present$ (also denoted as D_1) and $D = absent$ (also denoted as D_0) are the possible formulations that can be hypothesized on the random variable D. Generic propositions of a given variable are denoted with its corresponding lower case letter, for example, $P(d)$ can refer either to $P(D_1)$ or $P(D_0)$.

Probability function. A function P that assigns a probability to each value in the random variable domain (and thus to each potential logical proposition derived from the variable). Building on the above example, we can talk about the probability of D as the duple $P(d) = \langle P(D_1), P(D_0)\rangle$. The sum of probabilities of all the values of the domain must be equal to 1:

$$P(D_1) + P(D_0) = 1 \tag{1}$$

When this function is defined without any dependence on other random variables, we call it *prior probability.*

Joint probability. Having a set of different propositions, a_1, ..., a_n, the probability of all of them happening at the same time is defined by the joint probability function, represented as $P(a_1 \wedge ... \wedge a_n)$ or $P(a_1, ..., a_n)$.

Conditional probability. This function can be intuitively seen as the degree of belief in a variable after the observation of other variables related to the first. So the conditional probability of a proposition a given b is the probability of a when b is known to occur. It is commonly denoted as $P(a|b)$. Conditional probability can also be expressed as a function of prior and joint probabilities:

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} \tag{2}$$

This formula can be derived into the so called *product rule*:

$$P(a \wedge b) = P(a|b) \cdot P(b) = P(b|a) \cdot P(a) \tag{3}$$

Continuing the above example, when a disease is extensively studied, the probability of a disease d given the signal s is known and expressed as $P(d|s)$. This is also called *posterior probability.*

Conditional independence. Two propositions a and b are conditionally independent when they do not have any dependency relationship. In such case we can rewrite their probabilities as

$$P(a|b) = P(a); \quad P(b|a) = P(b); \quad P(a,b) = P(a) \cdot P(b) \tag{4}$$

Bayes' Law Can be derived from the conditional probability and the product rule formulations, and is stated as follows:

$$P(d|s) = \frac{P(s|d) \cdot P(d)}{P(s)} \tag{5}$$

This rule, together with the property of independence, are key in probabilistic reasoning and allows the establishment of relationships between probabilities and evidences. It allows to update the certainty value of a hypothesis or a diagnosis (prior probability $P(d)$), in the light of new evidence ($P(s)$) and the signal likelihood ($P(s|d)$), to obtain an "updated" posterior probability ($P(d|s)$).

Assuming we are able to exhaustively estimate all the probabilities concerning the variable D, we can rewrite the law as:

$$P(d|s) = \alpha \cdot P(s|d) \cdot P(d) = \alpha \cdot P(d \wedge s) \tag{6}$$

Since the sum of the probabilities $P(D = \mathit{present}|s)$ and $P(D = \mathit{absent}|s)$ must be equal to 1, we can treat α as a normalization factor.

3 Encoding

Our sensor model aims to implement the product of the probabilities $P(s|d) \cdot P(d) = P(d \wedge s)$ shown in Equation 6. This will be achieved using single stranded DNA in the encoding of the prior probabilities ($P(d)$) and double stranded complexes in the encoding of conditional probabilities ($P(s|d)$). Also the input evidences will need a specific DNA encoding. Details come below:

Encoding Input Evidences

Input evidences are encoded using single stranded DNA. A strand S_1 represents the presence of the signal, while S_0 represents its absence. As we talk about evidences, only one specie can be present at a time: either only input strands S_1 (meaning the signal is present) or S_0 (meaning the signal is not present). This input will tell the sensor that the prior probability of the disease needs to be updated according to the given evidence.

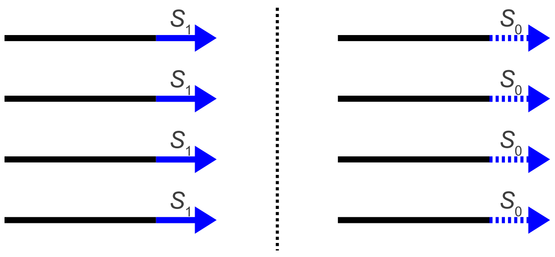


Fig. 1. Encoding input evidences. S_1 represents the presence of the signal; S_0 represents its absence. Only S_1 or S_0 species should be present at the same time.

In case of unwanted presence of one of the two signals in significant concentration, the input could not any more be considered as an evidence as the probabilities P_i ($i = 0, 1$) it would be different of 1 or 0. Thus the computation result would be altered and not valid.

Encoding Prior Probabilities

The prior probability of D is represented as the duple $P(d) = \langle P(D_1), P(D_0)\rangle$. Our model encodes each value using two different single stranded species: D_1 representing $P(D = \textit{present})$ and D_0 representing $P(D = \textit{absent})$. The probability values are implicitly encoded in the ratio of molecules of each specie against the total for D. If we denote the number of molecules of each specie A_i as $|A_i|$, we can express the probability as $P(d) = \left\langle \frac{|A_1|}{|A_1|+|A_0|}, \frac{|A_0|}{|A_1|+|A_0|} \right\rangle$. The Figure 2 shows an example DNA encoding of $P(d) = \langle 0.5, 0.5\rangle$. The coloured toeholds at the 5' end will allow their interaction with the molecules encoding the conditional probabilities.

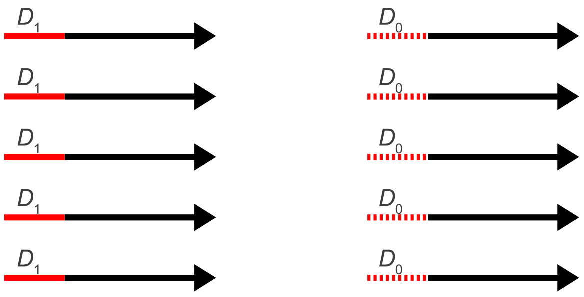


Fig. 2. Encoding prior probabilities. The model encodes each value using two different single stranded species: D_1 representing $P(D = \textit{present})$ and D_0 representing $P(D = \textit{absent})$. The probability $P(d) = \langle 0.5, 0.5\rangle$ is encoded as the ratio between the number of molecules of each specie and the total number of species for D.

Encoding Conditional Probabilities

The conditional probability of S given D needs to encode values for the following propositions:

- $(S = \textit{present}|D = \textit{present})$
- $(S = \textit{absent}|D = \textit{present})$
- $(S = \textit{present}|D = \textit{absent})$
- $(S = \textit{absent}|D = \textit{absent})$

The reader can see that for each proposition strand d interacting with a conditional probability molecule, two different outputs encoding two different joint probabilities can be released: $(S = \textit{present} \wedge d)$ and $(S = \textit{absent} \wedge d)$. Therefore, from the four different joint probabilities that could be released from the interaction of the D_i species and the conditional probability molecules (representing $(S = \textit{present} \wedge D = \textit{present})$, $(S = \textit{absent} \wedge D = \textit{present})$, $(S = \textit{present} \wedge D = \textit{absent})$ and $(S = \textit{absent} \wedge D = \textit{absent})$), the system needs to be able to select only the outputs corresponding to the input evidence:

- If the input evidence is S_1, the output strands released should encode ($S = \mathit{present} \wedge D = \mathit{present}$) and ($S = \mathit{present} \wedge D = \mathit{absent}$).
- If the input evidence is S_0, the output strands released should encode ($S = \mathit{absent} \wedge D = \mathit{present}$) and ($S = \mathit{absent} \wedge D = \mathit{absent}$)).

The desired behaviour described above for the conditional probability molecules can be attained using a motif equivalent to the AND gate presented by Seelig et al. [19]. Other motifs implementing such logic could be equally valid, but we have chosen this one due to its simplicity and iteration capability. Figure 3-A shows an example of how the strands building the joint probability $P(S = \mathit{present} \wedge D = \mathit{present})$ (depicted as $S_1 \wedge D_1$) are released in the presence of the input evidence S_1 and D_1. Figure 3-B shows the detailed motifs of the molecules that encode the conditional probabilities $P(s|d)$. Similarly to the case of prior probabilities, the probability figures are taken as ratios relating to the number of molecules of each motif. It also shows the formula to establish the correspondence between each motif and the conditional probability values they encode.

In order to ensure we are working with probability values, the following restrictions need to be ensured:

- $\frac{|S_0|D_0|}{|S_0|D_0|+|S_1|D_0|} + \frac{|S_1|D_0|}{|S_0|D_0|+|S_1|D_0|} = 1$ (derived from Equation 1)
- $\frac{|S_0|D_1|}{|S_0|D_1|+|S_1|D_1|} + \frac{|S_1|D_1|}{|S_0|D_1|+|S_1|D_1|} = 1$ (derived from Equation 1)
- $|S_0|D_0| + |S_1|D_0| = |S_0|D_1| + |S_1|D_1|$ (this ensures that different $|S_i|D_j|$ can be mixed in the output for a fixed i)

4 Inference Process

Let us imagine we need to diagnose a disease D with the help of its signal S. The following data is known upfront, due to empirical data:

- Prior probability of the disease:
  - $P(D = \mathit{present}) = 0.5$
  - $P(D = \mathit{absent}) = 0.5$
- Conditional probability of the signal given the disease:
  - $P(S = \mathit{absent}|D = \mathit{absent}) = 0.7$
  - $P(S = \mathit{present}|D = \mathit{absent}) = 0.3$
  - $P(S = \mathit{absent}|D = \mathit{present}) = 0.2$
  - $P(S = \mathit{present}|D = \mathit{present}) = 0.8$

Now we get the confirmation that the signal is present ($S = \mathit{present}$). What is now the probability of the disease being present given that the signal is present, $P(D = \mathit{present}|S = \mathit{present})$? Since we don't know the prior probability of the signal ($P(s)$), we cannot directly apply the Bayes' Law as stated in Equation 5. We apply the derivation stated in Equation 6 instead:

$P(D = \mathit{present}|S = \mathit{present}) = \alpha \cdot P(S = \mathit{present}|D = \mathit{present}) \cdot P(D = \mathit{present}) = \alpha \cdot 0.8 \cdot 0.5 = \alpha \cdot 0.4$

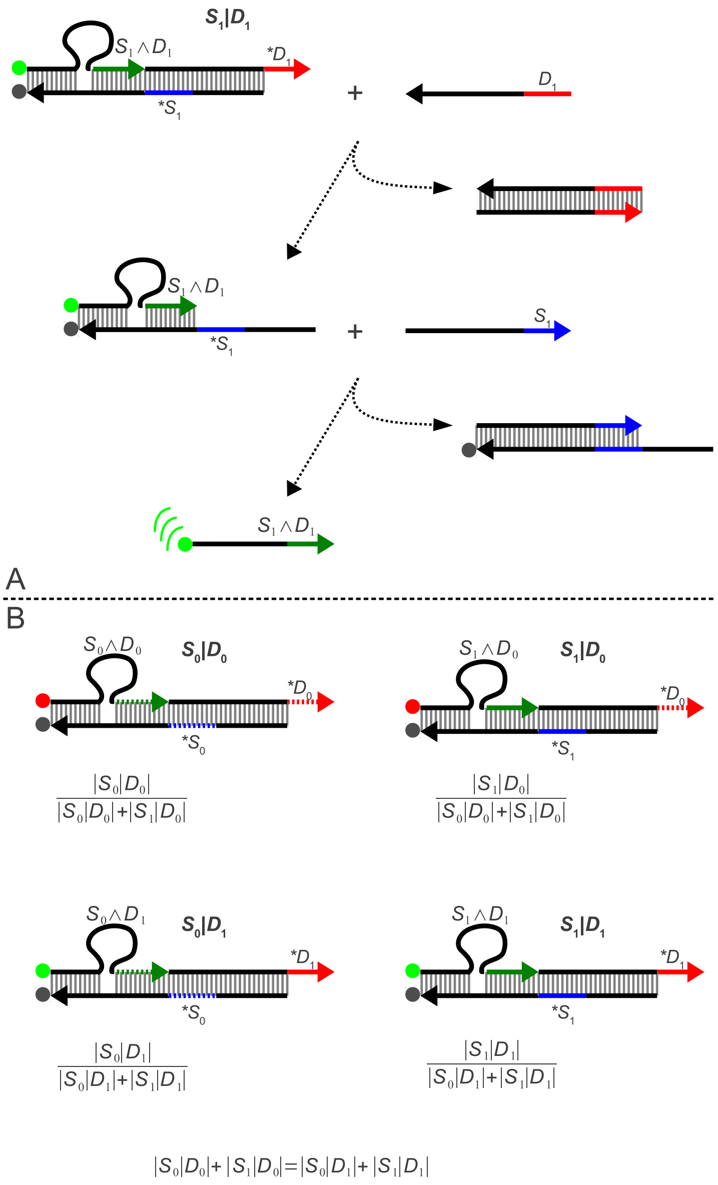


Fig. 3. Encoding conditional probabilities. (A) The top panel shows the hybridization flow releasing the joint probability strand $P(S = present \wedge D = present)$ (depicted as $S_1 \wedge D_1$) when the input evidence $S = present$ (depicted as S_1) and $D = present$ (depicted as D_1). It also releases a fluorophore from its quencher, which will allow the measurement of the output. DNA segments named with an asterisk prefix ($^*S_1$ and $^*D_1$ in this panel) are Watson-Crick complementary to the corresponding toeholds named without that asterisk. (B) The bottom panel shows the four motifs encoding the conditional probabilities $P(s|d)$, together with the formulas that relate their concentrations to the respective probability values.

In order to find α, we need to calculate $P(D = \mathit{absent}|S = \mathit{present})$ as well:

$P(D = \mathit{absent}|S = \mathit{present}) = \alpha \cdot P(S = \mathit{present}|D = \mathit{absent}) \cdot P(D = \mathit{absent}) = \alpha \cdot 0.3 \cdot 0.5 = \alpha \cdot 0.15$

Since $P(D = \mathit{present}|S = \mathit{present}) + P(D = \mathit{absent}|S = \mathit{present}) = 1$ (see Equation 1) we can derive $\alpha = 1.81$ and $P(D = \mathit{present}|S = \mathit{present}) = 0.73$).

Following the encoding model described in Section 3, we can reproduce with DNA the inference process described above:

- The prior probability $P(d) = \langle 0.5, 0.5 \rangle$ is encoded with two different DNA species, D_1 and D_0. Each specie will count 50 copies ($|D_1| = 50$, $|D_0| = 50$).
- The conditional probabilities are encoded as follows:
  - $P(S = \mathit{absent}|D = \mathit{absent})$ is encoded with 70 copies of the complex $S_0|D_0$.
  - $P(S = \mathit{present}|D = \mathit{absent}) = 0.3$ is encoded with 30 copies of the complex $S_1|D_0$.
  - $P(S = \mathit{absent}|D = \mathit{present}) = 0.2$ is encoded with 20 copies of the complex $S_0|D_1$.
  - $P(S = \mathit{present}|D = \mathit{present}) = 0.8$ is encoded with 80 copies of the complex $S_1|D_1$.
- The input evidence $S = \mathit{present}$ is encoded with a unique DNA specie, S_1, with a number of copies much bigger than the total number of molecules encoding conditional probabilities.

Then the DNA inference process would start by mixing evidences, prior and conditional probabilities all together (see Figure 4):

1. The strands D_j interact with the strands $S_i|D_j$. Assuming an ideal solution (perfectly mixed), the number of complexes $S_i|D_j$ "activated" by strands D_j would be updated as follows:
   - 35 copies of the complex $S_0|D_0$.
   - 15 copies of the complex $S_1|D_0$.
   - 10 copies of the complex $S_0|D_1$.
   - 40 copies of the complex $S_1|D_1$.
2. The input strands S_1 interact with the complexes $S_1|D_0$ and $S_1|D_1$, releasing 15 copies of the strand $S_1 \wedge D_0$ and 40 copies of the strand $S_1 \wedge D_1$.
3. The number of copies of each output strand is estimated by the increase of the different fluorescent colours (red for $S_1 \wedge D_0$ and green for $S_1 \wedge D_1$). The only step missing is the calculation of the probability encoded in that output, which is easily done normalizing both values as follows: $P(D|S) = \left\langle \frac{|S_1|D_1|}{|S_1|D_0|+|S_1|D_1|}, \frac{|S_1|D_0|}{|S_1|D_0|+|S_1|D_1|} \right\rangle = \langle 0.73, 0.27 \rangle$.

5 Discussion

The DNA biosensor presented here operates as a Bayesian inference device, which allows the introduction of quantitative information in the tests, highlighted by the molecular indicators or signals.

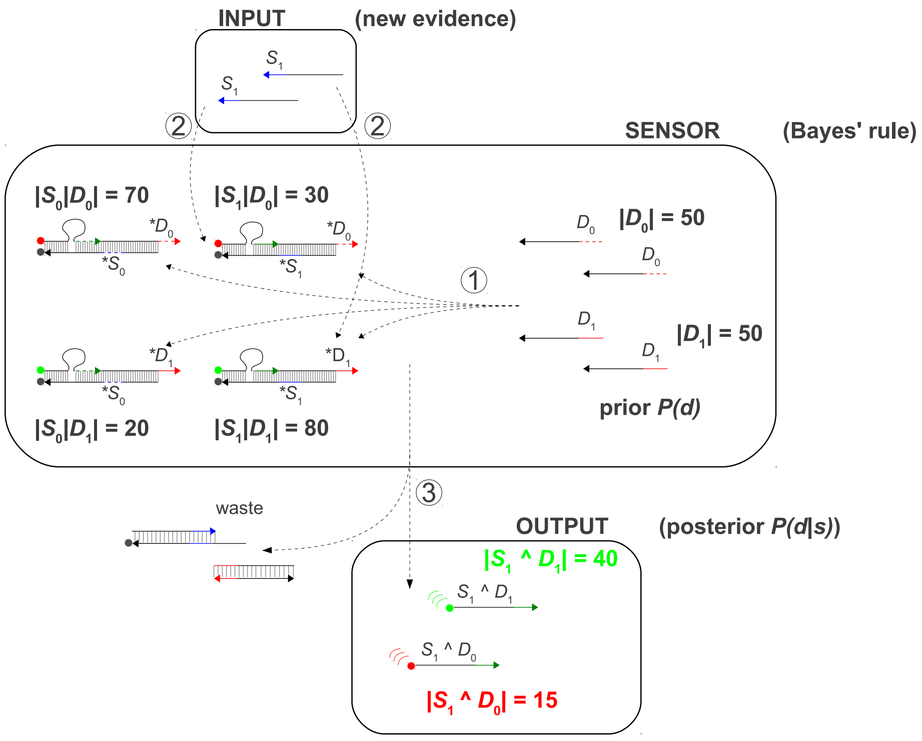


Fig. 4. Inference process. DNA segments named with an asterisk prefix are Watson-Crick complementary to the corresponding toeholds named without that asterisk. (1) The DNA probabilistic reasoning starts with the prior probability species D_i interacting with the conditional probability molecules $S_i|D_j$. (2) Then the input evidence species S_1 interact with the molecules $S_i|D_j$ that had interacted previously with the species D_i, releasing the strand species $S_1 \wedge D_1$ and $S_1 \wedge D_0$. (3) Finally, the probability $P(D|S = \textit{present})$ is inferred by normalization of the red and green fluorescence emissions.

Its operation is inspired by the stochasticity of the competing transitions presented by Adar et al.[1], also used by the authors in another work to modulate the ratio between the drug and the drug suppressor in the output [4]. Our work is also based on transitions competing stochastically, but using the DNA strand displacement operation to eliminate the dependence on the enzyme Fok I. Moreover, we have aimed to identify the way to map the basic concepts of probability theory and Bayesian inference and map them into DNA strand displacement motifs, so that they can be used as design patterns when implementing Bayesian reasoning with DNA.

In order this model to have realistic applications in genetic diagnosis, it needs to deal with more than one signal $(s^1, ..., s^n)$ for the same disease d (superscripts denote the signal number). According to Equation 6, the following formulation of the Bayes' Law would need to be solved: $P(d|s^1, ..., s^n) = \alpha \cdot P(d) \cdot P(s^1, ..., s^n|d)$. Assuming conditional independence of the signals given the disease (as in the Naïve Bayes model [12]) we can apply Equation 4 and derive the following expression: $P(d|s^1, ..., s^n) = \alpha \cdot P(d) \cdot P(s^1|d) \cdot ... \cdot P(s^n|d)$. We can see that the first two terms of the product (ignoring α) correspond to the formulation we have used for just one signal. Substituting those terms by application of Equation 2 we get this final expression: $P(d|s^1, ..., s^n) = \alpha \cdot P(s^1 \wedge d) \cdot ... \cdot P(s^n|d)$. Translating this into our DNA encoding model:

- $P(d)$, $P(s^1)$ and the evidences S_i^1 $(i = 0, 1)$ are encoded as described in Sections 3 and 4, with subscripts denoting absence (0) or presence (1) of signal.
- Since the output of the previous steps has the form of species $S_i^1 \wedge D_k$ $(i = 0, 1;\ k = 0, 1)$, the devices encoding $P(s^2|d)$ will need to accept the strands $S_i^1 \wedge D_k$ instead of the strands D_k. The output of this step will release species $S_i^1 \wedge S_j^2 \wedge D_k$.
- Previous step would be repeated for each $P(s^x|d)$ $(x = 1, ..., n)$ until the last signal is reached.

Another important matter to be addressed is the translation of the biological data coming from real samples into the input evidence species. When the evidence to be sensed is determined by the presence of a specific nucleic acid strand, that strand could be directly taken as the input evidence strand S_1 the system expects. The problem comes if the signal is determined by the absence of a specific nucleic acid strand. How can that be mapped into an input evidence strand S_0? One possible solution is the addition of an extra "pre-processing" layer, consisting of extra DNA device as described in Figure 5: if no input signal from the samples, the device in the pre-processing layer works as being the input strand S_0; but if an input signal is present, it unreleases the strand S_1. Another potential solution would be the use of a DNA aptamer [25]. Other non DNA aptamers [8] could also be exploited to allow our model take other molecules different from nucleic acids as inputs.

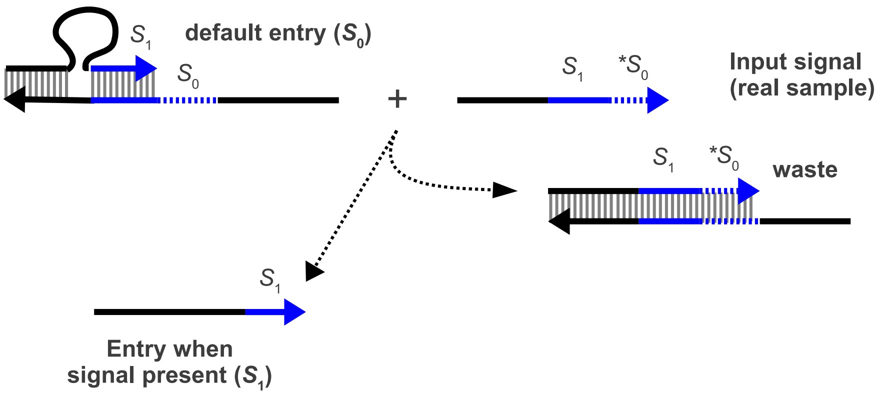


Fig. 5. Processing input as absence of DNA strand. If no signal available among the input samples, the device in the pre-processing layer works as being the input strand S_0 (top-left). However, if an input signal is present (top-right), it unreleases the strand S_1 (bottom-left), leaving a waste molecule that will not react in the system.

6 Conclusions and Future Works

We have introduced a new DNA model for realization of Bayesian inference. The model is completely autonomous, enzyme-free and it is based on DNA strand displacement techniques. Its implementation can be based on experimentally verified and general design derived in [19]. According to the properties examined in [7], the model can be characterized as partly scalable, time-responsive and energy-efficient.

We think the models presented in this paper can empower new quantitative applications of probabilistic genetic diagnosis in vitro. We plan to construct the model in a wet lab and to continue enhancing this model, so that it can be generalized to work with all types of Bayesian networks (and not only the ones following the Naïve Bayes approach [12]).

Acknowledgments. Research was partially supported by project BACTOCOM funded by grant from the European Commission under the VII Framework Programme (FET Proactive area), by Comunidad de Madrid and UPM, and by Spanish Ministerio de Ciencia e Innovación under project TIN2009-14421.

References

1. Adar, R., Benenson, Y., Linshiz, G., Rosner, A., Tishby, N., Shapiro, E.: Stochastic computing with biomolecular automata. Proceedings of the National Academy of Sciences of the United States of America 101(27), 9960–9965 (2004)

2. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science 266(5187), 1021–1024 (1994)
3. Benenson, Y., Adar, R., Paz-Elizur, T., Livneh, Z., Shapiro, E.: DNA molecule provides a computing machine with both data and fuel. Proc. Natl. Acad. Sci. USA 100(5), 2191–2196 (2003)
4. Benenson, Y., Gil, B., Ben-Dor, U., Adar, R., Shapiro, E.: An autonomous molecular computer for logical control of gene expression. Nature 429, 423–429 (2004)
5. Benenson, Y., Paz-Elizur, T., Adar, R., Keinan, E., Livneh, Z., Shapiro, E.: Programmable and autonomous computing machine made of biomolecules. Nature 414(6862), 430–434 (2001)
6. Cardelli, L.: Strand Algebras for DNA Computing. In: Deaton, R., Suyama, A. (eds.) DNA 15. LNCS, vol. 5877, pp. 12–24. Springer, Heidelberg (2009)
7. Chiniforooshan, E., Doty, D., Kari, L., Seki, S.: Scalable, Time-Responsive, Digital, Energy-Efficient Molecular Circuits Using DNA Strand Displacement. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 25–36. Springer, Heidelberg (2011)
8. Cho, E.J., Lee, J.W., Ellington, A.D.: Applications of Aptamers as Sensors. Annual Review of Analytical Chemistry 2(1), 241–264 (2009)
9. Frezza, B.M., Cockroft, S.L., Ghadiri, M.R.: Modular multi-level circuits from immobilized DNA-based logic gates. J. Am. Chem. Soc. 129(48), 14875–14879 (2007)
10. Lim, H.-W., Lee, S.H., Yang, K.-A., Lee, J.Y., Yoo, S.-I., Park, T.H., Zhang, B.-T.: In vitro molecular pattern classification via dna-based weighted-sum operation. Biosystems 100(1), 1–7 (2010)
11. Lipton, R.J.: DNA solution of hard computational problems. Science 268(5210), 542–545 (1995)
12. Minsky, M.: Steps toward artificial intelligence. Proceedings of the IRE 49(1), 8–30 (1961)
13. Sainz de Murieta, I., Rodríguez-Patón, A.: DNA biosensors that reason. Biosystems (in press, 2012)
14. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, 1st edn. Morgan Kaufmann (September 1988)
15. Ran, T., Kaplan, S., Shapiro, E.: Molecular implementation of simple logic programs. Nature Nanotechnology 4(10), 642–648 (2009)
16. Rodríguez-Patón, A., Larrea, J.M., Sainz de Murieta, I.: Inference with DNA Molecules. In: Calude, C.S., Hagiya, M., Morita, K., Rozenberg, G., Timmis, J. (eds.) Unconventional Computation. LNCS, vol. 6079, p. 192. Springer, Heidelberg (2010)
17. Rodríguez-Patón, A., Sainz de Murieta, I., Sosík, P.: Autonomous Resolution Based on DNA Strand Displacement. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 190–203. Springer, Heidelberg (2011)
18. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall series in artificial intelligence. Prentice Hall (December 2002)
19. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science 314(5805), 1585–1588 (2006)
20. Shortliffe, E.H., Buchanan, B.G.: A model of inexact reasoning in medicine. Mathematical Biosciences 23(3-4), 351–379 (1975)
21. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proceedings of the National Academy of Sciences 107(12), 5393–5398 (2010)

22. Takahashi, K., Yaegashi, S., Kameda, A., Hagiya, M.: Chain Reaction Systems Based on Loop Dissociation of DNA. In: Carbone, A., Pierce, N.A. (eds.) DNA 11. LNCS, vol. 3892, pp. 347–358. Springer, Heidelberg (2006)
23. Yurke, B., Turberfield, A.J., Mills, A.P., Simmel, F.C., Neumann, J.L.: A DNA-fuelled molecular machine made of DNA. Nature 406(6796), 605–608 (2000)
24. Zhang, B.-T., Jang, H.-Y.: A Bayesian Algorithm for In Vitro Molecular Evolution of Pattern Classifiers. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 458–467. Springer, Heidelberg (2005)
25. Zhang, D.Y., Winfree, E.: Dynamic allosteric control of noncovalent dna catalysis reactions. Journal of the American Chemical Society 130(42), 13921–13926 (2008)

DNA Self-Assembly and Computation Studied with a Coarse-Grained Dynamic Bonded Model

Carsten Svaneborg[1], Harold Fellermann[1,2], and Steen Rasmussen[1,3]

[1] Center for Fundamental Living Technology, Department of Physics, Chemistry and Pharmacy, University of Southern Denmark, Campusvej 55, DK-5320 Odense, Denmark
science@zqex.dk

[2] Complex Systems Lab., Barcelona Biomedical Research Park, Universitat Pompeu Fabra, Dr. Aiguadé 88, 08003 Barcelona, Spain
harold@sdu.dk

[3] Santa Fe Institute, 1399 Hyde Park Road, Santa Fe NM 87501, USA
steen@sdu.dk

Abstract. We study DNA self-assembly and DNA computation using a coarse-grained DNA model within the directional dynamic bonding framework [C. Svaneborg, Comp. Phys. Comm. 183, 1793 (2012)]. In our model, a single nucleotide or domain is represented by a single interaction site. Complementary sites can reversibly hybridize and dehybridize during a simulation. This bond dynamics induces a dynamics of the angular and dihedral bonds, that model the collective effects of chemical structure on the hybridization dynamics. We use the DNA model to perform simulations of the self-assembly kinetics of DNA tetrahedra, an icosahedron, as well as strand displacement operations used in DNA computation.

1 Introduction

Sequence specific hybridization of DNA single strands makes DNA molecules a flexible programmable building block. By choosing the right sequences, DNA self-assembly behavior can be programmed to produce well defined nano-structures. In the pioneering work of Seeman et al., branched DNA constructs have been utilized to self-assemble into a variety of structures [32,8,36,35]. With DNA origamis Rothemund invented a way to fold long DNA single strands into well defined planar structures by adding a large number of short stabilizing oligomer strands[26]. Later it was demonstrated how to let the planar origamis self-assemble into 3D nano-structures such as a box [2]. Ever since the pioneering work of Adlemann in 1994 [1], DNA has also been recognized as a massively parallel, versatile, and inexpensive computing substrate. In order for such substrate to be of practical interest, however, it is desirable that the computational framework is scalable and that individual computational elements can be combined to form circuits. Recently, a scalable approach to enzyme-free DNA computing has been proposed where circuits consist of relatively short DNA strands that communicate via strand displacement [31,25].

The Poland-Scheraga (PS) model has been very successful in predicting thermal melting and renaturing of long DNA strands [24,13]. It describes a DNA double strand as a 1D lattice where each base-pair is either hybridized or open. To each state is associated a free energy that has a sequence specific contribution from nearest neighbor interactions [29] as well as a polymer contribution from the conformational entropy of internal bubbles and frayed ends. Generalizations of the PS model exists, where the single strands are represented as semi-flexible polymers on a 3D lattice [12,14]. This provides a conceptual simplification since the polymer free energy contributions are given implicitly. The Dauxois-Peyrard-Bishop[22] (DPB) model represents a DNA double strand as a 1D lattice, but each base-pair is described by a continuous base-pair extension. The DPB model is defined by a Hamiltonian which includes a hybridization potential and a harmonic term penalizing deviations between nearest neighbor extensions.

The chemical structure of short DNA oligomers can be studied with atomistic molecular dynamics simulations such as Amber[6,7] and Charmm[4,17]. However, if we are interested in mesoscopic properties of long DNA molecules, it is more effective to utilize coarse-grained simulation models. Coarse-graining is the statistical mechanical process by which microscopic details are systematically removed, producing an effective mesoscopic model [16,21]. The major computational advantage of coarse-graining is that it allows us to focus our computational resources on studying the structures and dynamics at the mesoscopic level.

Coarse-grained models describe a nucleotide by a small number of effective interaction sites. In the “three site per nucleotide” model of de Pablo and co-workers, three sites represent the phosphate backbone site, the sugar group, and the base, respectively[28,27]. There is also a number of “two site per nucleotide” models, e.g. the model of Ouldridge and co-workers [19,20], where one site represents the base and another site the backbone and the sugar ring. Savelyev and Papoian [30] have formulated a “one site per nucleotide” model. As the number of interaction sites per nucleotide is reduced, the chemical structure is progressively lost. In simulations of DNA tagged nanoparticles, even more coarse-grained models are used. DNA molecules have been modeled e.g. as semi-flexible polymers with attractive sites on each monomer [11], or as a single sticky site that can be hybridized with free complementary free sticky sites [18]. While the chemical structure of DNA has been completely eliminated, these models still retain the DNA sequence specific hybridization effects on nanoparticle self-assembly.

We are interested in studying the statistical mechanics of hybridizing DNA strands and in particular the kinetics of DNA self-assembly and DNA computation using a DNA model that is as coarse-grained as possible. We have implemented a general framework allowing directional bonds to be reversibly formed and broken during molecular dynamics simulations[33]. Along with the bonds, the angular and dihedral interactions required to model the residual effects of chemical structure are also dynamically introduced and removed as dictated by the bond dynamics. This framework allows us to simulate reversible hybridization of complementary beads and chains built from such beads. In the present paper, we study a minimal dynamic bonding DNA model. For simplicity, we assume that the binding energy, as well as the bond, angular, and dihedral potentials are independent of sequence, and we have chosen a force field that produces a flat ladder-like structure in the double stranded state. Our motivation for these choices are to minimize the number of parameters required to specify the DNA model.

Dynamic bonding DNA models combine ideas from most of the existing DNA models. We regard them as dynamic generalizations of statistical mechanical theories and simultaneously as simplifications of coarse-grained DNA models. As in the PS model, a complementary base-pair can either be hybridized or open. When a base-pair is hybridized, it is characterized by a continuous hybridization potential as in the DPB model. Dynamic bonding DNA models can also be regarded as off-lattice generalizations of the lattice PS model [12]. Rather than trying to model chemical structure with interaction sites as in the "two and three site per nucleotide" models [28,27,19,20] dynamic bonding DNA models use angular and dihedral interactions to model the residual effects of local chemical structure. Dynamic bonded DNA double stands can reversibly melt and reanneal, which is not possible with the "one site per nucleotide model" of Savelyev and Papoian [30]. Finally, as in the sticky DNA models[18], a single bead in a dynamic bonding DNA model can equally well represent a domain.

Sect. 2 presents the dynamic bonding DNA model, which is used in Sect. 3 to study self-assembly of DNA constructs and DNA-computing constructs. Sect. 4 ends the article with a conclusions.

2 Dynamic Bonding DNA Model

In the present dynamic bonding DNA model, single stranded DNA (ssDNA) is represented by a string of nucleotide beads connected by stiff springs representing directional backbone bonds. Instead of using a four letter alphabet representing the ACGT nucleotides, in the present paper we increase the alphabet maximally to avoid getting trapped in transiently hybridized states. Physically, this corresponds to assuming that each bead represents a short sequence of nucleotides i.e. a domain, and that two non-complementary beads or domains are unable to hybridize. A novel feature of our DNA model is that it involves dynamic hybridization bonds, which are introduced or removed between complementary interaction sites or beads when they enter or exit the hybridization reaction radius. Along with the bonds, we dynamically introduce or remove angular and dihedral interactions in the chemical neighborhood of a hybridizing bead pair. These interactions are introduced based on the local bond and bead type pattern, and hence allows us to retain some effects of the local chemical structure in coarse-grained models. We utilize bonds carrying directionality to represent the 3'-5' backbone structure of DNA molecules. This allows us to introduce dihedral interactions that can distinguish between parallel and anti-parallel strand alignments. We have implemented this framework in a modified version of the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [23,33].

The DNA model relies on two ingredients, a Langevin dynamic for propagating a system in time and space, and a dynamic directional bonding scheme [33] that propagates the chemical structure of the system. The force on bead i is given by a Langevin equation

$$\mathbf{F}_i = -\boldsymbol{\nabla}_{\mathbf{R}_i} U - \frac{m}{\Gamma}\dot{\mathbf{R}}_i + \xi_i \quad \text{with} \quad U = U_{\text{bond}} + U_{\text{angle}} + U_{\text{dihedral}} + U_{\text{pair}}.$$

Here, the first term denotes a conservative force derived from the potential U. The second term is a velocity dependent friction, and the third a stochastic driving force characterized by $\langle \xi_i(t)\xi_j(t')\rangle = k_BTm/(\Gamma\Delta t)\delta_{ij}\delta(t-t')$. The potential U comprises four terms representing bond, angular, dihedral, and non-bonded pair interactions, respectively. The friction and stochastic driving force implicitly represents the effect of a solvent with a specified friction and temperature. The Langevin dynamics is integrated using a Velocity Verlet algorithm with a time step $\Delta t = 0.001\tau_L$ and $\Gamma = 2\tau_L$ using a customized version of LAMMPS [23,33].

Here and in the rest of the paper we use reduced units defined by the Langevin dynamics and DNA model. The unit of energy is $\epsilon = k_BT$, where we set Boltzmann's constant k_B to unity. The bead-to-bead distance along a single strand defines the unit of length σ which correponds to the rise distance of DNA. The mass is $m = 1$ for all beads. A Langevin unit of time is defined as $\tau_L = \sigma\sqrt{m/\epsilon}$. The diffusion coefficient of DNA model strand is $D(n) = k_BT\Gamma/(mn)$ where n is the total number of beads. This an be can be equated with the DNA diffusion coefficient of a particular experimental conditions to obtain a time mapping. Extrapolating the data in Ref. [34] yields $\tau_L \approx 1.6 \times 10^{-12}s$ for $n = 20$.

Fig. 1a shows complementary nucleotide beads with the same hue but different levels of color saturation. As a simplification, we allow each bead only to hybridize with a single complementary bead. The DNA model has two types of bond interactions: permanent backbone bonds (shown green/blue) and dynamic

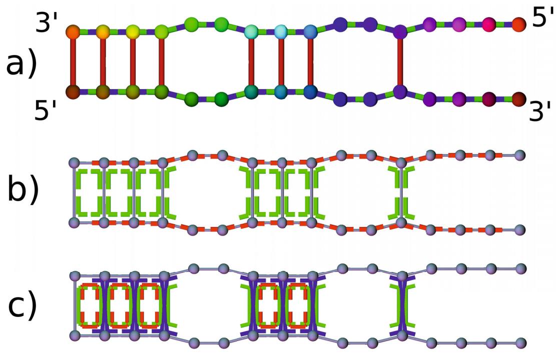


Fig. 1. Illustrative DNA conformation. a) complementary beads, backbone and hybridization bonds, b) angular interactions indicated by two lines parallel to the involved bonds, c) dihedral interactions indicated by three lines parallel to the involved bonds. The figure is explained in the text.

hybridization bonds (shown red). Backbone bonds and hybridization bonds are characterized by the two potentials:

$$U_{\text{bond,bb}}(r) = \frac{U_{\text{min,bb}}}{(r_{\text{c}}^{\text{b}} - r_{0}^{\text{b}})^2} \left((r - r_{0}^{\text{b}})^2 - (r_{\text{c}}^{\text{b}} - r_{0}^{\text{b}})^2 \right),$$

and

$$U_{\text{bond,hyb}}(r) = \begin{cases} \frac{U_{\text{min,hyb}}}{(r_{\text{c}}^{\text{h}} - r_{0}^{\text{h}})^2} \left((r - r_{0}^{\text{h}})^2 - (r_{\text{c}}^{\text{h}} - r_{0}^{\text{h}})^{-2} \right) & \text{for} \quad r < r_{c}^{\text{h}} \\ 0 & \text{for} \quad r \geq r_{\text{c}}^{\text{h}}. \end{cases}$$

In the simulations, we use $U_{\text{min,bb}} = 100\epsilon$, $r_{0}^{\text{b}} \equiv 1\sigma$, and $r_{\text{c}}^{\text{b}} = 1.2\sigma$, $r_{0}^{\text{h}} = 2\sigma$ and $r_{\text{c}}^{\text{h}} = 2.2\sigma$. Note that $U_{\text{bond,hyb}}(r) \leq 0$ for all distances. When two non-hybridized beads of complementary type are within a reaction distance r_{c}^{h} a hybridization bond is introduced between them. If they move further apart than r_{c}^{h} again, the hybridization bond is broken. The pair-interaction between beads is given by a soft repulsive potential, while we use the same potential for angular and dihedral interactions. They are given by

$$U_{\text{pair}}(r) = A \left[1 + \cos\left(\frac{\pi r}{r_{\text{c}}^{\text{p}}} \right) \right] \quad \text{for} \quad r < r_{\text{c}}^{\text{p}},$$

where we use $A = 1\epsilon$ and $r_{\text{c}}^{\text{p}} = 1\sigma$ in the simulations, and

$$U(\Theta; \Theta_0, U_{\text{min}}) = -\frac{U_{\text{min}}}{2} \left(\cos[\Theta - \Theta_0] + 1 \right),$$

Along the backbone of single strands we use a permanent angular interaction defined by $U(\Theta; \Theta_0 = \pi, U_{\text{min}} = 25\epsilon)$. This determines the persistence length of single strands. In Fig. 1b backbone angular interactions are shown as thick red lines around the central bead defining the angle.

In real DNA molecules, the hydrogen bonds between Watson-Crick complementary nucleotides act together with stacking interactions and the phosphordiester backbone bonds to give rise to a helical equilibrium structure of the double strand. In our coarse-grained model, we utilize angular and dihedral interactions to determine the ladder-like equilibrium structure of our DNA model. To control the stiffness of the double strands and to ensure anti-parallel 3'-5' alignment of the two single strands, we have assigned directionality to the backbone bonds [33]. This is also necessitated by the fact that the 3' and 5' carbons of the nucleotide sugar ring have been merged into the single nucleotide bead. Fig. 1a shows the backbone bonds colored green/blue to indicate the 3' and 5' ends, respectively.

When a hybridization bond is introduced, we also dynamically add angular interactions between the hybridization bond and the neighboring backbone bonds. These angular interactions are characterized by the potential $U(\Theta; \Theta_0 = \pi/2, U_{\text{min,a}})$, which favors a right angle conformation. When a hybridization bond is broken, concomitantly all the associated angular interactions are removed. In Fig. 1b the angular interactions are shown as green lines indicating the angle.

Besides introducing angular interactions, we also dynamically introduce dihedral interactions. A dihedral interaction involves four beads connected by three bonds, which defines a particular bond pattern, where the bonds can either be a hybridization bond, a $3' - 5'$ backbone bond, or a $5' - 3'$ backbone bond. Three bond patterns are possible. The bond pattern corresponding to red dihedrals in Fig. 1c, is characterized by $U(\Theta; \Theta_0 = 0, U_{\text{min,d}})$ which favors a planar (cis) conformation. The bond pattern corresponding to blue dihedrals is characterized by $U(\Theta; \Theta_0 = \pi, U_{\text{min,d}}, a = 0)$ which favors parallel backbone (trans) conformation. The last dihedral pattern corresponding to green dihedrals is characterized by $U(\Theta; \Theta_0 = 0, U_{\text{min,d}})$ which favors a parallel (cis) conformation. Note that without the directional backbone bonds, we would not be able to distinguish between these two latter dihedral patterns.

During a simulation, at each time we introduce a hybridization bond, we also introduce up to four angular interactions and up to eight dihedral interactions, less if the hybridization bond is at the end of a strand. Let Δ be the total decrease in binding energy when two beads hybridize inside a chain, and we assign one third of this energy to bond, angular, and dihedral interactions, respectively. This choice does not affect the static properties of the model, which are determined by the total energy associated with a conformation, however it does influence the dynamic properties. Hence $U_{\text{min,hyb}} = \Delta/3$, $U_{\text{min,a}} = \Delta/12$, and $U_{\text{min,d}} = \Delta/18$. We define $\Delta = 10\epsilon$ as a reference energy. Since only the ratio Δ/T enters the partition function of the model, this effectively fixes the absolute melting temperature of the stands. With the present model, the time spend per particle per step is approximately $1 \times 10^{-5}s$ on a standard PC.

3 Results

Three dimensional DNA structures can be built by utilizing the self-assembly properties of complementary strands and by linking several stands into a e.g. end-linked constructs. In particular, we have designed four constructs each comprising three end-linked 16 bead long strands. By programming the complementarity of the strands, we have designed the constructs to self-assemble into a tetrahedron[10,9]. We have also programmed the complementarity of 12 DNA constructs each comprising 5 end-linked 8 bead long strands. These constructs have been designed to self-assemble into an icosahedron[3]. We estimate that the melting temperatures are $T_m(8) \approx 1.3\epsilon$, and $T_m(16) \approx 1.6\epsilon$ from a separate set of melting simulations (not shown).

Fig. 2 shows visualizations of the DNA constructs during the self-assembly process. Initially the constructs are randomly placed into the simulation box. Progressively, complementary strands hybridize with each other, and the constructs form fragments that ultimately yield the designed target structures. The time scale of the self-assembly dynamics is determined by the time it takes the constructs to diffuse, collide, and hybridize completely. Since we have the simulation trajectory, we can also characterize the detailed time dependence of the self-assembly dynamics. Fig. 3 shows the fraction of hybridized bonds as a function of time. By analyzing the bond structure, we can furthermore study the

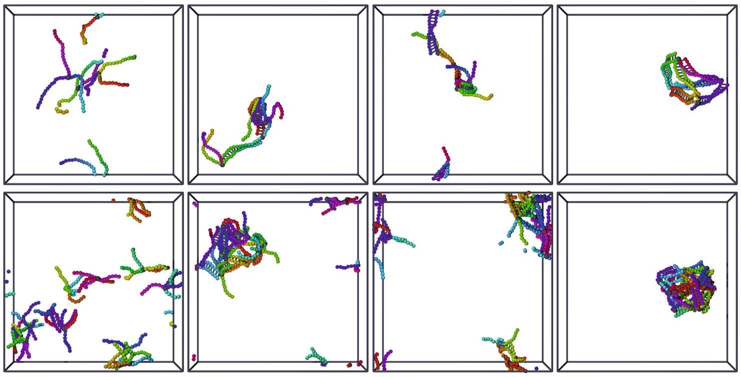


Fig. 2. Self-assembly of a tetrahedron from four 3-functional DNA constructs (top row) and an icosahedron from twelve 5-functional DNA constructs (bottom row). Snapshots correspond to times $t = 1000\tau_L$, $10.000\tau_L$, $20.000\tau_L$, $50.000\tau_L$ steps (top row), and for $t = 1000\tau_L$, $20.000\tau_L$, $30.000\tau_L$, $60.000\tau_L$ steps (bottom row). Simulations have been performed at $T = 1.0\epsilon$. Note that periodic boundary conditions apply to the simulation box.

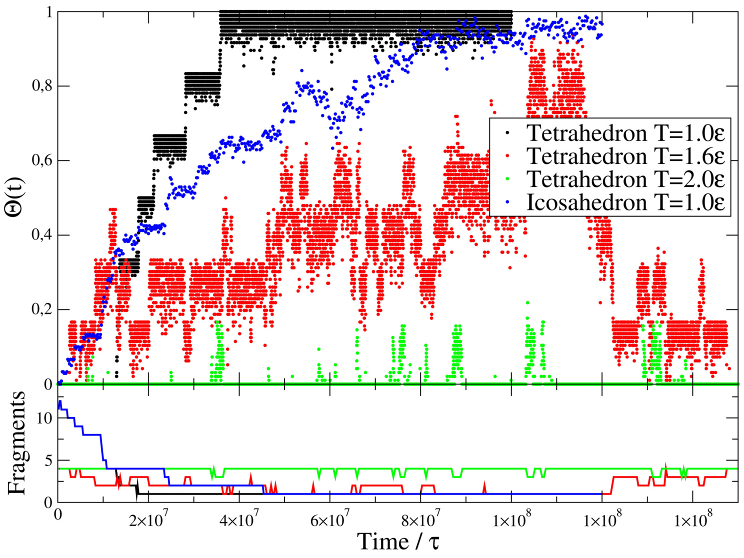


Fig. 3. Fraction of hybridized bonds (top) and number of fragments (bottom) vs. time and temperature during the self-assembly of a tetrahedron and an icosahedron.

evolution of the number of structural fragments. For the icosahedron, we see a slow increase in the hybridized bond fraction towards unity as the structure is progressively assembled, while the number of fragments drops simultaneously from initially twelve free constructs to one when all constructs form a single icosahedron. However, even through we only have a single fragment, it takes further time for the remaning hybridization bonds to be formed. The equilibrium hybridization bond fraction does not appear to be reached at the end of the simulation at $1 \times 10^8\tau$. For the tetrahedron, we observe a similar increase in the fraction of hybridized bonds, however with six distinct steps corresponding to the hybridization of each edge of the tetrahedron.

The self-assembly dynamics is stochastic and depends on initial conditions and random diffusive motion. We have run some of the simulations twice to see how they approach equilibrium along different trajectories. The equilibrium hybridization bond fraction appears to have been reached by the tetrahedron self-assembly simulations. For tetrahedra, we observe that self-assembly at higher temperatures leads to a marked decrease in the average hybridization bond fraction similar to melting of DNA double strands. At $T = 1.8\epsilon$ the temperature is above the melting temperature of the DNA constructs, and they only transiently hybridize. Since we have a single fragment at equilibrium, the bond reduction is most likely due to DNA bubbles. From the data sets we can estimate that the melting temperature of the tetrahedron i.e. $\Theta(T_\mathrm{m}) = 0.5$ is approximately $T_\mathrm{m}(\mathrm{Tetrahedron}) \approx 1.5\epsilon$.

In the strand displacement approach to DNA computation, individual gates consist of one DNA template that is composed of several logical domains. In their initial state, all domains but one are hybridized to one or more complementary strands and are therefore inert. The only exposed single strand domain of each gate is a short toehold region at one end of the template. This toehold region can reversibly bind a complementary signal strand which is designed to be longer than the toehold domain and complementary to the next domain(s) of the template. The newly binding signal is then able to hybridize to all matching domains of the template, thereby displacing strands that where previously bound [37]. The displaced strands can be fluorescently marked output signals, or internal signals that can bind to toehold regions of downstream gates. By choosing domains of appropriate length, it can be guaranteed that toehold binding is reversible, whereas the final strand displacement is effectively irreversible, thus computation is energetically downhill and kinetically irreversible, if and only if the correct input strands are present and match the logical setup of the gates. It has been shown that this approach leads to modular logic gates that enable the design of large scale DNA circuits [5,15].

Fig. 4 shows simulations of the strand displacement process underlying Seelig et al.'s DNA computing approach [31]. The top row shows the successful displacement of an initially hybridized 12 bead long signal strand from a 20 bead long template by a 20 bead long signal strand: once the signal strand diffuses to and binds to the toehold region, branch migration occurs quickly (during 300 time units) and the formerly bound signal strand is displaced irreversibly. The bottom row, on the other hand, shows how the displacement stalls in the presence of mismatches: here, a mismatch in the domain (last 10 beads) permits further hybridization of the signal strand. The newly binding and the original signal strand compete for matching bases in a random walk process until the non-matching strand dehybridizes again and leaves the gate available for potential matching signals (not present in the simulation).

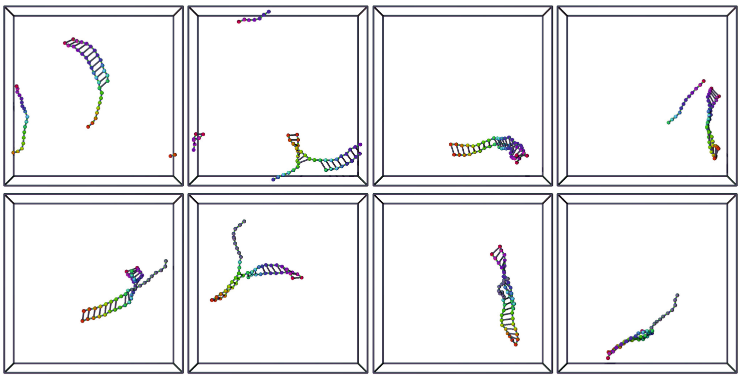


Fig. 4. Simulations of strand displacement. A 20 bead long oligomer displaces a 12 bead long oligomer initially hybridized to a template for times t =500, 1.600, 1.700, 1.900τ_L (top row). A 10+10 bead long oligomer where the latter half is non-complementary fails to displace a short oligomer hybridized to a template for times t = 100, 3.000, 7.000, 10.000τ_L. Simulations are run at temperature $T = 1\epsilon$. Non-complementary beads are show as gray.

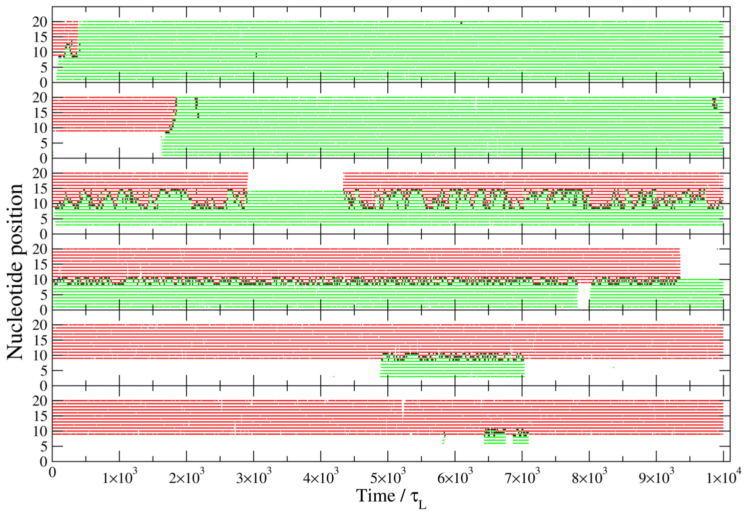


Fig. 5. Time evolution of branch migration on individual template nucleotide beads. From top to bottom a) A 12 bead long oligomer (red) being displaced by a 20 bead long oligomer (green) on the template. b) a 20 bead long oligomer with different random conditions. c) 12 bead oligomer (red) competing with a $12 + 8$ oligomer with 12 complementary and 8 non-complementary beads (green), d) a 12 bead oligomer (red) competing with a $10+10$ oligomer, e) a 12 bead oligomer (red) competing with a $8+10$ oligomer, f) a 12 bead oligomer (red) competing with $5 + 10$ oligomer. Also shown is the branch migration point (black).

Fig. 5 shows statistics of the displacement processes for several runs: the graphs depict hybridized bases of the original (red) and the newly binding signal strand (green), as well as the branch migration point (black). In the case of matching signals (top two simulations), it can be seen that displacement occurs quickly and essentially irreversible once the original strand is fully displaced. In the third simulation the signal strand and hybridized strand has the same length, and the interface is seen to diffuse forwards and backwards. A single dehybridization event is also observed for the original strand. In the case of mismatching signals (bottom three simulations), the displacement cannot proceed further than nucleotide 10, and the interface randomly moves between positions 8 and 10, until – occasionally – the mismatching signal dehybridizes from the toehold region (lack of green markers). In this case, the number of beads complementary to the toehold region (here 10, 8, and 5 beads) determines the equilibrium between hybridized and dehybridized configurations, and thus the performance and availability of the gate. Fig. 5 also depicts a source of potential failure in logical gates based on strand displacement, as the output signal can spontaneously dehybridize even in the absence of a matching input signal (as observed in the fourth simulation).

4 Conclusions

With these initial simulations, we have demonstrated that our coarse-grained DNA model can succesfully simulate DNA assembly as well as DNA strand displacement dynamics which form the basis of state-of-the-art DNA computing approaches. We have successfully simulated self-assembly of DNA tetrahedra and icosahedra from four and twelve branched DNA constructs, respectively. Simulations show that the constructs self-assemble into the expected target structures.

We have further simulated successful displacement of an output strand when a matching input strand is present. In the presence of mismatches, we could demonstrate how the displacement process is prevented. Our simulations also capture potential failures of gates based on strand displacement, namely spontaneous release of the output strand in the absence of an input signal. These proof of concept simulations demonstrate how our coarse-grained model can be used to optimize the length and arrangement of toehold and domain structures in DNA computing approaches.

While such gate optimizations do not necessarily require spatially resolved models, our coarse-grained DNA model enables us to study systems that integrate DNA assembly and computing within a single framework. This enables us to use these simulations as a starting point for building and testing statistical mechanical theories describing these complex systems.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no 249032 (MATCHIT). Funding for this work is provided in part by the Danish National Research Foundation through the Center for Fundamental Living Technology (FLinT).

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problem. Science 266, 1021 (1994)
2. Andersen, E.S., Dong, M., Nielsen, M.M., Jahn, K., Subramani, R., Mamdouh, W., Golas, M.M., Sander, B., Stark, H., Oliveira, C.L.P., Pedersen, J.S., Birkedal, V., Besenbacher, F., Gothelf, K.V., Kjems, J.: Self-assembly of a nanoscale DNA box with a controllable lid. Nature 459, 73 (2008)
3. Bhatia, D., Mehtab, S., Krishnan, R., Indi, S.S., Basu, A., Krishnan, Y.: Icosahedral DNA nanocapsules by modular assembly. Angew. Chem. 121, 4198 (2009)
4. Brooks, B.R., Brooks III, C.L., Mackerell Jr., A.D., Nilsson, L., Petrella, R.J., Roux, B., Won, Y., Archontis, G., Bartels, C., Boresch, S., Caflisch, A., Caves, L., Cui, Q., Dinner, A.R., Feig, M., Fischer, S., Gao, J., Hodoscek, M., Im, W., Kuczera, K., Lazaridis, T., Ma, J., Ovchinnikov, V., Paci, E., Pastor, R.W., Post, C.B., Pu, J.Z., Schaefer, M., Tidor, B., Venable, R.M., Woodcock, H.L., Wu, X., Yang, W., York, D.M., Karplus, M.: CHARMM: The Biomolecular Simulation Program. J. Comput. Phys. 30, 1545 (2009)
5. Cardelli, L.: Strand algebras for DNA computing. Nat. Comput. 10, 407 (2011)
6. Case, D.A., Darden, T.A., Cheatham III, T.E., Simmerling, C.L., Wang, J., Duke, R.E., Luo, R., Walker, R.C., Zhang, W., Merz, K.M., et al.: AMBER 11. University of California, San Francisco (2010)
7. Cheatham III, T.E., Young, M.A.: Molecular dynamics simulation of nucleic acids: Successes, limitations, and promise. Biopolymers 56, 232 (2000)
8. Chen, J., Seeman, N.C.: Synthesis from DNA of a molecule with the connectivity of a cube. Nature 350, 631 (1991)
9. Erben, C.M., Goodman, R.P., Turberfield, A.J.: A self-assembled DNA bipyramid. J. Am. Chem. Soc. 129, 6992 (2007)
10. Goodman, R.P., Schaap, I.A.T., Tardin, C.F., Erben, C.M., Berry, R.M., Schmidt, C.F., Turberfield, A.J.: Rapid chiral assembly of rigid DNA building blocks for molecular nanofabrication. Science 310, 1661 (2005)
11. Hsu, C.W., Sciortino, F., Starr, F.W.: Theoretical description of a DNA-linked nanoparticle self-assembly. Phys. Rev. Lett. 105, 55502 (2010)
12. Jost, D., Everaers, R.: A unified description of poly- and oligonucleotide DNA melting: nearest-neighbor, Poland-Sheraga and lattice models. Phys. Rev. E 75, 041918 (2007)
13. Jost, D., Everaers, R.: A unified Poland-Scheraga model of oligo-and polynucleotide DNA melting: Salt effects and predictive power. Biophys. J. 96, 1056 (2009)
14. Jost, D., Everaers, R.: Prediction of RNA multi-loop and pseudoknot conformations from a lattice-based, coarse-grain tertiary structure model. J. Chem. Phys. 132, 095101 (2010)
15. Lakin, M.R., Youssef, S., Cardelli, L., Phillips, A.: Abstractions for DNA circuit design. J R Soc. Interface 9, 470 (2012)
16. Langowski, J.: Polymer chain models of DNA and chromatin. Eur. Phys. J. E Soft Matter 19, 241 (2006)
17. MacKerell Jr., A.D., Banavali, N., Foloppe, N.: Development and current status of the CHARMM force field for nucleic acids. Biopolymers 56, 257 (2000)
18. Martinez-Veracoechea, F.J., Mladek, B.M., Tkachenko, A.V., Frenkel, D.: Design rule for colloidcal crystals of DNA-functionalized particles. Phys. Rev. Lett. 107, 045902 (2011)

19. Ouldridge, T.E., Louis, A.A., Doye, J.P.K.: DNA nanotweezers studied with a coarse-grained model of DNA. Phys. Rev. Lett. 104, 178101 (2010)
20. Ouldridge, T.E., Louis, A.A., Doye, J.P.K.: Structural, mechanical, and thermodynamic properties of a coarse-grained DNA model. J. Chem. Phys. 134, 085101 (2011)
21. de Pablo, J.J.: Polymer simulations: From DNA to composites. Annu. Rev. Phys. Chem. 62 (2011)
22. Peyrard, M., Cuesta-Lopez, S., James, G.: Modelling DNA at the mesoscale: a challenge for nonlinear science? Nonlinearity 21, T91 (2008)
23. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. J. Comp. Phys. 117, 1 (1995), http://lammps.sandia.gov
24. Poland, D., Scheraga, H.A.: Phase transitions in one dimension and the helix-coil transition in polyamino acids. J. Chem. Phys. 45, 1456 (1966)
25. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science 332, 1196 (2011)
26. Rothemund, P.W.K.: Folding DNA to create nanoscale shapes and patterns. Nature 440, 297 (2006)
27. Sambriski, E.J., Ortiz, V., de Pablo, J.J.: Sequence effects in the melting and renaturation of short DNA oligonucleotides: structure and mechanistic pathways. J. Phys. Condens. Matter 21, 034105 (2009)
28. Sambriski, E.J., Schwartz, D.C., de Pablo, J.J.: A mesoscale model of DNA and its renaturation. Biophys. J. 96, 1675 (2009)
29. SantaLucia, J.J., Hicks, D.: The thermodynamics of DNA structural motiefs. Annu. Rev. Biophys. Biomol. Struct. 33, 415 (2004)
30. Savelyev, A., Papoian, G.A.: Chemically accurate coarse graining of double-stranded DNA. PNAS 107, 20340 (2010)
31. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science 314, 1585 (2006)
32. Seeman, N.C.: Nucleic acid junctions and lattices. J. Theor. Biol. 99, 237 (1982)
33. Svaneborg, C.: LAMMPS framework for dynamic bonding an application modeling DNA. Comp. Phys. Comm. 183, 1793 (2012)
34. Tinlan, B., Pluen, A., Sturm, J., Weill, G.: Persistence length of single-stranded DNA. Macromolecules 30, 5763 (1997)
35. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature 394, 529 (1998)
36. Xhang, Y., Seeman, N.C.: Construction of a DNA-truncated octahedron. J. Am. Chem. Soc. 116, 1661 (1994)
37. Zhang, D.Y., Winfree, E.: Control of DNA strand displacement kinetics using toehold exchange. J. Am. Chem. Soc. 131, 17303 (2009)

Space and Energy Efficient Computation with DNA Strand Displacement Systems

Chris Thachuk and Anne Condon

Department of Computer Science, University of British Columbia, Vancouver, BC, Canada

Abstract. Chemical reaction networks (CRN's) are important models of molecular programming that can be realized by logically reversible, and thus energy efficient, DNA strand displacement systems (DSD's). Qian *et al.,* [12] showed that energy efficient DSD's are Turing-universal; however their simulation of a computation requires space (or volume) proportional to the number of steps of the computation. Here we show that polynomially space bounded computations can be simulated in both a space and energy efficient manner using logically reversible CRN's and DSD's. A consequence of our proofs is that determining whether a particular molecular species can be produced from an initial pool of molecules of a CRN or DSD is PSPACE-hard, and thus also verifying the correctness of CRN's and DSD's is PSPACE-hard.

1 Introduction

The area of molecular programming enjoys active research from both theoreticians and experimentalists due in part to its promise of embedded logical computation that can naturally interface with biological systems. For instance, *if* a condition is detected in a cell, *then* a certain therapeutic agent can be released. Molecular programs can be written in the language of chemical reaction networks (CRN) which detail how sets of reactant molecules can be transformed into new sets of product molecules. A most widely studied and experimentally practical model of computation in molecular programming entails so-called DNA strand displacement systems (DSD). DSDs leverage the fact that substrings of DNA strands will hybridize to their perfect complements and can also displace other bound strands sharing the same substring (see Fig. 1), and can in general realize any CRN [12,10]. By a careful, non-trivial design of strands one can realize a complex, yet deterministic computation. DSDs have been experimentally implemented and verified to simulate logic gates [14], neural networks [13], and DNA walkers [15], among numerous other applications.

Aside from the potential biological and chemical applications, DSDs and CRNs are also of independent interest due to their promise for realizing energy efficient computation. Rolf Landauer proved that logically irreversible computation—computation as modeled by a standard Turing machine—dissipates energy proportional to the number of bits of information lost, such as previous state information, and therefore cannot be energy efficient [8]. Surprisingly, Charles Bennett showed that in principle energy efficient computation is possible, by proposing a logically reversible universal Turing machine and by identifying nucleic acids (RNA/DNA) as a potential medium for reversible computation [1]. A logically reversible computation is a chain from an initial state to a final state where each intermediate state has exactly one predecessor and one successor. Bennett's seminal work was space inefficient as his reversible Turing machine simulation required $O(T(n))$ space to simulate a non-reversible machine that required $T(n)$ steps to complete, regardless of its space usage. He later proved that PSPACE equals reversible PSPACE [2]—the class of problems solvable in deterministic polynomial space can be solved by a reversible Turing machine in polynomial space. This result has since been generalized to prove DSPACE equals reversible DSPACE [9]. Until recently, it remained unclear if a physical system could realize logically reversible computation. Qian *et al.,* [12] gave a DSD implementation of a stack machine capable of energy efficient Turing universal computation. Similar to Bennett's seminal work, their implementation requires space proportional to the number of steps in the computation as it consumes *fuel* molecules to drive the overall process forward. Condon *et al.,* [4] demonstrated that, in principle, logically reversible and space efficient computations can be realized in CRNs and DSDs by giving an n-bit Gray code counter that progresses through 2^n states using only $O(poly(n))$ space.

In this work, we ask the question: can space *and* energy efficient computation be realized? We answer in the affirmative by showing how any problem in PSPACE can be solved by a logically reversible CRN using polynomial space. Our CRN can be realized by an energy efficient DSD implementation. Not only do our results further characterize the computational power of CRNs and DSDs, they shed light on the complexity of a number of important related problems such as CRN and DSD model checking and verification [6,7]. We show that even determining if an arbitrary state is reachable from an initial state of a CRN or DSD—a question that must be solved when verifying the correctness of a CRN or DSD—is PSPACE-hard. We show that the problem is PSPACE-complete for restricted classes of CRNs and DSDs. Our results also gives strong evidence that predicting low energy barrier folding pathways for multiple interacting nucleic acid strands is PSPACE-complete.

In section 2, we give definitions and the necessary background information for our results. In section 3 we develop our main result by showing how a PSPACE-complete problem can be solved by a logically reversible CRN. In section 4 we show how our CRN can be realized as an energy efficient DSD. In section 5 we demonstrate a number of consequence of our result and resolve the complexity of a number of related problems. We summarize our results in section 6.

2 Preliminaries

A *chemical reaction equation* details a process whereby certain molecule types can be consumed—the *reactants*—and others produced—the *products*—within some reaction volume. For simplicity, we assume that the reaction volume is a closed system. A reaction may also require the presence of *catalyst* molecules of certain types. We refer to all three categories, generically, as *signals*. For example, the reaction $A \overset{C}{\rightarrow} B$ consumes a signal of type A and produces a signal of type B in the presence of the catalyst signal C. This reaction is *irreversible*; however, the reaction $A \overset{C}{\rightleftharpoons} B$ is *reversible* meaning that a signal of type A can also be produced by consuming a signal of type B in the presence of the catalyst signal C. A *chemical reaction network* (CRN) is a set of chemical reactions, in addition to a multiset of signals present within the reaction volume, prior to any reaction occurring, called the *initial* set. A CRN is *proper* if every reaction consumes the same number of signals that it produces. The *state* of a CRN is the current composition of signals within the reaction volume. The CRN can move from some current state, S, to a new state, S', by applying any reaction requiring reactants and catalysts present in S. In general, many reactions may be applicable for the current state. We define a computation of a CRN $\mathcal{C}$, as a trace of the states from $\mathcal{C}$'s initial state S_{init} to some final target state S_{end}. We say that $\mathcal{C}$ is a *logically reversible CRN* if the computation forms a chain from S_{init} to S_{end} such that in any non-terminal state along the chain, exactly two reactions are possible: a reversal of the previous reaction, and one other reaction. In the CRNs considered in this work, we assume applicable reactions are selected with equal probability and with the same rate in the forward and reverse direction. Thus, a computation of length n for a logically reversible CRN will perform an unbiased random walk along the state chain and is expected to reach the end state within $O(n^2)$ steps [5].

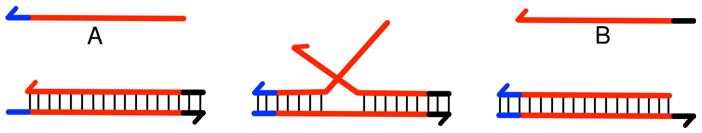


Fig. 1. A gate implementing the reaction $A \rightleftharpoons B$ via toehold-mediated branch migration

A *DNA strand displacement* (DSD) system is an implementation of a CRN, consisting of single stranded *signals* and double stranded *gates* that facilitate reactions. Strands in the system are composed of two types of domains: short *toehold* domains, and *long domains*. Toehold domains bind reversibly, and long domains irreversibly, to complementary regions on gates. The fundamental operation in a DSD is *toehold mediated strand displacement*, whereby the toehold of a signal strand can bind to an unbound complementary toehold domain of a gate and, if the adjacent long domain is complementary, it can displace a currently bound signal strand of the same length (see Fig. 1).

Consider an example DSD where the initial state consists of two copies of the A signal, but only one copy of the gate $\mathcal{T}$ is available in the reaction volume. It would be impossible to produce two copies of signal B. To properly account for gates at the CRN level of abstraction, we augment chemical reactions with unique *tags*. For example, the reaction of Fig. 1 can be described by the tagged reaction $A+\mathcal{T} \rightleftharpoons \mathcal{T}'+B$, denoting that a gate of type $\mathcal{T}$ is required to consume signal A, produce signal B, and results in the gate being reversed—that is, the same gate can only be used next to consume a B signal, produce an A signal, and reset itself to the forward orientation. We define the space complexity of a trace for a *tagged* CRN as the sum of two quantities: the maximum signal set size of any state in the trace, and the number of tags required to complete the computation. Intuitively, this corresponds to the required size of the reaction volume. In the remainder of the paper we only consider tagged CRNs; however, for simplicity we omit the actual tag signals in the reaction equations. We observe the following obvious, but useful property of proper CRNs.

Lemma 1. *A proper CRN with initial set S will always have $|S|$ free signals during a computation.*

CRNs can be implemented by DSDs in a number of ways. We will leverage one such implementation in our results, which relies on the assumption that certain signals only occur as a single copy within the reaction volume. The use of a single copy mutex species is used to ensure that a strand displacement cascade which implements any particular reaction will occur as a transaction and therefore appear *atomic*. Specifically, either the entire cascade implementing a reaction will succeed, or it will return to the state prior to beginning the cascade. Importantly, the mutex molecule is sequestered during the cascade and therefore another reaction cannot begin.

Theorem 1 (Qian *et al.,* [12], Condon *et al.,* [4]). *Any logically reversible tagged CRN requiring $O(s)$ space can be simulated by a DSD in $O(poly(s))$ space that ensures reactions appear atomic and occur in the same logical reaction sequence.*

Finally, we formally define three problems we reason about in this work.

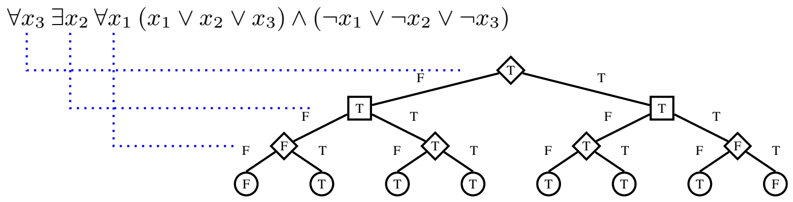
CRN REACHABILITY (CRNR)
Instance: A chemical reaction network with initial state S_{init} and an arbitrary state S'.
Question: Is S' reachable from S_{init}?

DSD REACHABILITY (DSDR)
Instance: A DNA strand displacement system with initial state S_{init} and an arbitrary state S'.
Question: Is S' reachable from S_{init}?

TOTALLY QUANTIFIED 3-SATISFIABILITY (Q3SAT)
Instance: A totally quantified boolean formula ψ of n variables in prenex normal form, $\forall x_n \exists x_{n-1} \forall x_{n-2} \ldots Q_1 x_1\ \phi$, where ϕ is an unquantified boolean formula of m clauses in conjunction normal form, each containing a literal for 3 distinct variables.
Question: Is the formula ψ satisfiable?

3 Space Efficient CRN Simulation of PSPACE

Our goal is to demonstrate that any problem in PSPACE can be solved by a space efficient, logically reversible, tagged CRN. To that end, we will show how a CRN with those properties can be constructed to solve any arbitrary instance of the Q3SAT problem. We present our solution in three logical parts. In section 3.1, we demonstrate how to construct a CRN for verifying if a 3SAT formula is satisfied. In section 3.2, we present



$\forall x_3\ \exists x_2\ \forall x_1\ (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$
T
F
T
F
T
F
T
F
T
F
T
F
T
F
T
T
T
F
T
F
T
T
T
T
T
T
F

Fig. 2. Solving a Q3SAT instance. Edge labeled paths from root to leaf denote variable assignments. Nodes are satisfied based on quantifier and satisfiability of left and right subtrees.

an elegant solution for traversing a perfect binary tree in post-order that is both space efficient and logically reversible. In section 3.3, we demonstrate how the two CRNs can be integrated and then modified to capture the semantics of strictly alternating variable quantifiers in the Q3SAT instance.

To understand the intuition behind our construction, consider that a perfect binary tree of height n, with each level of the tree representing a variable, has 2^n leaves, each with a unique path from the root specifying a unique variable assignment. A tree defined in this manner can be used to express the semantics of strictly alternating quantifiers in the Q3SAT instance (see Fig. 2). Leaf nodes are considered satisfied if and only if the current variable assignment satisfies the unquantified 3SAT formula of the Q3SAT instance. Internal nodes can be used to propagate satisfiability of a partially solved instance up the tree. For example, if an internal node represents a universally quantified variable, then it is marked as true if and only if both of its subtrees are satisfied. Similarly, a node representing an existentially quantified variable is marked false if and only if both subtrees are not satisfied. In this straightforward manner, the overall quantified formula can be determined to be satisfied or not, once the root is marked. Since the satisfiability of a node can immediately be determined once that of its two subtrees is known, we perform a post-order traversal of the tree. Furthermore, we exploit the fact that once the satisfiability of a subtree is marked, the satisfiability of its descendants is irrelevant and can be *forgotten*. This allows us to smartly reuse space in our tree traversal procedure.

3.1 Verifying a 3SAT Instance Variable Assignment

We first demonstrate how the formula ϕ can be verified as satisfied or unsatisfied for a particular variable assignment. A variable assignment ensures exactly one signal for each variable x_i is present: x_i^T for a true assignment, and x_i^F otherwise. We first introduce the necessary reactions to verify an individual clause and demonstrate how the overall formula can be determined true or false.

Verifying an Arbitrary Clause. Recall that in a 3SAT instance, each clause consists of exactly three literals, each for a distinct variable. As such, there are exactly eight possible truth assignments and we create a reversible reaction for each. The reactions for verifying the i^{th} clause, containing literals for variables x_j, x_k and x_l are given in Fig. 3 (left). When the clause signal molecule $C_i^?$ is present, exactly one of the eight reactions can be applied, specified by the current variable assignment. The variable signals act as catalysts and the $C_i^?$ signal is consumed producing either a C_i^T signal if the clause is satisfied, or C_i^F otherwise.

For example, suppose C_i represents the following clause: $(x_j \vee \neg x_k \vee x_l)$. The reaction having catalysts x_j^F, x_k^T, and x_l^F will produce C_i^F. The other seven reactions will produce C_i^T. Note that for a particular variable assignment, only one reaction will apply in both the forward and reverse direction, ensuring the process is logically reversible.

Verifying the Overall Formula. The overall process of verifying the formula ϕ can be thought of as a process that is initiated by consuming the signal $\phi^?$ and completes by producing either the signal ϕ^T, if ϕ is satisfied, or ϕ^F otherwise. The variable assignment signals are catalysts, and their values are maintained after the process completes.

$$
\begin{array}{llcll}
(1) & C_i^? & \overset{x_j^F + x_k^F + x_l^F}{\rightleftharpoons} & C_i^{[F/T]} & ,_{1 \le i \le m} \\
 & C_i^? & \overset{x_j^F + x_k^F + x_l^T}{\rightleftharpoons} & C_i^{[F/T]} & ,_{1 \le i \le m} \\
 & & \vdots & & \\
 & C_i^? & \overset{x_j^T + x_k^T + x_l^T}{\rightleftharpoons} & C_i^{[F/T]} & ,_{1 \le i \le m}
\end{array}
\qquad
\begin{array}{llcll}
(2) & \phi^? & \rightleftharpoons & C_1^? & \\
(3) & C_i^T & \rightleftharpoons & C_{i+1}^? & ,_{1 \le i < m} \\
(4) & H_i^? + C_i^F & \rightleftharpoons & H_i^{\checkmark} + \phi^F & ,_{1 \le i \le m} \\
(5) & C_m^T & \rightleftharpoons & \phi^T &
\end{array}
$$

Fig. 3. (left) Eight reaction equations to verify an arbitrary 3SAT clause C_i for each combination of variable assignments. The product of the reaction is C_i^T for assignments that satisfy the i^{th} clause, and C_i^F otherwise. (right) Reaction equations to verify the overall 3SAT formula ϕ, consisting of m clauses.

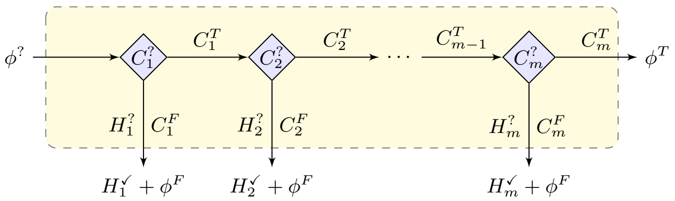


Fig. 4. Flow control when verifying a formula ϕ having m clauses.

For the formula to be true, all clauses must be satisfied. However, any combination of unsatisfied clauses will result in ϕ being false. For this reason, care must be taken that clauses are checked systematically to ensure reversibility. The overall process is depicted in Fig. 4 and the reactions are given in Fig. 3 (right). The process checks each clause, in sequence, and if the current clause is unsatisfied then the ϕ^F signal is immediately produced in addition to a history signal denoting the first clause to be unsatisfied. The sole purpose of the history signal is to ensure the reversibility of the computation, should the ϕ^F signal be produced. Otherwise, all clauses are satisfied, and thus the signal ϕ^T can be produced and is sufficient to ensure the computation is reversible.

Lemma 2. *A 3SAT boolean formula of m clauses over n variables can be verified by a logically reversible tagged CRN in $O(m)$ reaction steps using $\Theta(m+n)$ space.*

Proof. Importantly, we must now establish that the process is logically reversible. We argue formally by induction on m, the number of clauses of the 3SAT formula ϕ. In addition to the clause history domains, we assume initially that the signal $\phi^?$ is present and exactly one signal for each variable x_i denoting its truth assignment—x_i^T or x_i^F. Consider the base case when $m = 1$. Given the initial set of signals, only reaction (2) can be applied which produces $C_1^?$. At this point, exactly two reactions can occur: a reversal of the previous reaction, or the clause checking reaction that consumes $C_1^?$ and corresponds to the current variable assignment of the three variables in clause 1. If clause 1 is satisfied (not satisfied), C_1^T (C_1^F) is produced. In both cases, other than reversing the previous reaction, only one reaction is possible. If the clause is satisfied,

ϕ^T is next produced ending the process. If the clause is unsatisfied, ϕ^F and $H_1^{\checkmark}$ are next produced, ending the process. Note that in both cases, only the reverse of the previous reaction could be applied next. Thus, the process is logically reversible. Suppose the same holds for $m-1$ clauses and consider the case when ϕ has m clauses. We consider two cases:

Case 1 (The first $m-1$ clauses of ϕ are satisfied). By the inductive hypothesis, signal $C_m^?$ will eventually be produced, in a logically reversible manner. As before, other than the reverse of the previous reaction, only one clause reaction will be applicable and will produce either C_m^T or C_m^F. If C_m^T is produced, the reverse of the previous reaction can be applied, or ϕ^T is next produced, ending the process. Similarly, if C_m^F was produced, either the reverse of the previous reaction can be applied, or $\phi^F + H_m^{\checkmark}$ is next produced, ending the process. Note that in either case, only one reaction can be applied next: the reverse of the last reaction. Thus, the process is logically reversible.

Case 2 (At least one of the first $m-1$ clauses of ϕ are unsatisfied). By the inductive hypothesis, this case will correctly produce ϕ^F and a history signal denoting the first unsatisfied clause. The new reactions pertaining to clause m are not applicable and thus inconsequential.

It is easy to see that in the worst case, $O(m)$ reactions steps are required. Finally, we establish the space claim. The initial set of signals has size $\Theta(m+n)$ as it consists of the n variable signals, m clause history domains and the signal $\phi^?$. The CRN is proper, therefore by Lemma 1 the number of signals will remain the same throughout the computation. The CRN has $\Theta(m)$ reactions since there are a constant number for each of the m clauses and the overall formula verification. Since each reaction is applied at most once when verifying a formula, one tag per reaction is sufficient, therefore establishing the $\Theta(m+n)$ space bound. □

3.2 A Space Efficient Post-Order Tree Traversal

Next we demonstrate how to perform a post-order traversal of a perfect binary tree in a space-efficient manner. Importantly, the procedure must be logically reversible. The intuition is captured in Fig. 5. For any node with a left and right child, once the descendants of the left child have been recursively traversed (Fig. 5 (a)), the left child can be marked (Fig. 5 (b)). Any information stored in those descendant nodes is no longer required and the whole traversal of that subtree can be reversed (Fig. 5 (c)), the traversal can move to the right child (Fig. 5 (d)), the right subtree can be recursively traversed (Fig. 5 (e)), and finally the right child marked (Fig. 5 (f)).

Lemma 3. *Given a perfect binary tree of height h, all descendants of the root can be traversed in post-order, by a logically reversible tagged CRN, in $\Theta(3^h)$ reaction steps, using $\Theta(h)$ space.*

Proof. As each reaction of the CRN is reversible, after every reaction step, the reverse of the previous reaction can always be applied. To demonstrate the CRN is logically reversible, we need to demonstrate that at any point there is at most one other reaction that can be applied. We will further establish the invariant that each reaction strictly

$$(6) \quad l_i^? \overset{x_i^F + r_i^? + r_j^\checkmark}{\rightleftharpoons} l_i^\checkmark, \; {\scriptstyle 1 \le i \le h \atop 1 \le j < i}$$

$$(7) \quad x_i^F \overset{l_i^\checkmark + r_i^? + l_j^?}{\rightleftharpoons} x_i^T, \; {\scriptstyle 1 \le i \le h \atop 1 \le j < i}$$

$$(8) \quad r_i^? \overset{r_{i+1}^? + l_i^\checkmark + x_i^T + r_j^\checkmark}{\rightleftharpoons} r_i^\checkmark, \; {\scriptstyle 1 \le i \le h \atop 1 \le j < i}$$

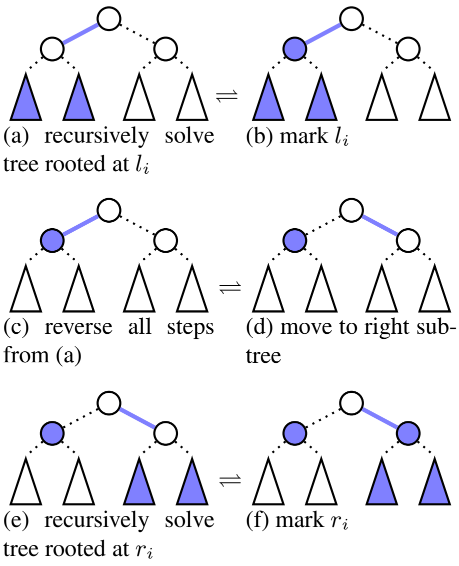

(a) recursively solve tree rooted at l_i
(b) mark l_i
(c) reverse all steps from (a)
(d) move to right subtree
(e) recursively solve tree rooted at r_i
(f) mark r_i

Fig. 5. A logically reversible post-order traversal of all descendants of the root of a height h perfect binary tree

alternates in being applied in the forward and reverse direction, ensuring at most one tag is required for each type of reaction. We will argue by structural induction. Let s_h denote the number of reaction steps required for a tree of height h.

Consider the base case when $h = 1$ with initial set $\{r_2^?, x_1^F, l_1^?, r_1^?\}$. Reaction (8) cannot be applied until the signal x_1^T is present which is produced by reaction (7). Similarly, reaction (7) cannot be applied until signal $l_1^\checkmark$ is present. Thus, it is easy to see that reaction (6) must first be applied—marking the left subtree—followed by reaction (7)—moving to the right subtree—and finally reaction (8)—marking the right subtree and completing the traversal in $s_1 = 3$ reaction steps. Each reaction was only applied once, in the forward direction, so the strictly alternating invariant is trivially maintained.

Suppose the traversal completes in s_{h-1} reactions steps, is logically reversible, and the strictly alternating invariant is maintained for a tree of height $h-1$. Consider a tree of height h, having initial set $S = \{r_{h+1}^?\} \cup \bigcup_{1 \le i \le h} \{x_i^F, l_i^?, r_i^?\}$. Before reaction (6) (and thus reaction (7) and (8)) can be applied, the signals $\bigcup_{1 \le j < h} \{r_j^\checkmark\}$ must be present. As the left subtree is selected, the signal $r_h^?$ is present, and by the induction hypothesis, the only available action is to produce these signals in s_{h-1} logically reversible reaction steps, that maintain the strictly alternating invariant, by traversing the subtree rooted at l_h (see Fig. 5 (a)). Importantly, the signals $\bigcup_{1 \le j < h-1} \{r_j^?\}$ are now absent and therefore no reaction affecting levels $1, \dots, h-2$ can occur. Other than reversing the previous reaction, which produced signal $r_{h-1}^\checkmark$, only reaction (6) can be applied for level h, thus producing $l_h^\checkmark$ (see Fig. 5 (b)). Next, observe that reaction (7) cannot be applied until the signals $\bigcup_{1 \le j < h} \{l_j^?\}$ are present. Other than reversing the previous reaction, only a reversal of all s_{h-1} reaction steps that traversed the left subtree can be applied next, yielding the required signals to next apply reaction (7), producing signal x_h^T, denoting a move to the right subtree (see Fig. 5 (c) and (d)).

Note that the reversal of the left subtree will maintain the strictly alternating invariant as it ensures all lower level reactions have been reset to their initial state, in order to be used again in the right subtree. Similar to reaction (6), reaction (8) cannot be applied at level h until the right subtree is traversed in s_{h-1} logically reversible reaction steps (see Fig. 5 (e)). Other than reversing the previous reaction, only reaction (8) can next be applied at level h producing the signal $r_h^{\checkmark}$ and ensuring no further reactions on lower levels can occur. The traversal is complete and no reaction, other than the reverse of the previous, can occur. Thus, the overall traversal is logically reversible, and is clearly in post-order. As the strictly alternating invariant was maintained for all lower level reactions, and all reactions at level h have been applied for the first time, and only once, the invariant is maintained for a tree of height h.

Exactly 3 reactions occurred at level h, and $3s_{h-1}$ reactions were required for the two traversals and one reversal of the height $h-1$ subtrees, giving us the recurrence $s_h = 3s_{h-1} + 3$. Solving s_h with $s_1 = 3$ gives us the closed form expression $\frac{3}{2}(3^h - 1)$, establishing the claimed $\Theta(3^h)$ reaction steps.

Finally, consider the space claim. As we have shown that reactions strictly alternate being applied in the forwards and reverse direction, at most one tag for each of the $\Theta(h)$ reactions is sufficient. Consider that the initial set for a tree of height h is $S = \{r_{h+1}^?\} \cup \bigcup_{1 \le i \le h} \{x_i^F, l_i^?, r_i^?\}$ and therefore $|S| = 3h + 1$. Since the CRN is proper, we immediately establish the space claim by Lemma 1. $\square$

3.3 Solving a Q3SAT Instance

We now have the means to verify if a variable assignment satisfies a 3SAT formula ϕ. We can also traverse a perfect binary tree in post-order, and in the process enumerate all possible variable assignments for ϕ. What remains is to combine these processes together in order to determine if a Q3SAT instance can be satisfied. We approach the integration in two parts. First, we will demonstrate how the formula verification process can be triggered immediately prior to the tree-traversal marking a leaf node and how the verification reactions can be entirely reversed, prior to the next time the verification procedure must run. This effectively demonstrates how any problem in NP can be solved by a logically reversible CRN in polynomial space, if we specify the end of computation as the presence of the signal ϕ^T, or the signal ϕ^F in conjunction with the signals for the final variable assignment to be enumerated. Finally, we demonstrate how the tree traversal reactions of Fig. 5 can be augmented in order to capture the semantics of alternating universal and existential quantifiers, thus demonstrating how any problem in PSPACE can be solved in polynomial space by a logically reversible CRN.

Integrating Formula Verification and Tree Traversal. Recall the sequence of logical steps in traversing level 1 of the tree, *i.e.,* the leaves: (1) *mark left*, (2) *move right*, (3) *mark right*. We augment the reactions for level 1 to force the following sequence: (1a) *verify ϕ*, (1b) *mark left*, (2a) *reverse (1a)*, (2b) *move right*, (3a) *verify ϕ*, (3b) *mark right*. This new sequence ensures two invariants: (i) the current variable assignment is verified prior to marking the current leaf, and (ii) the verification procedure is fully reversed prior to the next verification.

The augmented reactions are given in Fig. 6. Both reactions marking a leaf have been split into two variants, each ensuring the verification procedure has completed by requiring as a catalyst one of the two possible outcomes of the verification process. In addition, we add new signals to record whether or not the variable assignment for a particular leaf is a satisfying assignment for ϕ. These signals will be used later to propagate satisfiability up the tree, once quantifiers have been integrated. Note that the reaction to move to the right leaf now requires the signal $\phi^?$ as a catalyst. This forces all steps performed in the previous verification to reverse. After moving to the right leaf, and thus swapping the value of variable x_1, the verification process can again run immediately prior to marking the right leaf. Importantly, we want to ensure that the verification procedure is completely integrated into the leaf level reactions and cannot perform any reactions while the traversal is marking higher level nodes. This is easily accomplished by augmenting reactions (2)-(5) to require $r_2^?$ as a catalyst. Note that the augmented variants of the tree traversal reactions are also fully distinguishable by their catalysts (and products), thus ensuring the process is logically reversible.

	New Catalysts		
	$\phi^?$	$\phi^{\mathbf{F}}$	$\phi^{\mathbf{T}}$
mark left		$\mathbf{L_1^?}+l_1^? \overset{\cdots}{\rightleftharpoons} l_1^{\checkmark}+\mathbf{L_1^F}$	$\mathbf{L_1^?}+l_1^? \overset{\cdots}{\rightleftharpoons} l_1^{\checkmark}+\mathbf{L_1^T}$
move right	$x_1^F \overset{\cdots}{\rightleftharpoons} x_1^T$		
mark right		$\mathbf{R_1^?}+r_1^? \overset{\cdots}{\rightleftharpoons} r_1^{\checkmark}+\mathbf{R_1^F}$	$\mathbf{R_1^?}+r_1^? \overset{\cdots}{\rightleftharpoons} r_1^{\checkmark}+\mathbf{R_1^T}$

Fig. 6. Integrating the 3SAT verification procedure into the leaf level reactions of the tree traversal procedure. Two reaction variants are created for marking leaf nodes as either satisfied or unsatisfied based on the result of the verification procedure. The *move right* reaction requires $\phi^?$ as a catalyst, thus ensuring the verification procedure is reversed prior to the next verification step. Existing catalysts omitted for space.

Integrating Quantifiers into the Tree Traversal. Integrating quantifiers in non-leaf levels of the tree is relatively straightforward. Recall that the levels of the tree strictly alternate between universal and existential quantification. For each level, we create four variants of the correct quantifier for both the left and right node marking reactions to additionally produce a signal indicating if the current subtree is satisfied. The reaction variants for marking a left node are given in Fig. 7. These reactions require as catalysts the signals indicating if the left and right subtree of the current node is satisfied and therefore four variants are sufficient to consider all cases, for each type of quantifier. As with the leaf level reactions, the augmented reaction variants can be fully distinguished by their catalysts ensuring the computation remains logically reversible, and the correct reactions are reversed.

Ending the Computation. Once both subtrees of the root have been solved the output signal can be produced based on the satisfiability of the subtrees and on the quantifier imposed on the root level variable x_n. The reaction equations for the universal quantifier are shown in Fig. 8. Modifying the reactions for an existential quantifier is straightforward.

	New Catalysts			
	$\mathbf{L^F_{i-1}} + \mathbf{R^F_{i-1}}$	$\mathbf{L^F_{i-1}} + \mathbf{R^T_{i-1}}$	$\mathbf{L^T_{i-1}} + \mathbf{R^F_{i-1}}$	$\mathbf{L^T_{i-1}} + \mathbf{R^T_{i-1}}$
∀ levels	$\mathbf{L_i^?}+l_i^? \overset{\cdots}{\rightleftharpoons} l_i^{\checkmark}+\mathbf{L_i^F}$	$\mathbf{L_i^?}+l_i^? \overset{\cdots}{\rightleftharpoons} l_i^{\checkmark}+\mathbf{L_i^F}$	$\mathbf{L_i^?}+l_i^? \overset{\cdots}{\rightleftharpoons} l_i^{\checkmark}+\mathbf{L_i^F}$	$\mathbf{L_i^?}+l_i^? \overset{\cdots}{\rightleftharpoons} l_i^{\checkmark}+\mathbf{L_i^T}$
∃ levels	$\mathbf{L_i^?}+l_i^? \overset{\cdots}{\rightleftharpoons} l_i^{\checkmark}+\mathbf{L_i^F}$	$\mathbf{L_i^?}+l_i^? \overset{\cdots}{\rightleftharpoons} l_i^{\checkmark}+\mathbf{L_i^T}$	$\mathbf{L_i^?}+l_i^? \overset{\cdots}{\rightleftharpoons} l_i^{\checkmark}+\mathbf{L_i^T}$	$\mathbf{L_i^?}+l_i^? \overset{\cdots}{\rightleftharpoons} l_i^{\checkmark}+\mathbf{L_i^T}$

Fig. 7. Integrating quantifiers to non-leaf levels of the tree traversal. For both universal and existential levels, four variants of the left node reactions are created to process the four combinations of left and right subtree satisfiability. The integration is identical for right node reactions. Existing catalysts omitted for space.

$$\begin{aligned}
(9)\quad \psi^? &\overset{L^F_{n-1}+R^F_{n-1}+r^{\checkmark}_j}{\rightleftharpoons} \psi^{\mathbf{F}} &&, 1\le j<n\\
\psi^? &\overset{L^F_{n-1}+R^T_{n-1}+r^{\checkmark}_j}{\rightleftharpoons} \psi^{\mathbf{F}} &&, 1\le j<n\\
\psi^? &\overset{L^T_{n-1}+R^F_{n-1}+r^{\checkmark}_j}{\rightleftharpoons} \psi^{\mathbf{F}} &&, 1\le j<n\\
\psi^? &\overset{L^T_{n-1}+R^T_{n-1}+r^{\checkmark}_j}{\rightleftharpoons} \psi^{\mathbf{T}} &&, 1\le j<n
\end{aligned}$$

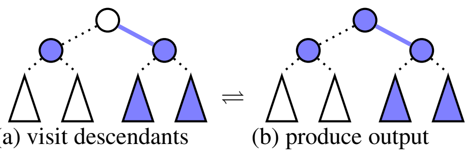


Fig. 8. After both subtrees of the root have been solved a solution can be determined based on the quantifier of the root level. Equations are shown assuming the root variable x_n is universally quantified.

Recall that reactions at level $n-1$ cannot proceed unless the signal $r_n^?$ is present. We could have the reaction producing the solution signal also consume $r_n^?$. This would end the computation chain as only reversing the previous reaction would be possible next. However, for reasons we will make clear in the following section, the signal $r_n^?$ is never altered and therefore after the solution signal is produced, the entirety of the tree traversal steps will be reversed before reaching the end of the computation chain. The entire configuration of the CRN system will appear identical to the initial configuration, with the exception that the output has been written (*i.e.,* the $\psi^?$ signal has been consumed and been replaced by ψ^F or ψ^T). See Fig. 9 for a schematic of the logically reversible computation chain.

Theorem 2. *Any arbitrary instance of* Q3SAT *with n variables and m clauses can be solved by a logically reversible tagged CRN in $O(m\,3^n)$ reaction steps using $\Theta(m+n)$ space.*

Proof. Let ψ be the totally quantified boolean formula of the instance and ϕ be the unquantified 3SAT formula. By Lemma 2 a set of $\Theta(m)$ reactions can be created to verify if ϕ is satisfied, or not, for a particular variable assignment. By Lemma 3, a set of $\Theta(n)$ reactions can be created to traverse the height n tree representing all possible assignments of the n variables. Furthermore, the above modifications demonstrate how these two processes can be integrated into one logically reversible computation chain, and how quantifiers can be added to the non-leaf levels to determine if there is a satisfying solution for ψ by propagating satisfiability of subtrees up to higher levels. Importantly, the modifications only increase the number of reactions by a constant factor and are designed to maintain the property that the computation is logically reversible. Consider that the number of reaction steps acting on a tree node, prior to reaching the root, has not increased. However, prior to marking every leaf in the traversal, the verification procedure is run for the current variable assignment (and reversed in between). Therefore, by Lemmas 2 and 3, the root of the height n tree can be reached, and a solution signal produced, within $O(m\,3^n)$ reaction steps. As forcing the entire tree traversal to reverse prior to the end of computation only doubles the number of reaction steps, the claim on computation length is established.

Next, consider the space required of the combined CRN. The modified verification procedure requires the following initial set, where $\mathcal{T}_{3sat}$ is the set of required tags: $\mathcal{S}_{3sat} = \bigcup_{1 \le i \le m} \{C_i^?, H_i^?\} \cup \{\phi^?, r_2^?\} \cup \mathcal{T}_{3sat}$. The augmented tree traversal procedure requires the following initial set, where $\mathcal{T}_{tree}$ is the set of required tags: $\mathcal{S}_{tree} = \bigcup_{1 \le i < n} \{l_i^?, x_i^?, r_i^?, L_i^?, R_i^?\} \cup \{r_n^?, \phi^?, \psi^?\} \cup \mathcal{T}_{tree}$. The space required for the combined CRN is therefore $|\mathcal{S}_{q3sat}| = |\mathcal{S}_{tree} \cup \mathcal{S}_{3sat}|$. As the combined CRN maintains the property that reactions strictly alternate being applied in the forward and reverse direction, then one tag for each of the $\Theta(m+n)$ reactions is sufficient and $|\mathcal{S}_{tree}| \in \Theta(m+n)$. □

As Q3SAT is a complete problem for PSPACE [11], we immediately have the following.

Corollary 1. *Any problem in* PSPACE *can be solved by a logically reversible tagged CRN using polynomial space.*

4 Space and Energy Efficient DSD Simulation of PSPACE

The remarkable consequence that Bennett's work demonstrates is that energy consumption is not necessarily an intrinsic cost of computation. In particular, if the computation is logically reversible, there is no theoretical energy expenditure. However, there must be a reasonable probability the actual solution can be observed. This can be problematic in a logically reversible computation which is free to immediately reverse once reaching a solution state. Qian *et al.,* [12] solved this problem by using fuel to provide a slight bias for remaining in a solution state once the computation completes. However, in our result, since reactions must be reused efficiently in both directions to maintain a polynomial space bound, they cannot be biased in general.

To overcome this, we have designed our reactions that produce an output signal to ensure the next logical step in the computation is to reverse the tree traversal. This effectively doubles the length of the logically reversible computation chain and established the important property that the output signal can be observed in strictly more than half of the states (see Fig. 9). Notice that this was also the case for Bennett's original reversible Turing machine implementation[1]. As the computation performs an unbiased random walk along the logically reversible computation state space, the steady state probability of observing the output signal is $p > 0.5$. This probability can be further

[1] The forward traversal of the tree, production of the output signal, and reversal of the traversal are analogous to the *compute*, *copy output*, and *retrace* phases of Bennett's original reversible Turing machine simulation [1].

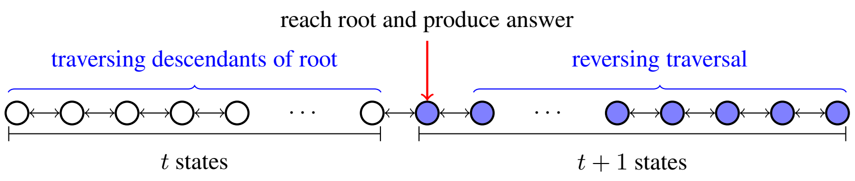


Fig. 9. The logically reversible computation chain of the Q3SAT CRN. In more than half of the states, the output signal is present (shown shaded).

increased in a number of ways. For instance, at the DSD level, we could design the gates which implement the reactions producing the output signal to have a slight bias in the forward direction, by manipulating relative toehold lengths, effectively biasing our overall computation towards the second half of the chain shown in Fig. 9 [17]. As this reaction is only performed once, the gate implementing the reaction is not reused and therefore, the bias is not problematic for the overall computation to complete.
Combining Theorem 1 and Corollary 1 we have the following.

Theorem 3. *Any problem in* PSPACE *can be solved by a space and energy efficient DSD.*

We note that the CRN and DSD description given here is a non-uniform model of computation. Specifically, the CRN description is dependent on, and encodes, a particular problem instance. Therefore, different problem instances will result in different CRN descriptions and thus a different DSD implementation. Particularly at the DSD level, where synthesizing strands and gates is challenging, it would be desirable if only the input strands differed between unique instances. This may be achievable by constructing a more general quantified boolean formula that is within a polynomial size of the original encoding described here. In such a construction, part of the input would describe which clauses are active for the particular problem instance. The generalized formula would be for a fixed number n of variables and could be used to solve any instance having at most n variables. We will explore the details of such a construction in the full version of this manuscript.

5 Complexity of Verifying CRNs and DSDs

Next we show there exists a polynomial time and space reduction from an arbitrary Q3SAT instance I into an instance I' of the CRN reachability problem (Q3SAT $\leq_p$ CRNR), such that I can be solved if and only if I' can be solved.

Theorem 4. *The reachability problem for CRNs (*CRNR*) is* PSPACE*-hard.*

Proof. Given an arbitrary Q3SAT instance, construct the CRN of Theorem 2 which is of polynomial size and can therefore be constructed in polynomial time and space. Ask the question of whether the state $S_{init}/\{\psi^?\} \cup \{\psi^T\}$ can be reached from S_{init}, where S_{init} is the initial state of the CRN. □

By Lemma 1 it is easy to see the reachability problem for proper CRNs is in PSPACE. Whether other forms of CRNs are in PSPACE is dependent on their definition and how the required space to complete a computation is accounted for. Any tagged CRN accounts for the necessary fuel as part of the size of the reaction volume and therefore, by this interpretation, is in PSPACE.

Corollary 2. *The reachability problem for proper/tagged CRNs is* PSPACE*-complete.*

We note that other results are known for unrestricted CRNs which are not studied here. (Un-tagged) reversible CRNs correspond to reversible Petri nets where the reachability problem is EXPSPACE-complete [3]. CRN reachability has also been studied for the probabilistic case [16,18] and nondeterministic case [18] and the connection with Petri nets was also explored [18].

By Theorem 1 and Theorem 4 we immediately have the following analogous results for DSDs.

Corollary 3. *The reachability problem for DSDs (*DSDR*) is* PSPACE*-hard.*

Clearly the reachability problem is PSPACE-complete for the set of DSDs implementing a proper CRN. When fuel molecules are considered part of the space usage, as would be the case for closed volumes that are studied here, then the reachability problem is PSPACE-complete. We also conjecture that a DSD instance created by the above chain of reductions (*i.e.,* Q3SAT $\leq_p$ CRNR $\leq_p$ DSDR), can be adapted to show the minimum energy barrier folding pathway prediction problem is PSPACE-complete. The challenge in achieving this result is to properly consider the possibility of blunt-end displacements, which are generally assumed to not occur when reasoning about DSD systems. We will explore the details of the proof for the full version of the manuscript.

Conjecture 1. The minimum energy barrier pseudoknot-free folding pathway problem for multiple interacting nucleic acid strands is PSPACE-complete.

6 Conclusions

In this work, we asked the question: can space *and* energy efficient computation be realized by chemical reaction networks (CRN) and DNA strand displacement systems (DSD)? We have shown this can be achieved in general by giving a logically reversible space efficient CRN implementation capable of solving any problem in PSPACE—the class of all problems solvable in polynomial space. Furthermore, our CRN can be realized by a space and energy efficient DSD. In addition to further characterizing the computational power of standard molecular programming systems, our result has a number of important consequences. For instance, we show that even determining if a certain state is reachable in a CRN, such as a desirable or undesirable configuration, is PSPACE-hard, effectively demonstrating the intrinsic complexity of model checking and formal verification of chemical reaction networks. We further show the problem is PSPACE-complete for a restricted class of CRNs. The results also hold at the DSD level and give strong evidence of the hardness at the sequence level for the related problem of predicting minimum energy barrier folding pathways between two configurations of multiple interacting nucleic acid strands.

Acknowledgments. We would like to thank the anonymous reviewers for their suggestions and Erik Winfree for helpful discussions regarding energy efficient computation.

References

1. Bennett, C.H.: Logical reversibility of computation. IBM journal of Research and Development 17(6), 525–532 (1973)
2. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM Journal on Computing 18(4), 766–776 (1989)
3. Cardoza, E., Lipton, R., Meyer, A.R.: Exponential space complete problems for Petri nets and commutative semigroups. In: Proceedings of the Eigth Annual ACM Symposium on Theory of Computing, pp. 50–54. ACM (1976)
4. Condon, A., Hu, A., Maňuch, J., Thachuk, C.: Less Haste, Less Waste: On Recycling and Its Limits in Strand Displacement Systems. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 84–99. Springer, Heidelberg (2011)
5. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1. Wiley (1971)
6. Lakin, M.R., Phillips, A.: Modelling, Simulating and Verifying Turing-Powerful Strand Displacement Systems. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 130–144. Springer, Heidelberg (2011)
7. Lakin, M.R., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of DNA strand displacement devices using probabilistic model checking. Journal of the Royal Society Interface (2012)
8. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development 5(3), 183–191 (1961)
9. Lange, K.J., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. Journal of Computer Systems Science 60(2), 354–367 (2000)
10. Cardelli, L.: Two-domain DNA strand displacement. In: Proc. of Developments in Computational Models (DCM 2010). Electronic Proceedings in Theoretical Computer Science, vol. 26, pp. 47–61 (2010)
11. Papadimitriou, C.M.: Computational complexity. Addison-Wesley, Reading (1994)
12. Qian, L., Soloveichik, D., Winfree, E.: Efficient Turing-Universal Computation with DNA Polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
13. Qian, L., Winfree, E., Bruck, J.: Neural network computation with DNA strand displacement cascades. Nature 475(7356), 368–372 (2011)
14. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science 314(5805), 1585–1588 (2006)
15. Shin, J.-S., Pierce, N.A.: A synthetic DNA walker for molecular transport. J. Am. Chem. Soc. 126, 10834–10835 (2004)
16. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing 7(4), 615–633 (2008)
17. Winfree, E.: Personal communication (2012)
18. Zavattaro, G., Cardelli, L.: Termination Problems in Chemical Kinetics. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 477–491. Springer, Heidelberg (2008)

Author Index