Satoshi Murata
Satoshi Kobayashi (Eds.)

LNCS 8727

DNA Computing and Molecular Programming

20th International Conference, DNA 20
Kyoto, Japan, September 22–26, 2014
Proceedings

Springer

Lecture Notes in Computer Science 8727

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Satoshi Murata Satoshi Kobayashi (Eds.)

DNA Computing and Molecular Programming

20th International Conference, DNA 20
Kyoto, Japan, September 22-26, 2014
Proceedings

Springer

Volume Editors

Satoshi Murata
Tohoku University
Graduate School of Engineering
Department of Bioengineering and Robotics
6-6-01 Aoba-yama,
Sendai 980-8579, Japan
E-mail: murata@molbot.mech.tohoku.ac.jp

Satoshi Kobayashi
University of Electro-Communications
Graduate School of Informatics and Engineering
Department of Communication Engineering and Informatics
1-5-1 Chofugaoka Chofu
Tokyo 182-8585, Japan
E-mail: kobayashi.satoshi@uec.ac.jp

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-319-11294-7 e-ISBN 978-3-319-11295-4
DOI 10.1007/978-3-319-11295-4
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014947970

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 20: 20th International Conference on DNA computing and Molecular Programming held during September 22–26, 2014 at Kyoto University, Kyoto, Japan. The DNA conference series aim to draw together computer science, mathematics, chemistry, nanotechnology, molecular biology, and physics to address the analysis, design, and synthesis of information-based molecular systems.

Presentations are sought in all areas that relate to biomolecular computing, including, but not restricted to: algorithms and models for computation on biomolecular systems; computational processes in vitro and in vivo; molecular switches, gates, devices, and circuits; molecular foldings and self-assembly of nanostructures; analysis and theoretical models of laboratory techniques; molecular motors and molecular robotics; studies of fault-tolerance and error correction; software tools for analysis, simulation, and design; synthetic biology and in vitro evolution; applications in engineering, physics, chemistry, biology, and medicine.

Authors who wished to present their works were asked to select one of two submission tracks: Track (A) (full paper) or Track (B) (one page abstract with supplementary document). Track (B) is primarily for authors submitting experimental results who plan to submit to a journal rather than publish in the conference proceedings.

We received 55 submissions for oral presentations: 20 submissions in Track (A) and 35 submissions in Track (B). Each submission was reviewed by at least 3 reviewers. The Program Committee finally decided to accept 10 papers in Track (A) and 13 papers in Track (B). The topics of accepted presentations are well balanced between theoretical and experimental works. This volume contains the accepted Track (A) papers.

We express our sincere appreciation to Shawn Douglas, Cristian S. Calude, Hiroshi Sugiyama, Rhiju Das, Anne Condon, and Masaki Sano for their excellent keynote talks. Thanks are also given to all the authors who presented their works at oral and poster sessions. We would like to thank Nadrian C. Seeman and Hiroyuki Asanuma for their excellent special talks in a one-day workshop on molecular robotics. Last but not least, the editors would like to thank the members of the Program Committee and the reviewers for all their hard work of reviewing and providing constructive comments to authors.

July 2014

Satoshi Murata
Satoshi Kobayashi

Organization

DNA 20 was organized by Kyoto University in cooperation with the Internatinal 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	UEC Tokyo, 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	University of Oxford, UK
Hao Yan	Arizona State University, USA
Erik Winfree	California Institute of Technology, USA

Organizing Committee

Hirohide Saito (Co-chair)	Kyoto University, Japan
Masayuki Endo (Co-chair)	Kyoto University, Japan
Akinori Kuzuya (Co-chair)	Kansai University, Japan

Program Committee

Satoshi Murata (Co-chair)	Tohoku University, Japan
Satoshi Kobayashi (Co-chair)	UEC Tokyo, Japan
Robert Brijder	Hasselt University, Belgium
Luca Cardelli	Microsoft Research, UK
David Doty	California Institute of Technology, USA
Andrew Ellington	The University of Texas at Austin, USA
Andre Estevez-Torres	CNRS France
Kaoru Fujioka	Fukuoka Womens University, Japan
Shougo Hamada	Cornell University, USA
Natasha Jonoska	University of South Florida, USA

Lila Kari	University of Western Ontario, Canada
Steffen Kopecki	University of Western Ontario, Canada
Akinori Kuzuya	Kansai University, Japan
Chengde Mao	Purdue University, USA
Pekka Orponen	Aalto University, Finland
Jennifer Padilla	Boise State University, USA
John Reif	Duke University, USA
Alfonso Rodriguez-Paton	Universidad Politecnica de Madrid, Spain
Yannick Rondelez	University of Tokyo, Japan
Hirohide Saito	Kyoto University, Japan
Robert Schweller	University of Texas-Pan American, USA
Shinnosuke Seki	Aalto University, Finland
Friedrich Simmel	Technische Universität München, Germany
David Soloveichik	UCSF, USA
Darko Stefanovic	University of New Mexico, USA
Chris Thachuk	University of Oxford, UK
Andrew Turberfield	University of Oxford, UK
Erik Winfree	California Institute of Technology, USA
Damien Woods	California Institute of Technology, USA
Peng Yin	Harvard Medical School, USA
Bernard Yurke	Boise State University, USA
Byoung-Tak Zhang	Seoul National University, South Korea

External Reviewers

Banda Peter
Carl Brown
Mingjie Dai
Srujan Kumar Enaganti
Nikhil Gopalkrishnan
Alireza Goudarzi
Hiroaki Hata
Kojiro Higuchi
Alexandra Keenan
Manasi Kulkarni
Matthew R. Lakin
Cameron Myhrvold
Luvena Ong
Kai Salomaa
Amirhossein Simjour
Wei Sun
Sungwook Woo

Sponsoring Institutions

Kyoto University
The Kyoto University Foundation
Grant-in-Aid for Scientific Research on Innovative Areas “Molecular Robotics”, MEXT, Japan
The Uehara Memorial Foundation
Kato Memorial Bioscience Foundation
Support Center for Advanced Telecommunications Technology Research, Foundation

Table of Contents

Design Principles for Single-Stranded RNA Origami Structures

Cody W. Geary and Ebbe Sloth Andersen[*]

Interdisciplinary Nanoscience Center, Aarhus University,
Gustav Wieds Vej 14, 8000 Aarhus, Denmark
esa@inano.au.dk

Abstract. We have recently introduced an experimental method for the design and production of RNA-origami nanostructures that fold up from a single strand while the RNA is being enzymatically produced, commonly referred to as cotranscriptional folding. To realize a general and scalable architecture we have developed a theoretical framework for determining RNA crossover geometries, long-distance interactions, and strand paths that are topologically compatible with cotranscriptional folding. Here, we introduce a simple parameterized model for the A-form helix and use it to determine the geometry and base-pair spacing for the five types of RNA double-crossover molecules and the curvature resulting from crossovers between multiple helices. We further define a set of paranemic loop-loop and end-to-end interactions compatible with the design of folding paths for RNA structures with arbitrary shape and programmable curvature. Finally, we take inspiration from space-filling curves in mathematics to design strand paths that have high-locality, programmed folding kinetics to avoid topological traps, and structural repeat units that might be used to create infinite RNA ribbons and squares by rolling circle transcription.

Keywords: RNA, structure, folding, kinetics, space-filling curves.

1 Introduction

RNA molecules have a greater structural diversity than DNA and are thus of interest as design substrate for biomolecular engineering. The field of RNA nanotechnology has taken inspiration from the modular nature of structurally characterized RNA molecules to develop a design paradigm where structural modules can be composed to achieve complex geometric shapes and lattices [13]. However, RNA modular design is currently limited to rather small RNAs, suggesting that there is still a need to develop a more standardized design approach such as that used in DNA nanotechnology [2] and best exemplified by the DNA origami method [22]. Particularly, in DNA nanotechnology the double-crossover (DX) motif plays a central role in organizing DNA helices into large arrays, but the RNA field has yet to leverage RNA DX motifs to build similarly large structures.

[*] Corresponding author.

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 1–19, 2014.

DX molecules have been described extensively for B-form DNA helices [9], but to our knowledge not in great detail for A-form RNA helices, which we will now describe. Several papers have revealed that using a simplified model considering only the helical twist of 11 base pairs (bps) per turn for RNA is not sufficiently accurate for fully realizing design in RNA. For example, for the RNA/DNA hybrid design of Mao and colleagues it was revealed that the inclination of the bases in RNA influences the tiling behavior of the RNA/DNA nanostructures [17]. Likewise, in Delebecque et al. RNA assemblies designed considering only the helical twist required the inclusion of unpaired bases within the tile [5], which we here propose generate a required flexibility to counteract the distance offset caused by the base-pair inclination. Furthermore, studies of RNA assemblies based on the paranemic crossover motif (PX) have found that the ideal tile designs differ significantly from the DNA versions, due to the difference in width of the major and minor grooves of RNA compared to DNA [1]. Recently, we initiated the design and testing of RNA double-crossover molecules with crossover spacings based on calculated distances and three-dimensional (3D) modeling, and demonstrated very robust formation of RNA DX motifs [12]. Here, we extend upon this work and describe the DX motifs for RNA in detail, and propose a new method for the systematic design of RNA nanostructures.

A useful feature of RNA structure is that it can be produced directly from the RNA polymerase enzyme by the cotranscriptional self-assembly process. In this process the RNA folds locally as it is transcribed and the structure gradually builds up as more sequence is produced by the enzyme. The autonomous enzymatic process furthermore allows engineered RNA structures to be genetically encoded and expressed in cells, much like natural RNAs. We have recently demonstrated the design of single-stranded RNA origami structures and their production by both heat-annealing and/or cotranscriptional folding [12]. To develop this method further we consider the cotranscriptional folding process as it relates to requirements for the order of folding and assembly events along the RNA strand, and propose geometrical and topological principles for single-stranded RNA nanostructures.

2 Simple Model of a Double Helix

In DNA nanotechnology simplified helices are often used when designing larger constructs to decrease the computational requirements for handling large 3D objects, since only a few parameters are needed to determine crossover positions [24]. By contrast, many artificial RNA nanostructures are still designed using fully-atomistic models [23,14], in a tedious process requiring specialized experience, or by using over-reduced models that do not capture important details of the RNA helix [5]. Thus, there is a need for an appropriately simplified RNA model that can still be easily manipulated. Here, we propose such a model that is a parameterization describing only the positions of phosphate atoms on the helix, which we call the “P-stick model”. Our model generates the atomic positions of phosphates relative to the central axis of the helix, based on five parameters: base inclination relative to the helix-axis, rise between bases, helix

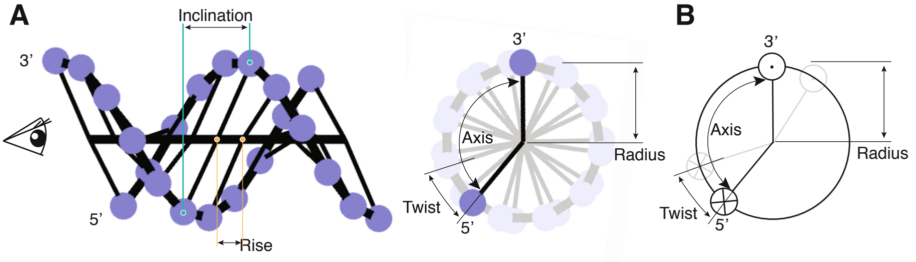


Fig. 1. P-stick model for the A-form helix. (A) Left: Model of A-form helix shown in side view with indications of inclination and rise distances along helical axis. Right: End view of the same A-form helix model as seen along the helical axis from the left as indicated by the eye symbol. The radius, axis and twist angles are indicated. Lines connecting the phosphates to the central axis depict the connectivity of the helix and do not represent base-pairs. **(B)** Schematics of P-stick model with 3' end indicated by circle with a dot (tip of arrow) and 5' end indicated by a circle with a cross (end of arrow). The second bp is shown in grey to indicate twist angle and the right handed rotation of the helix.

Table 1. Parameters for P-stick helix model. Parameters are measured by the authors based on data from Arnot et. al. 1973 [3] and Dickerson et al. 1982 [6]. We also note that the inclination variable measures the inclination of paired-phosphates relative to the helical axis, and should not be confused with base-pair inclination.

Parameters	Variables	A-form	B-form
Radius	R	8.7 Å	9.3 Å
Rise	D	2.81 Å	3.4 Å
Inclination	I	-7.45 Å	3.75 Å
Axis	A	139.9°	170.4°
Twist	T	32.73°	34.48°
Helicity	H	11 bp	10.44 bp

radius, axis angle across the minor groove, and helicity in bps per turn (described in Fig 1A).

Parameters for RNA A-form helices were measured from standard helices generated by Westhof's Assemble2 program [16], which are based on classical A-form parameters [3]. Parameters for DNA B-form helices were derived from various literature sources: The helicity was chosen to be 10.44 bps/turn based on NMR measurements in solution [26] and parameters found to be the optimal for producing twist-corrected DNA origami structures [27]. The inclination between phosphates was found by aligning the B-DNA crystal structure (PDB-ID 1BNA) [7] and averaging for all phosphate positions giving a value of 3.748 Å. Parameters for both A- and B-form helices are provided in Table 1.

The P-stick model consists of two equations that generate xyz coordinates for the phosphate atoms along each strand of a helix, with the axis centered on the origin pointing along the x-axis:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p \cdot D \\ R \cdot \cos\left(p \cdot \left(\frac{2\pi}{H}\right)\right) \\ R \cdot \sin\left(p \cdot \left(\frac{2\pi}{H}\right)\right) \end{bmatrix},\ p|p \in \mathbb{Z} \tag{1}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p \cdot D + I \\ R \cdot \cos\left(p \cdot \left(\frac{2\pi}{H}\right) + \left(\frac{A \cdot \pi}{180^\circ}\right)\right) \\ R \cdot \sin\left(p \cdot \left(\frac{2\pi}{H}\right) + \left(\frac{A \cdot \pi}{180^\circ}\right)\right) \end{bmatrix},\ p|p \in \mathbb{Z} \tag{2}$$

Eq(1) describes the strand running from 3' to 5' along the x-axis and Eq(2) describes the complement strand running 5' to 3'.

3 RNA Crossover Motifs

The alignment of phosphates between helices is taken as an indicator of positions where a crossover junction can join the helices [24]. Here we use the P-stick model to predict optimal crossover spacings for A- and B-form helices. To construct a crossover between two helices we first align one pair of phosphates between the helices, then break the backbone bond on the same side of each phosphate and rejoin them with the opposite strand. Likewise, to construct a DX between two helices we orient the helices in parallel and rotate the helices to align two pairs of phosphates in the plane, and perform two crossover operations on the backbone.

3.1 The RNA DX Types

DXs can either be formed between anti-parallel (A) or parallel (P) strands, and the spacing between the two crossovers (measured by the number of half turns) can either be even (E) or odd (O). Even-spaced crossovers result in both crossovers bridging the same strand, while odd-spacing results in the two crossovers formed between opposite strands (Fig. 2). Thus, DX molecules can take the form of AE, AO, PE and PO. Two possibilities exist for PO since the DX can bridge either the major or minor groove. For B-form helices the major groove is wide (W) and the minor groove is narrow (N), and the DXs are thus called PON and POW [9]. For A-form helices the major groove is narrow and deep (D) and the minor groove is wide and shallow (S), and thus we propose to name the A-form PO-DX motifs POD and POS. PO-DXs with more than one groove spanning the crossovers are named depending on which groove is predominant. PO-DXs that span a single groove width have the property of being paranemic, and have been described and studied for both DNA and RNA, although with non-ideal spacings for RNA[1]. Interestingly, the DX molecules all have different pseudosymmetry axes (shown in Fig. 2) that are important for their use as building blocks, which we will discuss later.

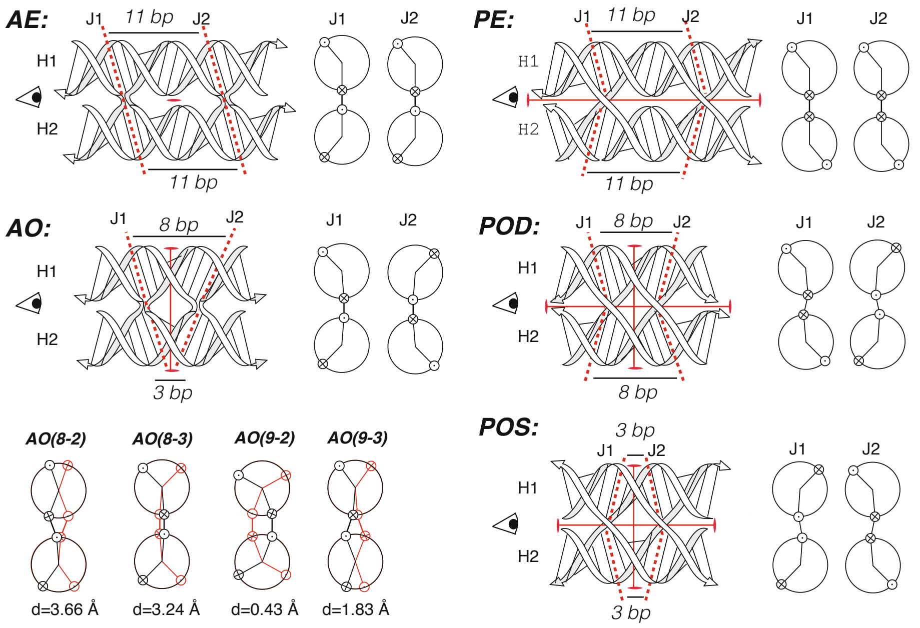


Fig. 2. Minimal A-form DX molecules. A-form DX of type AE, AO, PE, POD and POS (see section 3.1 for naming conventions) that can be expanded by $11 * n$ bps. **Left:** Ribbon diagrams show how the helix inclination affects the base-pair spacing between the two crossovers. Red dashed lines show the junctions and their inclinations. Base-pair spacing is indicated above and below, with horizontal lines. Red lines and ellipse shapes show C2 pseudosymmetry axes. Ribbon diagrams are based on P-stick models and drawn using positions of P and C3 atoms. Helices (H) and junctions (J) are numbered from top to bottom and left to right, respectively. **Right:** The two junctions are shown with schematics similar to Fig. 1B as seen along the helical axis from the eye symbol. The DX adjusted rotation of the helices are shown. **Bottom left:** Four AO DXs with different spacing are shown in side view, where black diagram is J1 and red diagram is J2. The distance between P atoms at crossovers is written below each side view.

3.2 Spacing between A-Form DXs

We use the P-stick model to calculate optimal crossover positions, which is done by measuring the distance between pairs of phosphates on two strands and finding the pairs for which the distances are at a minimum. For each crossover pattern that we analyze, we provide distance and angular measurements to qualify how well-aligned the crossover pattern is (Table 2). The distance between two crossovers (DX-$dist_x$) is measured from the midpoint between the phosphates of one crossover to the midpoint of phosphates of the second crossover. The angular gap (ΔAngle) of the crossovers is calculated by perfectly aligning the phosphates of the first crossover, and then measuring the angular offset of the phosphates from the center-line of the second crossover. We calculate the bp spacing, derived from the number of phosphates between the crossovers, and the direction of the strands as they pass through the crossovers. Because of the different strand directionality in odd- versus even-spaced junctions, the number of bps in the junction have the following relations:

$$\begin{aligned} bp &= P - 1 \quad \text{for AE and PE} \\ bp &= P \quad \text{for AOD and POD} \\ bp &= P - 2 \quad \text{for AOS and POS.} \end{aligned}$$

Thus, for AO junctions where the minor groove of the first helix (H1) aligns with the major groove of the second helix (H2), the two strands must be considered differently. In Table 2, bp values (b_2,b_1) for each set of phosphate spacings (p_2,p_1) are reported.

An interesting feature of all AO- and PO-DXs is that the crossovers occur between opposite strands which, due to the incline of the phosphates, results in an asymmetry. The asymmetry is much more apparent for A-form RNA helices than for B-form DNA helices, and results in a substantial difference in the number of bps on each side of the crossover. Our calculations for the optimal AO-DX (Fig. 2) find that there should be 9 bps in H1 and only 2 bps in H2 (Table 2) and that the phosphates do not perfectly align, resulting in a large 42° angular gap between the crossovers, but a short distance of 0.4 Å between the two crossover phosphates along the helical axis.

Likewise, PO-DXs in RNA have different spacings depending on whether they cross the deep groove (POD) or the shallow groove (POS) of the RNA, where POD-DX have a spacing of 8 bp on each helix and a distance of 12.2 Å between crossovers and POS-DX have only 3 bp on each helix and a distance of 18.7 Å between crossovers. All spacings for RNA crossovers can be extended by $n * 11$ bps, for positive integer n. Spacings for DNA crossovers can be extended by roughly $n*21$ bps, but because of the 10.44bp/turn helicity they are not perfectly equivalent.

Using the previously defined variables for Eq(1), we define the alignment of two helices with parameters that translate and rotate H2 relative to H1, such that the two helices are aligned with phosphate-1 of each strand superimposed. The primary strand is thus defined similarly to Eq(1), where p_1 represents the nth phosphate in strand 1 on H1:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} (p_1 - 1) \cdot D \\ R \cdot \cos\left(\frac{2\pi(p_1-1)}{H}\right) \\ R \cdot \sin\left(\frac{2\pi(p_1-1)}{H}\right) \end{bmatrix} \tag{3}$$

Likewise, the complement strand 2 is defined by a shift and rotation about the helical axis by the axis and inclination variables, similar to Eq(2):

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} (p_1 - 1) \cdot D + I \\ R \cdot \cos\left(\frac{2\pi(p_1-1)}{H} + \left(\frac{A\cdot\pi}{180^\circ}\right)\right) \\ R \cdot \sin\left(\frac{2\pi(p_1-1)}{H} + \left(\frac{A\cdot\pi}{180^\circ}\right)\right) \end{bmatrix} \tag{4}$$

For the purpose of calculating the crossover spacing distances, H2 is aligned such that its phosphate-1 is superimposed on the phosphate-1 of H1, where variables for the translation and rotation of H2 are defined as follows:

$$Tr_x = I \tag{5}$$

$$Tr_y = R \cdot \cos\left(A \cdot \left(\frac{\pi}{180^\circ}\right)\right) - R \cdot \cos(\theta) \tag{6}$$

$$Tr_z = R \cdot \sin\left(A \cdot \left(\frac{\pi}{180^\circ}\right)\right) - R \cdot \sin(\theta) \tag{7}$$

where $\theta = \pi + A \cdot \left(\frac{\pi}{180^\circ}\right)$. The directionality of H2 is changed by $\delta = 1$ for antiparallel helices and $\delta = -1$ for parallel helices giving the following equations, where p_2 is the nth phosphate:

$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} (p_2 - 1) \cdot D + Tr_x \\ R \cdot \cos\left(\frac{2\pi\cdot(p_2-1)}{H} + \theta\right) + Tr_y \\ R \cdot \sin\left(\frac{2\pi\cdot(p_2-1)}{H} + \theta\right) + Tr_z \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix} = \begin{bmatrix} (p_2 - 1) \cdot D + \delta \cdot I + Tr_x \\ R \cdot \cos\left(\frac{2\pi\cdot(p_2-1)}{H} + \frac{\delta\cdot A\cdot\pi}{180} + \theta\right) + Tr_y \\ R \cdot \sin\left(\frac{2\pi\cdot(p_2-1)}{H} + \frac{\delta\cdot A\cdot\pi}{180} + \theta\right) + Tr_z \end{bmatrix} \tag{9}$$

The distance between phosphates is then calculated by measuring the euclidian distance between p_1 and p_2 on the pair of strands forming the crossover:

$$dist_{AO} = \sqrt{(x_1 - x_4)^2 + (y_1 - y_4)^2 + (z_1 - z_4)^2} \tag{10}$$

$$dist_{AE} = \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2 + (z_2 - z_3)^2} \tag{11}$$

Thus, for any crossover domain of length p_1 and p_2, we need only evaluate the equations to get the table of distances (P-$dist_{xyz}$ in Table 2).

Using the P-stick model we have calculated parameters for what we propose are the ideal spacings for all four DX types in both A-form and B-form helices.

Table 2. Base-pair spacing for A- and B-form DX molecules. Numbers are given for phosphate and base-pair spacings on helix 1 and 2. The spacings are minimum spacings and can be extended by full helical turns (+11 bps for RNA, and +21 bps for DNA). DX-$dist_x$ is the distance between P-P midpoints of the two crossovers. P-$dist$ is the P-P distance at the J2 crossover. *Negative numbers for p_1 and p_2 are converted to phosphate spacings by taking: -p+2. **Has symmetry equivalent: $(p_2,p_1) = (p_1,p_2)$.

Helix	DX	(p_2,p_1)	(b_2,b_1)	DX-$dist_x$ (Å)	ΔAngle (°)	P-$dist_x$ (Å)	P-$dist_{xyz}$ (Å)
A	AE/PE	(12,12)	(11,11)	30.9	0	0	0
	AO	(8,4)	(8,2)	14.1	25.4	1.83	6.20
	AO	(8,5)	(8,3)	15.5	9.0	3.24	6.48
	AO	(9,4)	(9,2)	15.5	41.7	0.43	4.49
	AO	(9,5)	(9,3)	16.9	25.4	1.83	6.20
	AO	(18,20)	(18,18)	50.6	72.8	10.26	25.17
	POD	(-6,-6)*	(8,8)	12.2	9.0	0	2.73
	POS	(5,5)	(3,3)	18.7	9.0	0	2.73
B	AE/PE	(22,22)	(21,21)	71.4	4.1	0	1.34
	AE/PE	(11,11)	(10,10)	34.0	15.2	0	4.91
	AE/PE	(11,12)**	(10,11)	35.7	17.2	1.7	3.56
	AO	(5,7)	(5,5)	17.0	44.1	0.3	5.68
	AO	(16,17)	(16,15)	52.7	26.8	2.0	4.86
	AO	(16,18)	(16,16)	54.4	44.1	0.3	6.19
	AO	(26,28)	(26,26)	88.4	44.1	0.3	5.37
	POW	(-4,-5)*,**	(6,7)	22.4	17.2	1.7	3.50
	POW	(-15,-15)*	(17,17)	58.1	2.1	0	0.69
	PON	(6,6)	(4,4)	13.3	2.0	0	0.66
	PON	(17,16)	(15,14)	49.0	17.2	1.7	3.73
	PON	(16,16)	(14,14)	47.3	13.2	0	4.26

In addition, we report measurements for commonly used spacings in the literature, for comparison (Table 2). Interestingly, we find that some of the commonly used DX spacings are not the best possible arrangements. However, we also note that non-ideal phosphate spacings may still be preferential for symmetry reasons, or out of a need to have domains of the same length, and thus might represent necessary compromises for structural or thermodynamic reasons that are not considered in our model. Of particular interest, we find that for narrow DNA-AE crossovers a 10/11 bps ((b_2/b_1), respectively) arrangement results in shorter P-$dist_{xyz}$ than a 10/10 bp spacing. Likewise, we find that DNA-AO crossovers have an asymmetry of 16/15 bps which is better than the regularly used 16/16 bp spacing, consistent with early modeling predictions for DNA [9]. For DNA crossovers, we find that the shortest crossover gaps are a PON crossover with 4/4 bps spacing and a POW crossover with 17/17 bps spacing. By contrast in RNA, the 18/18 bps spacing used in Delebecque et al. [5] to form RNA-AO crossovers has a large P-$dist_{xyz}$, likely explaining why the experimenters needed several unpaired bases to provide the necessary flexibility to form these crossovers.

3.3 Multiple Crossovers and Curvature

The DX motifs can be further combined into multiple crossover (MX) molecules. We define the distance between DX molecules in a MX molecule as follows: position 1 (n_1) is set at the first crossover between H1 and H2, where the 5' end of H1 enters the crossover from the left. Position 2 (n_2) is the position of the second crossover between H2 and H3. The spacing (s) is given as the difference between the crossover positions:

$$s = n_2 - n_1$$

and relates to the number of basepairs between the two crossovers. Only certain spacings are possible since some spacings will bring the helices to overlap and sterically clash.

Depending on the DX type, only certain spacings are allowed for producing MX arrays without steric clashes. For RNA-AE, spacings of

$$s = -5, -4, -3, -2, -1, 0, 1, 2$$

are allowed, as well as spacings of $s - 11 * n$. For RNA-PE, spacings of

$$s = -2, -1, 0, 1, 2, 3, 4, 5$$

are allowed, plus spacings of $s - 11 * n$. For both AE and PE, the curvature of multiple helices can be controlled in this fashion. AO has very strict spacing restrictions when the crossovers are placed in close proximity, since the two junctions bend in opposite directions (Fig. 3). This is because AO-DX has an even/odd number of bases in the shallow/deep groove, respectively, and it is thus not possible to have symmetrically opposed curvature on both junctions with this spacing. Thus, to obtain a symmetric molecule we choose a less optimal DX spacing of 9-3 (see Fig. 2 and Table 2), which results in a MX spacing of $s = -3$ between the three helices. The curvature is expected to cancel out by pushing the crossovers in opposite directions (see 3H-AO in Fig. 3) and this molecule has been tested and found to assemble [12]. Similarly, the POS and POD crossovers for RNA also require consideration of symmetry if they are to be used in MX arrays.

Interestingly, another consequence of symmetry is that if one forms MX arrays by repeating a PE crossover pattern then the bending of the array evens out because the structure becomes corrugated with alternating ridges and valleys. Likewise, for a continuous sheet of POS and POD crossovers, the bending also evens out internally in the sheet but results in a strained although flat conformation of the sheet.

The most flat RNA-AE spacing is $s = -1*(n*11)$, which bends only 7 degrees into the plane. However, as the spacing can be variegated row-by-row, another way to make planar sheets of AE or PE DXs is to alternate between two or more different spacings e.g. $s = 0, -2, 0, -3$ for AE as will be used later to construct large planar AE strand paths.

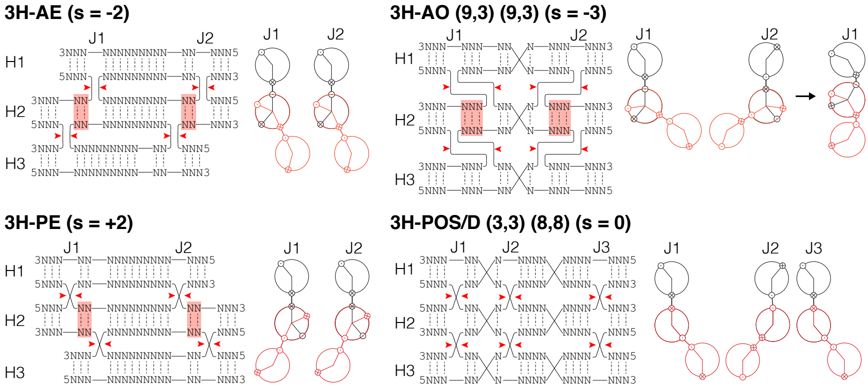


Fig. 3. Crossovers between three helices. RNA 3-helix double-crossover molecules shown as strand diagrams and in side-view. Helices (H) and junctions (J) are numbered from bottom to top and left to right, respectively. Spacing (s) between the sets of crossovers joining the three helices are marked in parenthesis behind the name of the motif, and shown in red on the strand diagrams. Red arrows indicate the positions of the crossovers. H1 and H2 are fixed in their common plane. H3 (marked in red) is rotated depending on the crossovers spacing. For AO crossovers, the curvature induced by crossovers on each side are in opposition and are expected to cancel out by deforming the geometry of the crossover junction, as indicated at the top right.

4 Single-Stranded RNA Origami

We have recently introduced a general architecture called single-stranded RNA origami, where we use the crossover motifs introduced above to fold a single RNA strand into complex topologies [12]. To force the otherwise multi-stranded architecture into a single-stranded form we use kissing-loop interactions to bridge the crossover junctions (Fig. 4), and use minimal spacings at helix junctions (named dovetail seams). The art of folding RNA structures from a single strand without getting into topological problems will be discussed below.

4.1 Paranemic Connectors

The structural diversity of RNA provides numerous long-range RNA-RNA interactions which can be used to form programmable contacts between specific elements in the RNA structure. Long-range interactions can be especially useful when considering crossover patterns that would otherwise require many separate strands. Many different connectors can be found from natural RNA structures, some of which are listed in Fig. 5. Loop-loop interactions are a class of pseudoknot that form between preformed secondary structure elements, bringing distant elements in a 2D structure together in a sequence-programmable fashion. The kissing-loops are typically in the range of 2-7 bps, and are limited to be less than one helical turn in order to avoid being topologically entangled.

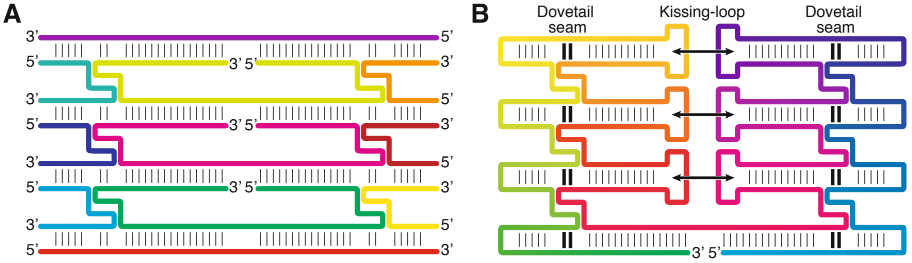


Fig. 4. Multi-stranded to single-stranded conversion. **(A)** Eleven strands in a multi-crossover motif between four helices. **(B)** Loops are inserted to convert the motif into a single-stranded architecture. Loops on the edges cap the end of the helix. Loop-loop interactions are inserted inside the structure to connect the helices across the crossover junctions.

Loop-receptor interactions (Fig. 5C) represent an entirely different class of programmable long-range interaction that have been extensively studied [4,15,10]. Unlike kissing-loop interactions, these tertiary contacts do not form any Watson-Crick bps in their assembly, but are still highly sequence specific. Because they do not form bps with each other, loop-receptor interactions do not form pseudoknotted structures, and also have a higher salt requirement compared to kissing-loop interactions. These properties make loop-receptor interactions of interest for use as paranemic connectors because they likely fold in a different time scale compared to kissing loops and are a completely orthogonal class of interaction from kissing loops.

Lastly, RNA-PX crossovers represent a third class of interaction that can be implemented sequence-specifically [1]. Of all the paranemic connectors we have presented here, PX crossovers are the only variety that have been implemented in DNA [25,28].

4.2 Dovetail Seams

Aside from paranemic type connectors, helices can also interact by stacking at their helix ends. In the context of single-stranded RNA origami we call these end-end interactions “dovetail seams” because they can click rigidly together without creating topological problems. The dovetail bps are only possible when the structure is at least three helices wide, and to our knowledge, there are no natural RNA molecules that adopt this architecture. The interesting feature of dovetail seams is that they can be used to program a specific stacking conformation of a junction, as well as define the curvature of the single-stranded RNA origami tile. The small stems formed by the dovetail seams are in themselves a secondary structure motif and do not increase the topological complexity of the structure. Additionally, as discussed in section 3.3, only certain spacings are allowed. For AE crossovers, the dovetail seams can have spacings of

$$s = -5, -4, -3, -2, -1, 0, 1, 2.$$

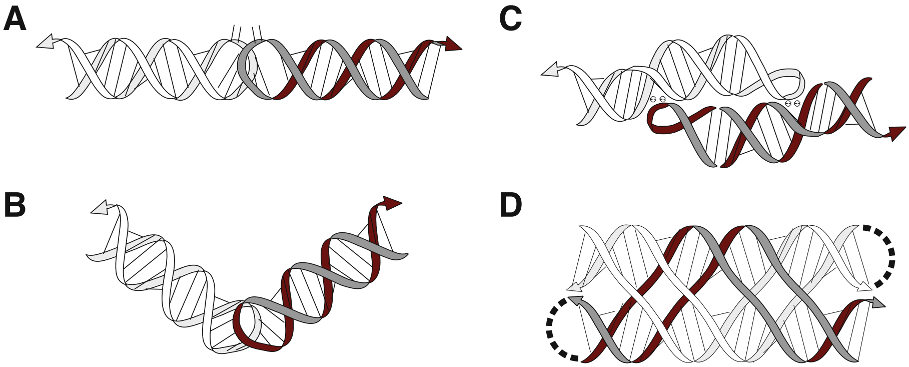


Fig. 5. Examples of paranemic interactions. (A) 180° kissing-loop interaction from HIV-1 [8]. **(B)** 120° kissing-loop motif [19]. **(C)** Loop-receptor interaction [4,15,10]. **(D)** Paranemic crossover motif PX [1].

The other DX types have similar allowable dovetail seam spacings, although we will focus on AE crossovers here as it is the simplest example.

4.3 Folding Path Design

The different types of DX lattices can all be used as the basis of designing single-stranded folding paths, simply by inserting loop-loop interactions as shown in Fig. 4. The AE crossover pattern is especially easy to design since it has a regular folding path and the curvature can be precisely controlled by the choice of dovetail spacings. As illustrated in Fig. 6, an 8 helix tall structure can be expanded in two-dimensions by utilizing multiple dovetail seams, and through the choice of the dovetail spacings on each row can be programmed to be relatively flat. Long helices can potentially cause topological problems, as they require a complement strand that wraps around the helix by more than a full turn. This problem can be circumvented by careful strand-path design to minimize the number of long helices, placing the long helices where they will be less likely to generate a trap, and by choosing the location of the 3' end so that it wraps less than one helical turn at the end of the folding proces. As a general rule we find that it is important that long helices are completely formed before pseudoknotted long-range interactions form connections that block access to the helix. In analogy to tying a knot, we wish to avoid cases where the end of the rope needs to be threaded back through the structure, and thus knots should be formed only in the bight of the rope. Even more complicated strand paths can be designed (Fig. 6), and in these cases the choice of strand path and position of the 5' and 3' ends becomes an even more important factor.

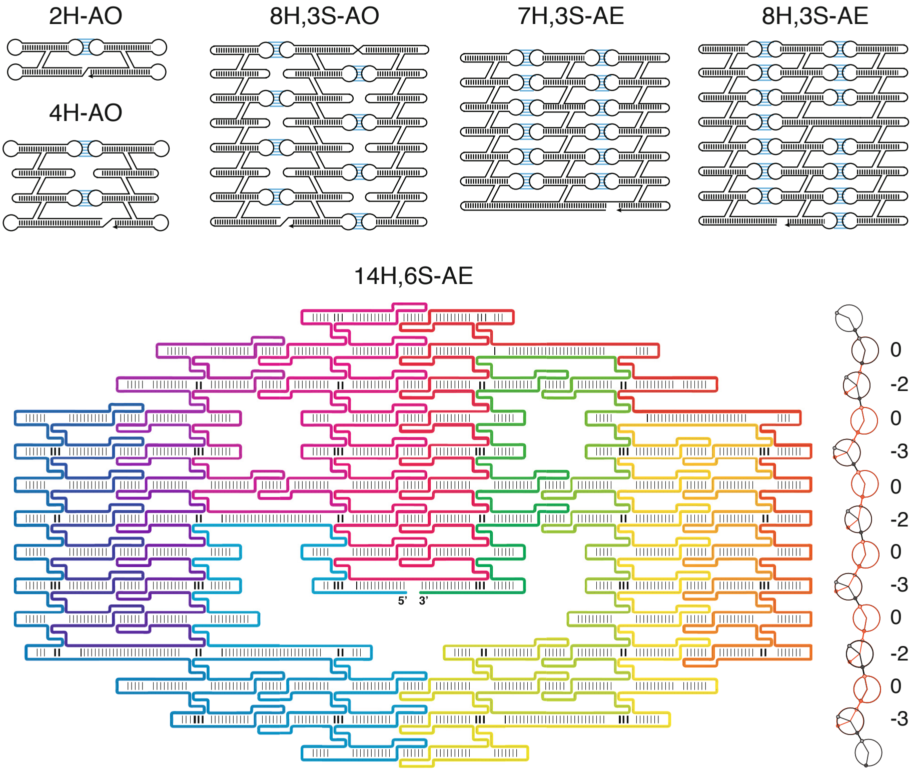


Fig. 6. Folding paths of single-stranded RNA origami structures. Top: Various strand paths for RNA origami in different size ranges, to illustrate how AO and AE junctions might be incorporated into arbitrary designs. The designs are named by the number of helices tall (H), the number of dovetail seams wide (S), and the type of DX used (AO or AE).AO strand paths are more intertwined than AE strand paths, and thus for larger designs AE patterns might be preferable. **Bottom:** An example of a single-stranded RNA origami with a complicated arbitrary strand path, here inspired from the DNA origami smiley face [22]. The dovetail junctions must have the same spacing within any given row to minimize strain in the structure. A particularly flat combination of spacings for AE is $s = 0, -2, 0, -3$, shown in side-profile at the right.5' and 3' ends are found near the middle of the structure (on the upper lip). The strand path has been colorized to highlight the order of synthesis.

Depending on the choice of the loop-loop placement within the strand path, the locality of interactions can either be decreased or increased. We choose to design structures by maximizing the locality of long-range interactions, with the goal of helping the kinetic folding of loops and their cognate partners. Following this logic, the shorter time between the expression of a loop and the expression of its partner loop, the less likely that these loops are to bind in the wrong place, especially if degenerate loop sequences are used. In terms of co-transcriptional folding where the 5' end is made first by the RNA polymerase and has time to fold before downstream sequence is produced, we believe this design principle can help to narrow choices in strand path design. In particular, locality seems to be beneficial for two main reasons: 1) the structural core forms faster and new elements can be added gradually, 2) loop-loop interaction sequences can be reused if the former loop is already base paired in the structure. The timing of formation of loop-loop interactions can be further programmed by choosing loop-loop interactions with different association-dissociation constants. Additionally, it may be possible to even further tune the folding pathway by implementing pausing sites for the RNA polymerase [18], by which a short 16 nucleotide sequence can be programmed into the DNA that causes the polymerase to pause for long enough to allow time for one folding event to complete before the next structure is synthesized.

4.4 Space-Filling Folding Paths

We have found inspiration in space-filling curves from mathematics for designing RNA folding paths with high locality. Some space-filling curves have similar strand paths to the structures shown in Fig. 6. In particular, the Peano curve has a pattern (Fig. 7B, top) that can be interpreted as RNA helices with dovetail seams and kissing-loop interactions (Fig. 7B, bottom). This strand path differs from natural RNAs in that sequences are left unpaired and that the 5' and 3' ends do not meet, where most natural RNAs fold into structures resulting in the two ends co-localized [29]. The Peano-inspired RNA structure also has some long stems internal to the structure that could form potential topological problems. However, these topological barriers can be bypassed with the ability to control the speed of transcription, pausing sites, and strength of kissing-loop interactions. We have highlighted all of the barrier-forming kissing loop interactions in orange and indicated the point in the sequence (shown as orange dots) in which the 2D structure is required to fold up to before the kissing loops form. Additionally, variations of the repeating pattern in the Peano-curve with either fewer or more kissing-loop interactions are expected to have an effect on the fold (Fig. 7A). We present a second Peano-inspired curve (Fig. 7C, top) and its translation into RNA secondary structure (Fig. 7C, bottom), to further illustrate the large variety of possible space-filling patterns that can be translated into RNA strand path patterns.

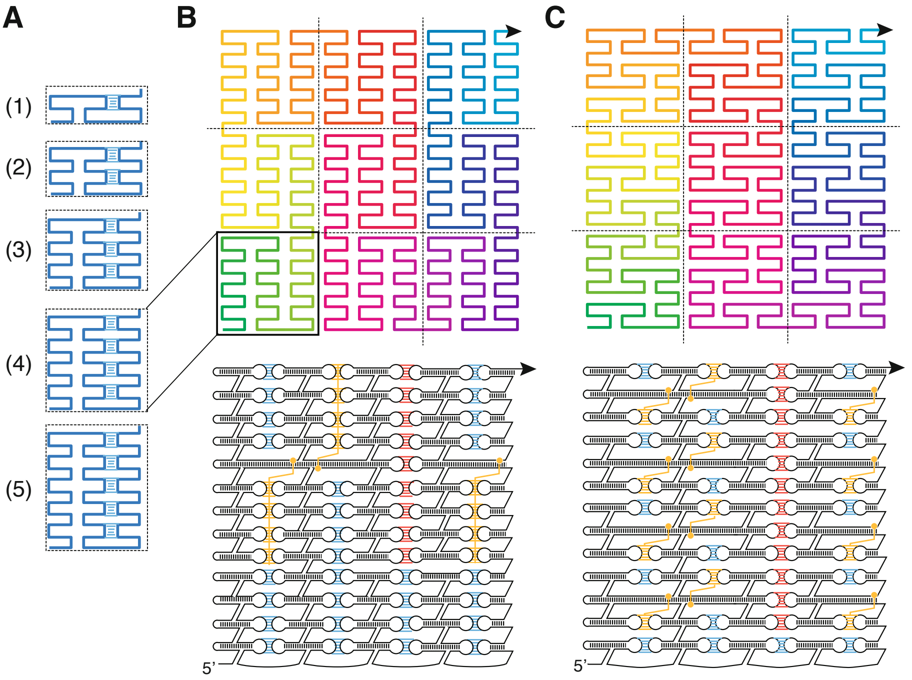


Fig. 7. Space-filling curves and their translation into RNA folding paths. (A) A set of simple repeating strand path patterns that can be tiled to form a Peano-like curve. **(B, top)** RNA Peano folding path based on pattern (4). The strand is colored to illustrate the strand directionality. Dotted lines indicate repetitions of the tile pattern. **(B, bottom)** The translation of the Peano curve into an RNA secondary structure. Colored lines indicate kissing loop interactions and their order of assembly, where blue forms first, orange second, and red last. Orange lines indicate kissing-loop interactions that have a requirement of slow folding, such that they must fold only after the 2D structure has formed up to the indicated orange dot. Red lines indicate a long seam of kissing-loop interactions. **(C)** A second RNA Peano-inspired folding path.

4.5 Rolling Circle Transcription of a Space-Filling RNA Structure

Rolling circle transcription (RCT) has recently been used to produced long transcripts of repeating RNA hairpin structures that further condense into sponge-like microspheres [20]. Likewise, rolling circle replication has been used to produce well-defined nanostructures out of DNA [21], although not nearly as large as has been demonstrated for RNA. Using the RNA origami architecture introduced in this paper it might be possible to produce well-ordered RNA structures by RCT. For example, an infinite 1D ribbon of RNA might be produced from only two repeat units (shown in light and dark blue, Fig. 8A). Such a ribbon shape would consist of alternating layers of kissing-loop and dovetail connections in a continuous 1D sheet.

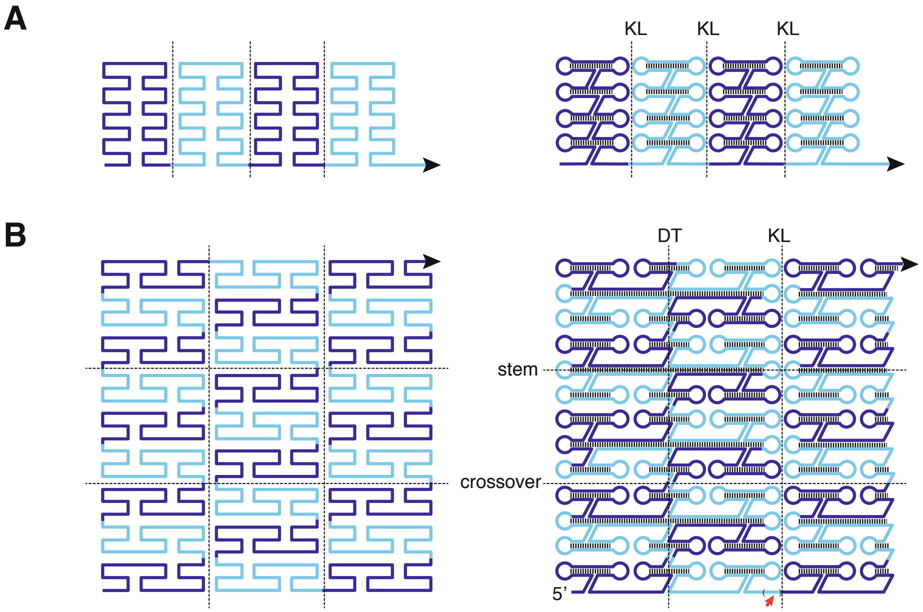


Fig. 8. Repeat units for 1D and 2D structure growth. (A) Curve and helix interpretation similar to the subpart of Fig. 7A with 4-helix repeat units shown in dark blue and cyan. Borders between repeat units are annotated as being composed of kissing-loop (KL) interactions. **(B)** Peano curve and helix interpretation similar to Fig. 7B with minimal repeating units shown in dark blue and cyan. The red arrow points to one position where a hairpin adopts a single-stranded form to fit the Peano path. Borders of the Peano curve are noted as being composed of either dovetail (DT), KL, stem or crossover interactions.

The space-filling curves of Peano inspire us to design repeating patterns of sequence and structure that could produce large and well-ordered space-filling structures by RCT. Taking the Peano curve as an example, we have made an attempt to define the sequence constraints for a tile that, when produced, will be able to adopt the infinite Peano-curve folding path while taking account of both 3D structure and topology. The basic repeating pattern is described in Fig. 8B, where just two RNA domains are required to produce the pattern. The structure is expected to raster back and forth, alternating between dovetail and kissing-loop seams. Several sequence constraints are imposed on the kissing-loop interactions and the dovetail seams, shown in more detail in Fig. 9A and 9B. Further illustrated in Fig. 9, some kissing loops form internally (denoted by different colors), while other have constraints because they are on the outside of the tile. One kissing loop and one dovetail are on the edge of the tile and are found to require base-pair palindromic symmetry in order to allow formation of the tile (marked in red).

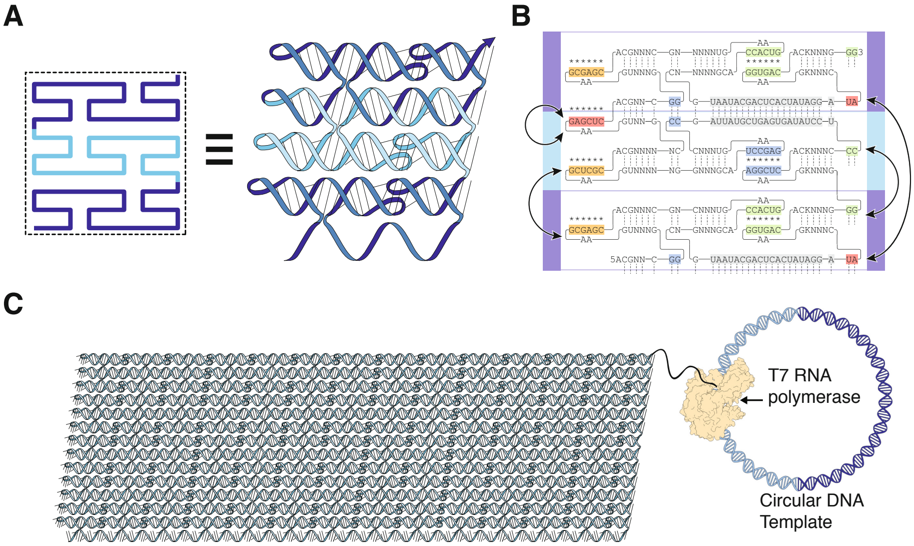

A
B
AA
ACGNNNC GN NNNNUG CCACUG ACKNNNG GG3
****** ******
GCGAGC GUNNNG CN NNNNGCA GGUGAC GKNNNC
AA AA
ACGNN C GG G UAAUACGACUCACUAUAGG A UA
******
GAGCUC GUNN G CC G AUUAUGCUGAGUGAUAUCC U
AA
AA
ACGNNNN NC CNNNUG UCCGAG ACKNNNC CC
****** ******
GCUCGC GUNNNN NG GNNNGCA AGGCUC GKNNNG
AA AA
AA
ACGNNNC GN CNNNUG CCACUG ACKNNNG GG
****** ******
GCGAGC GUNNNG CN GNNNGCA GGUGAC GKNNNC
AA AA
5ACGNN C GG G UAAUACGACUCACUAUAGG A UA
C
T7 RNA polymerase
Circular DNA Template

Fig. 9. Design of a circular RNA gene for producing a space-filling RNA structure. (A) The repeat units from Fig. 8B drawn as a ribbon diagram (similar to Fig. 2), with the repeating units shown in different colors. **(B)** Secondary structure with sequence constraints shown in different colors. Red indicate sequences that have to base pair with copies of itself and thus be base-pair palindromes. Sequences in orange, blue and green are orthogonal and pair according to color. Arrows indicate complementary domains. Grey sequence show the T7 promoter sequence and its complement, where GU mismatches have been introduced to disrupt the promoter signal in the opposite direction on the circular DNA template. **(C)** Ribbon diagram for an RNA structure produced from the circular DNA template gene by rolling circle transcription. T7 RNA polymerase and circular DNA template are shown to scale. Dark blue and cyan color of the circular DNA template indicate the two repeat domains.

We envision such a Peano tile may be encoded on a circular DNA template, which has an internal promoter sequence that is also encoded inside the RNA structure (positioned in the stem and marked in light grey). Fig. 9C shows the Peano-tile as it grows to larger structures. Even though it conceptually is tantalizing, this structure might not form for several reasons: First, the tile unit has many outward-facing interactions that might instead form internally in the tile, resulting in a misfold. Second, as mentioned before, the long stems may produce topological problems. Third, the assembly depends on the two-part tile unit to adopt an alternative conformation at key parts of the lattice. However, the correct interactions might still be preferred because, as the structure grows large, it may try to pack into its densest form, the desired assembly shape.

5 Discussion

We have introduced the A-form DX types and shown their extension to larger multi-crossover structures. The DX types have special properties concerning their bp spacing and internal symmetry. In the multi-crossover structures between several helices the DX types also have different properties concerning bending or flattening of the structures. As such, these motifs are important to consider when building extended architectures in 2D or 3D. This is especially important when designing RNA structures to be folded from a single strand, where geometry of crossovers and topological interactions have to be considered. We highlighted several paranemic assembly motifs well suited for making non-topologically linked interactions. We have demonstrated a methodology for folding path design using kissing-loop interactions and dovetail seams, which could easily be replaced with other tertiary motifs, of which there are plenty of good examples to choose from in the literature. To realize the design of complex structures produced by cotranscriptional folding, several theoretical aspects have to be further developed: 1) Sequence design taking account of pseudoknots, 2) The development of a kinetic model of folding, 3) Coarse grained modeling of the cotranscriptional folding process, and finally, 4) A theory for tile assembly when produced on a single strand.

References

1. Afonin, K.A., Cieply, D.J., Leontis, N.B.: Specific RNA self-assembly with minimal paranemic motifs. Journal of the American Chemical Society 130, 93–102 (2008)
2. Andersen, E.S.: Prediction and design of DNA and RNA structures. New Biotechnology 27, 184–193 (2010)
3. Arnott, S., Hukins, D.W., Dover, S.D., Fuller, W., Hodgson, A.R.: Structures of synthetic polynucleotides in the A-RNA and A'-RNA conformations: x-ray diffraction analyses of the molecular conformations of polyadenylic acid–polyuridylic acid and polyinosinic acid–polycytidylic acid. Journal of Molecular Biology 81, 107–122 (1973)
4. Costa, M., Michel, F.: Rules for RNA recognition of GNRA tetraloops deduced by in vitro selection: comparison with in vivo evolution. The EMBO Journal 16, 3289–3302 (1997)
5. Delebecque, C.J., Lindner, A.B., Silver, P.A., Aldaye, F.A.: Organization of intracellular reactions with rationally designed RNA assemblies. Science 333, 470–474 (2011)
6. Dickerson, R.E., Drew, H.R., Conner, B.N., Wing, R.M., Fratini, A.V., Kopka, M.L.: The anatomy of A-, B-, and Z-DNA. Science 216, 475–485 (1982)
7. Drew, H.R., Wing, R.M., Takano, T., Broka, C., Tanaka, S., Itakura, K., Dickerson, R.E.: Structure of a B-DNA dodecamer: conformation and dynamics. Proceedings of the National Academy of Sciences of the United States of America 78, 2179–2183 (1981)
8. Ennifar, E., Walter, P., Ehresmann, B., Ehresmann, C., Dumas, P.: Crystal structures of coaxially stacked kissing complexes of the HIV-1 RNA dimerization initiation site. Nat. Struct. Biol. 12, 1064–1068 (2001)

9. Fu, T.J., Seeman, N.C.: DNA double-crossover molecules. Biochemistry 32, 3211–3220 (1993)
10. Geary, C., Baudrey, S., Jaeger, L.: Comprehensive features of natural and in vitro selected GNRA tetraloop-binding receptors. Nucleic Acids Research 36, 1138–1152 (2008)
11. Geary, C., Chworos, A., Jaeger, L.: Promoting RNA helical stacking via A-minor junctions. Nucleic Acids Research 39, 1066–1080 (2011)
12. Geary, C.W., Rothemund, P.W.K., Andersen, E.S.: A single-stranded architecture for cotranscriptionally folded RNA tiles. Accepted in Science (2014)
13. Grabow, W.W., Jaeger, L.: RNA Self-Assembly and RNA Nanotechnology. Accounts of Chemical Research (2014)
14. Hao, C., Li, X., Tian, C., Jiang, W., Wang, G., Mao, C.: Construction of RNA nanocages by re-engineering the packaging RNA of Phi29 bacteriophage. Nature Communications 5, 3890 (2014)
15. Jaeger, L., Westhof, E., Leontis, N.B.: TectoRNA: modular assembly units for the construction of RNA nano-objects. Nucleic Acids Research 29, 455–463 (2001)
16. Jossinet, F., Ludwig, T.E., Westhof, E.: Assemble: an interactive graphical tool to analyze and build RNA architectures at the 2D and 3D levels. Bioinformatics 26, 2057–2059 (2010)
17. Ko, S.H., et al.: Synergistic self-assembly of RNA and DNA molecules. Nature Chemistry 2, 1050–1055 (2010)
18. Larson, M.H., et al.: A pause sequence enriched at translation start sites drives transcription dynamics in vivo. Science 344, 1042 (2014)
19. Lee, A.J., Crothers, D.M.: The solution structure of an RNA loop-loop complex: the ColE1 inverted loop sequence. Structure 6, 993–1007 (1998)
20. Lee, J.B., Hong, J., Bonner, D.K., Poon, Z., Hammond, P.T.: Self-assembled RNA interference microsponges for efficient siRNA delivery. Nature Materials 11, 316–322 (2012)
21. Lin, C., Wang, X., Liu, Y., Seeman, N.C., Yan, H.: Rolling circle enzymatic replication of a complex multi-crossover DNA nanostructure. Journal of the American Chemical Society 129, 14475–14481 (2007)
22. Rothemund, P.W.: Folding DNA to create nanoscale shapes and patterns. Nature 440, 297–302 (2006)
23. Severcan, I., Geary, C., Chworos, A., Voss, N., Jacovetty, E., Jaeger, L.: A polyhedron made of tRNAs. Nature Chemistry 2, 772–779 (2010)
24. Sherman, W.B., Seeman, N.C.: Design of minimally strained nucleic Acid nanotubes. Biophysical Journal 90, 4546–4557 (2006)
25. Shih, W.M., Quispe, J.D., Joyce, G.F.: A 1.7-kilobase single-stranded DNA that folds into a nanoscale octahedron. Nature 427, 618–621 (2004)
26. Wang, J.C.: Helical repeat of DNA in solution. Proceedings of the National Academy of Sciences of the United States of America 76, 200–203 (1979)
27. Woo, S., Rothemund, P.W.: Programmable molecular recognition based on the geometry of DNA nanostructures. Nature Chemistry 3, 620–627 (2011)
28. Yan, H., Zhang, X., Shen, Z., Seeman, N.C.: A robust DNA mechanical device controlled by hybridization topology. Nature 415, 62–65 (2002)
29. Yoffe, A.M., Prinsen, P., Gelbart, W.M., Ben-Shaul, A.: The ends of a large RNA molecule are necessarily close. Nucleic Acids Research 39, 292–299 (2011)

Fast Algorithmic Self-assembly of Simple Shapes Using Random Agitation

Ho-Lin Chen[1,⋆], David Doty[2,⋆⋆], Dhiraj Holden[2,⋆⋆⋆]
Chris Thachuk[2,†], Damien Woods[2,‡], and Chun-Tao Yang[1,§]

[1] Dept. of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
[2] California Institute of Technology, Pasadena, CA 91125, USA
holinchen@ntu.edu.tw, {ddoty,dholden,thachuk,woods}@caltech.edu,
havachoice@gmail.com

Abstract. We study the power of uncontrolled random molecular movement in a model of self-assembly called the nubots model. The nubots model is an asynchronous nondeterministic cellular automaton augmented with rigid-body movement rules (push/pull, deterministically and programmatically applied to specific monomers) and random agitations (nondeterministically applied to every monomer and direction with equal probability all of the time). Previous work on nubots showed how to build simple shapes such as lines and squares quickly—in expected time that is merely logarithmic of their size. These results crucially make use of the programmable rigid-body movement rule: the ability for a single monomer to push or pull large objects quickly, and only at a time and place of the programmers' choosing. However, in engineered molecular systems, molecular motion is largely uncontrolled and fundamentally random. This raises the question of whether similar results can be achieved in a more restrictive, and perhaps easier to justify, model where uncontrolled random movements, or agitations, are happening throughout the self-assembly process and are the only form of rigid-body movement. We show that this is indeed the case: we give a polylogarithmic expected time construction for squares using agitation, and a sublinear expected time construction to build a line. Such results are impossible in an agitation-free (and movement-free) setting and thus show the benefits of exploiting uncontrolled random movement.

1 Introduction

Every molecular structure that has been self-assembled in nature or in the lab was assembled in conditions (above absolute zero) where molecules are vibrating

[⋆] Supported by NSC grant number 101-2221-E-002-122-MY3.
[⋆⋆] Supported by National Science Foundation grants 0832824 & 1317694 (The Molecular Programming Project), CCF-1219274, CCF-1162589.
[⋆⋆⋆] Supported by NSF grant CCF-1219274.
[†] Supported by NSF grant CCF/HCC-1213127.
[‡] Supported by National Science Foundation grants 0832824 & 1317694 (The Molecular Programming Project), CCF-1219274, CCF-1162589.
[§] Supported by NSC grant number 101-2221-E-002-122-MY3.

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 20–36, 2014.

relative to each other, randomly bumping into each other via Brownian motion, and often experiencing rapid uncontrolled fluid flows. It makes sense then to study a model of self-assembly that includes, and indeed allows us to exploit and program, such phenomena. It is a primary goal of this paper to show the power of self-assembly under such conditions.

In the theory of molecular-scale self-assembly, millions of simple interacting components are designed to autonomously stick together to build complicated shapes and patterns. Many models of self-assembly are cellular automata-like crystal growth models, such as the abstract tile assembly model [9]. Indeed this and other such models have given rise to a rich theory of self-assembly [5,8,10]. In biological systems we frequently see much more sophisticated growth processes, where self-assembly is combined with active molecular motors that have the ability to push and pull large structures around. For example, during the gastrulation phase of the embryonic development of the model organism *Drosophila melanogaster* (a fly) large-scale (100s of micrometers) rearrangements of the embryo are effected by thousands of (nanoscale) molecular motors working together to rapidly push and pull the embryo into a final desired shape [4,7]. We wish to understand, at a high level of abstraction, the ultimate computational capabilities and limitations of such molecular scale rearrangement and growth.

The nubots model of self-assembly, put forward in [11], is an asynchronous nondeterministic cellular automaton augmented with non-local rigid-body movement. Unit-sized monomers are placed on a 2D hexagonal grid. Monomers can undergo state changes, appear, and disappear, using local cellular-automata style rules. However, there is also a non-local aspect to the model, a kind of rigid body movement that comes in two forms: movement rules and random agitations. A movement rule r, consisting of a pair of monomer states and two unit vectors, is a programatic way to specific unit-distance translation of a set of monomers in one step: if two adjacent monomers on the grid have states A and B and are in a prescribed orientation, then we may try to apply r so that one of A or B *moves* unit distance in a prescribed direction relative to the other. The rule r is applied in a rigid-body fashion: if A is to move right, it pushes anything immediately to its right and pulls any monomers that are bound to its left (roughly speaking), which in turn push and pull other monomers. The rule may not be applicable if it is blocked (i.e. if movement of A would force B to also move), which is analogous to the fact that an arm can not push its own shoulder. The other form of movement in the model is called *agitation*: at every point in time, every monomer on the grid may move unit distance in any of the six directions, at unit rate for each (monomer, direction) pair. An agitating monomer will push or pull any monomers that it is adjacent to, in a way that preserves rigid-body structure. Unlike movement, agitations are never blocked. Rules are applied asynchronously and in parallel in the model. Taking its time model from stochastic chemical kinetics, a nubots system evolves as a continuous time Markov process.

In summary, there are two kinds of movement in the model: (a) a *movement rule* is applied only to a pair of monomers with the prescribed states and orientation, and then causes the movement of one of these monomers (and other pushed/pulled monomers) but not the other, whereas (b) *agitations* are always applicable at every time instant, in every direction and to every monomer throughout the grid.

In previous work, the movement rule was exploited to show that nubots is very efficient in terms of its computational ability to quickly build complicated shapes and patterns. Agitation was treated as something to be robust against (i.e. the constructions in [11,2] work both with and without agitation), which seems like a natural requirement when building structures in a molecular-scale environment. However, it was left open as to whether the kind of results achieved with movement could be achieved without movement, but by exploiting agitation [2]. In other words, it was left open as to whether augmenting a cellular automaton with an uncontrolled form of random rigid-body movement would facilitate functionality that is impossible without it. Here we show this is the case.

Agitation, and the movement rule, are defined in such a way that larger objects move faster, and this is justified by imagining that we are self-assembling rigid-body objects in a nanoscale environment where there are uncontrolled and turbulent fluid flows in all directions interacting with each monomer at unit rate per monomer. It remains as an interesting open research direction to look at the nubots model but with a slower rate model for agitation and movement, specifically where we hold on to the notion of rigid body movement and/or agitation but where bigger things move slower, as seen in Brownian motion for example. Independent of the choice of rate model, one of our main motivations here is to understand what can be done with *asynchronous, distributed and parallel* self-assembly with *rigid body motion*: the fact that our systems work in a parallel fashion is actually more important to us than the fact they are fast. It is precisely this engineering of distributed asynchronous molecular systems that interests us.

The nubots model is related to, but distinct from, a number of other self-assembly and robotics models as described in [11]. Besides the fact that biological systems make extensive use of molecular-scale movements and rearrangements, in recent years we have seen the design and fabrication of a number of molecular-scale DNA motors [1] and active self-assembly systems which also serve to motivate our work, details of which can be found in previous papers on nubots [11,2].

1.1 Results and Future Work

Agitation nubots denotes the nubots model without the movement rule and with agitation (see Section 2 for formal definitions). The first of our two main results shows that agitation can be exploited to build a large object exponentially quickly:

Theorem 1. *There is a set of nubots rules $\mathcal{N}_{\text{square}}$, such that for all $n \in \mathbb{N}$, starting from a line of $\lfloor \log_2 n \rfloor + 1$ monomers, each in state* 0 *or* 1*, $\mathcal{N}_{\text{square}}$ in*

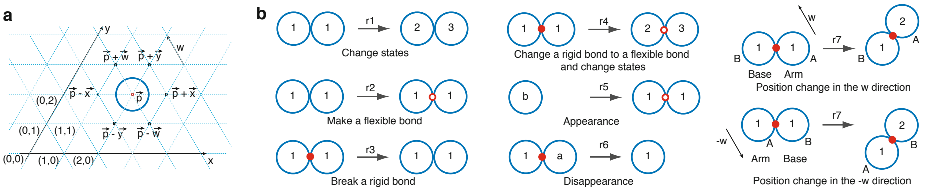


Fig. 1. Overview of nubots model. (a) A nubot configuration showing a single nubot monomer on the triangular grid. (b) Examples of nubot monomer rules. Rules r1-r6 are local cellular automaton-like rules, whereas r7 effects a non-local movement. A flexible bond is depicted as an empty red circle and a rigid bond is depicted as a solid red disk. Rules and bonds are described more formally in Section 2.

the agitation nubots model assembles an $n \times n$ *square in* $O(\log^2 n)$ *expected time,* $n \times n$ *space and* $O(1)$ *monomer states.*

The proof is in Section 4. Our second main result shows that we can achieve sublinear expected time growth of a length n line in only $O(n)$ space:

Theorem 2. *There is a set of nubots rules* $\mathcal{N}_{\mathrm{line}}$*, such that for any* $\epsilon > 0$*, for sufficiently large* $n \in \mathbb{N}$*, starting from a line of* $\lfloor \log_2 n \rfloor + 1$ *monomers, each in state* 0 *or* 1*,* $\mathcal{N}_{\mathrm{line}}$ *in the agitation nubots model assembles an* $n \times 1$ *line in* $O(n^{1/3+\epsilon})$ *expected time,* $n \times 5$ *space and* $O(1)$ *monomer states.*

The proof is in Section 5. Lines and squares are examples of fundamental components for the self-assembly of arbitrary computable shapes and patterns in nubots [11,2,3] and other self-assembly models [5,8].

Our work here suggests that random agitations applied in an uncontrolled fashion throughout the grid are a powerful resource. However, are random agitations as powerful as the programable and more deterministic movement rule used in previous work on nubots [11,2]? In other words can agitation *simulate* movement? More formally, is it the case that for each nubots program $\mathcal{N}$, there is an agitation nubots program $\mathcal{A}_{\mathcal{N}}$, that acts just like $\mathcal{N}$ but with some $m \times m$ scale-up in space, and a k factor slowdown in time, where m and k are (constants) independent of $\mathcal{N}$ and its input? This question is inspired by the use of simulations in tile assembly as a method to classify and separate the power of self-assembly systems, for more details see [6,10]. It would also be interesting to know whether the full nubots model, and indeed the agitation nubots model, are intrinsically universal [6,10]. That is, is there a single set of nubots rules that simulate any nubots system? Is there a single set of agitation nubots rules that simulate any agitation nubots system? Here the scale factor m would be a function of the simulated system $\mathcal{N}$. As noted in the introduction, it remains as an interesting open research direction to look at the nubots model but with a slower rate model for agitation and movement, as seen in Brownian motion, for example.

2 The Nubots Model

In this section we formally define the nubots model. Figure 1 gives an overview of the model and rules. Figure 3 shows a simple example construction using only local rules, while Figure 2 gives two examples of agitation.

The model uses a two-dimensional triangular grid with a coordinate system using axes x and y as shown in Figure 1(a). A third axis, w, is defined as running through the origin and $\overrightarrow{w} = -\overrightarrow{x} + \overrightarrow{y} = (-1, 1)$, but we use only the x and y coordinates to define position. The *axial directions* $\mathcal{D} = \{\pm\overrightarrow{x}, \pm\overrightarrow{y}, \pm\overrightarrow{w}\}$ are the unit vectors along axes x, y, w. A pair $\overrightarrow{p} \in \mathbb{Z}^2$ is called a *grid point* and has the set of six *neighbors* $\{\overrightarrow{p} + \overrightarrow{u} \mid \overrightarrow{u} \in \mathcal{D}\}$. Let S be a finite set of monomer states. A nubot *monomer* is a pair $X = (s_i, p(X))$ where $s_i \in S$ is a state and $p(X) \in \mathbb{Z}^2$ is a grid point. Two monomers on neighboring grid points are either connected by a *flexible* or *rigid* bond, or else have no bond (called a *null* bond). Bonds are described in more detail below. A *configuration* C is a finite set of monomers along with all of the bonds between them (unless otherwise stated a configuration consists of *all* of the monomers on the grid and their bonds).

One configuration *transitions* to another either via the application of a *rule* that acts on one or two monomers, or by an *agitation*. For a rule $r = (s1, s2, b, \overrightarrow{u}) \to (s1', s2', b', \overrightarrow{u}')$, the left and right sides of the arrow respectively represent the contents of the two monomer positions before and after the application of r. Specifically, $s1, s2, s1', s2' \in S \cup \{\mathsf{empty}\}$ are monomer states where empty denotes lack of a monomer, $b, b' \in \{\mathsf{flexible}, \mathsf{rigid}, \mathsf{null}\}$ are bond types, and $\overrightarrow{u}, \overrightarrow{u}' \in \mathcal{D}$ are unit vectors. b is a bond type between monomers with state $s1$ and $s2$, and $\overrightarrow{u} \in \mathcal{D}$ is the relative position of a monomer with state $s2$ to a monomer with state $s1$ (likewise for $b', s1', s2', \overrightarrow{u}'$). At most one of $s1, s2$ is empty (we disallow spontaneous generation of monomers from empty space). If $\mathsf{empty} \in \{s1, s2\}$ then $b = \mathsf{null}$, likewise if $\mathsf{empty} \in \{s1', s2'\}$ then $b' = \mathsf{null}$.

A rule either does not or does involve movement (translation). First, in the case of no movement we have $\overrightarrow{u} = \overrightarrow{u}'$. Thus we have a rule of the form $r = (s1, s2, b, \overrightarrow{u}) \to (s1', s2', b', \overrightarrow{u})$, where the monomer pair may *change state* ($s1 \neq s1'$ and/or $s2 \neq s2'$) and/or *change bond* ($b \neq b'$), examples are shown in Figure 1(b). If $s_i \in \{s1, s2\}$ is empty and s_i' is not, then the rule is said to induce the *appearance* of a new monomer at the empty location. If one or both monomer states go from non-empty to empty, the rule induces the *disappearance* of one or both monomers. Second, in the case of a movement rule, $\overrightarrow{u} \neq \overrightarrow{u}'$ and the rule has a specific form defined in [11,2]. Movement rules are not used in the *agitation nubots* model studied in this paper, and so their definition may be ignored by the reader. A rule is only applicable in the orientation specified by $\overrightarrow{u}$.

To define agitation we introduce some notions. Let $\overrightarrow{v} \in \mathcal{D}$ be a unit vector. The $\overrightarrow{v}$-boundary of a set of monomers S is defined to be the set of grid points outside of S that are unit distance in the $\overrightarrow{v}$ direction from monomers in S.

Definition 3 (Agitation set). *Let C be a configuration containing monomer A, and let $\overrightarrow{v} \in \mathcal{D}$ be a unit vector. The* agitation set $\mathcal{A}(C, A, \overrightarrow{v})$ *is defined to be the smallest monomer set in C containing A that can be translated by $\overrightarrow{v}$ such that:*

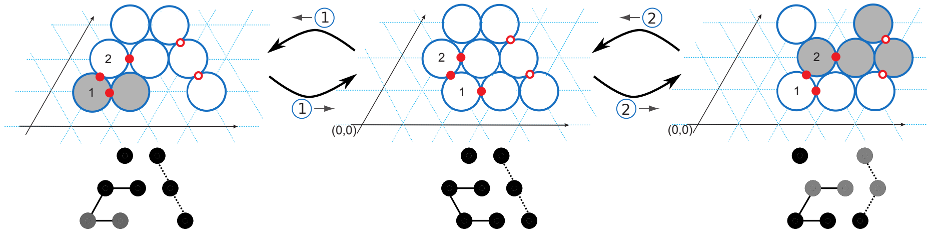


Fig. 2. Top: Example agitations. Starting from the centre configuration, there are 48 possible agitations (8 monomers, 6 directions each), any one of which is chosen with equal probability 1/48. The right configuration results from the agitation of the monomer at position $(1, 2)$ in the direction $\rightarrow$, starting from the centre configuration. The left configuration results from the agitation of the monomer at position $(2, 1)$ in the direction $\leftarrow$, starting from the centre configuration. The shaded monomers are the *agitation set*—the set of monomers that are moved by the agitation—when beginning from the centre configuration. Bottom: simplified ball-and-stick representation of the monomers and their bonds, which is used in a number of other figures.

(a) monomer pairs in C that are joined by rigid bonds do not change their relative position to each other, (b) monomer pairs in C that are joined by flexible bonds stay within each other's neighborhood, and (c) the $\overrightarrow{v}$-boundary of $\mathcal{A}(C, A, \overrightarrow{v})$ contains no monomers.

We now define agitation. An *agitation* step acts on an entire configuration C as follows. A monomer A and unit vector $\overrightarrow{v}$ are selected uniformly at random from the configuration of monomers C and the set of six unit vectors $\mathcal{D}$ respectively. Then, the agitation set $\mathcal{A}(C, A, \overrightarrow{v})$ of monomers (Definition 3) moves by vector $\overrightarrow{v}$.

Figure 2 gives two examples of agitation. Some remarks on agitation: It can be seen that for any non-empty configuration the agitation set is always non-empty. During agitation, the only change in the system configuration is in the positions of the constituent monomers in the agitation set, and all the monomer states and bond types remain unchanged. We let *agitation nubots* denote the nubots model without the movement rule. Agitation is intended to model movement that is not a direct consequence of a rule application, but rather results from diffusion, Brownian motion, turbulent flow or other uncontrolled inputs of energy.

An *assembly system* $T = (C_0, \mathcal{N})$ is a pair where C_0 is the initial configuration, and $\mathcal{N}$ is the set of rules. If configuration C_i transitions to C_j by some rule $r \in \mathcal{N}$, or by an agitation step, we write $C_i \vdash_{\mathcal{N}} C_j$. A *trajectory* is a finite sequence of configurations $C_1, C_2, \ldots, C_\ell$ where $C_i \vdash_{\mathcal{N}} C_{i+1}$ and $1 \leq i \leq \ell - 1$. An assembly system is said to *assemble* a shape or pattern if, starting from some initial configuration C_0, every trajectory evolves to the desired shape or pattern. An assembly system evolves as a continuous time Markov process. The rate for each rule application, and for each agitation step, is 1. If there are k applicable transitions for a configuration C_i (i.e. k is the sum of the number of rule and agitation steps that can be applied to all monomers), then the probability of any given transition being applied is $1/k$, and the time until the next transition is

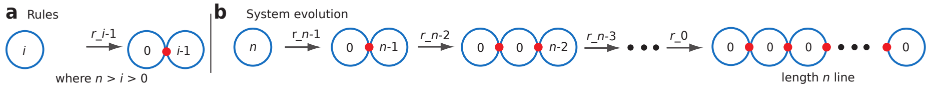


Fig. 3. A nubots system that slowly grows a length n line in $O(n)$ time, n monomer states, and using space $n \times 1$. (a) Rule set: $\mathcal{R}_n^{\text{slow line}} = \{r_i \mid r_i = (i, \mathsf{empty}, \mathsf{null}, \boldsymbol{x}) \rightarrow (0, i-1, \mathsf{rigid}, \boldsymbol{x}),\ \text{where } n > i > 0\}$. (b) Starting from an initial configuration with a single monomer in state n, the system generates a length n line. Taken from [11].

applied is an exponential random variable with rate k (i.e. the expected time is $1/k$). The probability of a trajectory is then the product of the probabilities of each of the transitions along the trajectory, and the expected time of a trajectory is the sum of the expected times of each transition in the trajectory. Thus, $\sum_{t \in \mathcal{T}} \Pr[t] \text{time}(t)$ is the expected time for the system to evolve from configuration C_i to configuration C_j, where $\mathcal{T}$ is the set of all trajectories from C_i to any configuration isomorphic (up to translation) to C_j, that do not pass through any other configuration isomorphic to C_j, and time(t) is the expected time for trajectory t.

The complexity measure *number of monomers* is the maximum number of monomers that appears in any configuration. The *number of states* is the total number of distinct monomer states that appear in the rule set. *Space* is the maximum, over the set of all reachable configurations, of the minimum-sized $l \times w$ rectangle (on the hex grid) that, up to translation, contains all monomers in the configuration.

2.1 Example: A Simple, But Slow, Method to Build a Line

Figure 3, from [11], shows a simple method to build a length n line in expected time n, using $O(n)$ monomer states. Here, the program is acting as an asynchronous cellular automata and is *not* exploiting the ability of a large set of monomers to quickly move via agitation. Our results show that we can do much better than this very slow and expensive (many states) method to grow a line.

3 Synchronization via Agitation

In this section we describe a fast method to synchronize the states of a line of monomers using agitation. Specifically, the *synchronization* problem is: given a length-m line of monomers that are in a variety of states but that all eventually reach some target state s, then after all m monomers have reached state s, communicate this fact to all m monomers in $O(\log m)$ expected time.

Lemma 4 (Synchronization). *A line of monomers of length $m \in \mathbb{N}$ can be synchronized (all monomers put into the same state) in $O(\log m)$ expected time, with $O(1)$ states, and in $m \times O(1)$ space.*

The proof is described in Figure 4 and its caption. The figure gives a synchronization routine that is used throughout our constructions. This is a modification of

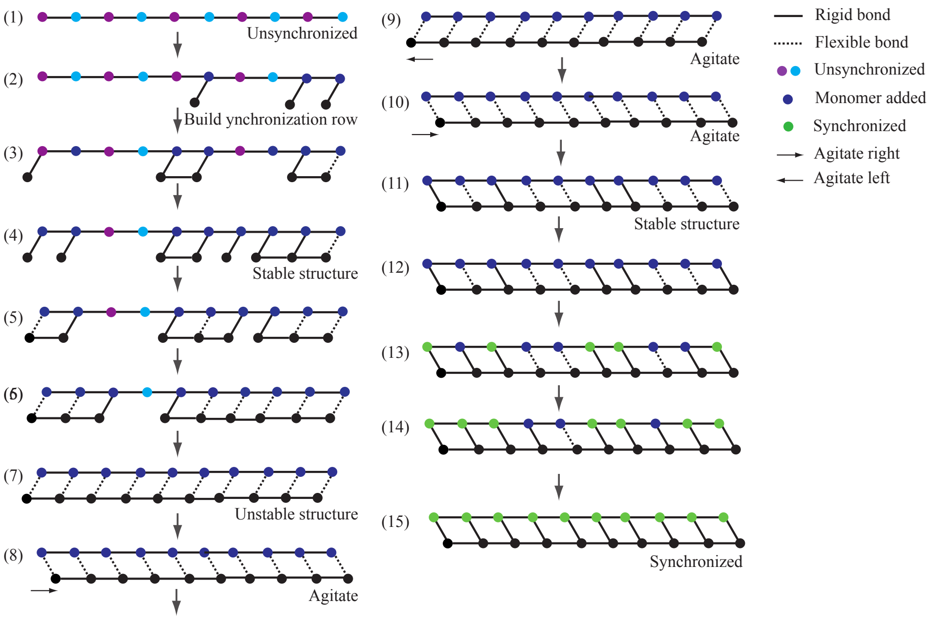


Fig. 4. Synchronization via agitation: a nubots construction to synchronize (or send a signal, or reach consensus) between n monomers in $O(\log n)$ expected time. Steps (1)–(6): build a row of monomers called the synchronization row. Rigid bonds are converted to flexible bonds in such a way that agitations do not change the relative position of monomers. A structure with this property is said to be *stable*. Specifically, monomers are added using rigid vertical bonds; new monomers join to left-right neighbours using rigid horizontal bonds; when a monomer is bound horizontally to both neighbours it makes its *vertical* bond flexible; monomers on the extreme left and right of the synchronization row are treated differently—their vertical bonds become flexible after joining any horizontal neighbour. This enforces that the entire structure is stable up until the final horizontal bond is added, and then the structure becomes unstable in such a way that the synchronization row can agitate left-right relative to the backbone row. Steps (7)–(10), the structure is not stable, and the synchronization row is free to agitate left and right relative to the backbone row. While agitating, the synchronization row spends half the time to the left, and half to the right, of the backbone row. However, whenever the synchronization row is to the right a rigid bond may form between any synchronization row monomer and the backbone monomer directly above, hence the first such bond forms in expected time $1/m$, where m is the length of the backbone. Then all bonds become rigid in $O(\log m)$ expected time, during which time (12)–(15) the backbone monomers change their state to the final *synchronized* state.

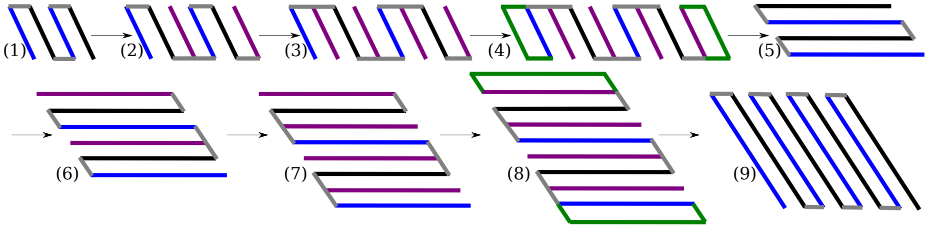


Fig. 5. An overview of the square doubling algorithm that grows an $m \times m$ zig-zag "comb" to a $2m \times 2m$ comb. (1) An initial $m \times m$ comb with vertical teeth, is (2) "half-doubled" to give a $\lfloor 1.5m \rfloor \times m$ comb, which is (3) again half-doubled to give a $2m \times m$ comb. (4)–(5) The internal bond structure is reconfigured to give a comb with horizontal teeth. (6)–(7) this comb is vertically doubled in size and then (8)–(9) reconfigured to give a $2m \times 2m$ comb with vertical teeth. The green lines indicate temporary synchronization rows that are used when reorientating the teeth of the comb.

the synchronization routine in [11], made to work with agitation instead of the movement rule.

4 Building Squares via Agitation

This section contains the proof of our first main result, Theorem 1.

Proof (Theorem 1). **Overview of Construction.** Figure 5 gives an overview of our construction. A binary string that represents $n \in \mathbb{N}$ in the standard way is encoded as a string x, of length $\ell = \lfloor \log_2 n \rfloor + 1$, of adjacent rigidly bound binary nubot monomers (each in state 0 or 1) placed somewhere on the hexagonal grid.

The leftmost of these monomers begins an iterated square-doubling process, that happens exactly ℓ times. Each iteration of this square-doubling process: reads the current most significant bit x_i of x, where $0 \leq i \leq \ell$, stores it in the state of a monomer in the top-left of the square and then deletes x_i. Then, if $x_i = 0$ it takes an $m \times m$ comb structure and doubles its size to give a $2m \times 2m$ comb structure, or if $x_i = 1$ it gives a $(2m+1) \times (2m+1)$ structure. We will prove that each square-doubling step takes $O(\log m)$ time. There are ℓ rounds of square-doubling, i.e. the number of input monomers ℓ act as a counter to control the number of iterations, and since $m \leq n$ throughout, the process completes in the claimed expected time of $O(\log^2 n)$. The main part of the construction, detailed below, lies in the details of how each doubling step works and an expected time analysis, and constitutes the remainder of the proof.

Doubling Construction. A single square-doubling consists of four phases: two horizontal "half-doublings" and two vertical half-doublings. Figure 5 gives an overview. Figure 6 gives the details of how we do the first of two horizontal half-doublings; more precisely, the figure shows how to go from an $m \times m$ structure to a structure of size $\lfloor 1.5m \rfloor \times m$. Assume we are at a configuration with m vertical comb teeth (Figure 6(1)) each of height m (plus some additional monomers).

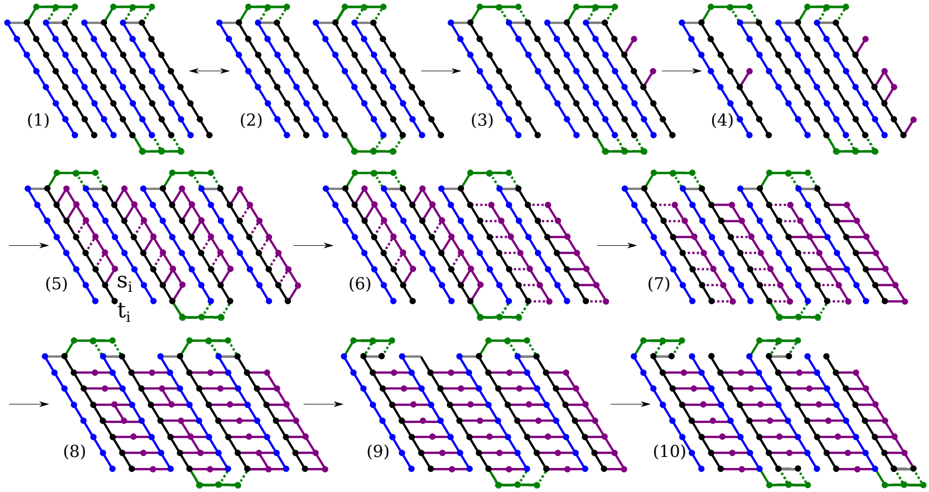


Fig. 6. The $m \times m$ to $\lfloor 1.5m \rfloor \times m$ horizontal half-doubling algorithm, for $m = 8$. This shows the details for step (1) to (2) of Figure 5. Monomer states are denoted using colours (bonds are also coloured for readability). Rigid bonds are solid, flexile bonds are dotted. See main text for details.

Teeth are numbered from the left $t_1, t_2, \ldots, t_m$. Each tooth monomer undergoes agitation. It can be seen in Figure 6(1)–(4), from the bond structure, that the only agitations that change the relative position of monomers are left or right agitations which move the green flexible bonds (depicted as dashed lines)—all other agitations move the entire structure without changing the relative positions of any monomers. Furthermore, left-right monomer agitations can create gaps between teeth t_i and t_{1+1} for even i only—for odd i, teeth t_i and t_{1+1} are rigidly bound. An example of a gap opening between tooth t_4 and tooth t_5 is shown in Figure 6(2). If a gap appears between teeth t_i and t_{1+1} then each of the m monomers in tooth t_i tries to attach a new purple monomer to its right (with a rigid bond, and each at rate 1), so attachment for any monomer to tooth i happens at rate m. (Note that the gap is closing and opening at some rate also—details in the time analysis.) After the first such purple monomer appears, the gap g_i, to the right of tooth t_i, is said to be "initially filled". For example, in Figure 6(4), gap g_2 is initially filled.

When gaps appear between teeth monomers, and then become initially filled, additional monomers are attached, asynchronously and in parallel. Monomers attaching to tooth t_i initially attach by rigid bonds as shown in Figure 6(4). As new monomers attach to t_i, they then attempt to bind to each other vertically, and after such a binding event they undergo a sequence of bond changes—see Figure 6(4)-(9). Specifically, let $s_{i,j}$ be the j^{th} monomer on the newly-forming "synchronization row" s_i adjacent to t_i. When the neighbors $s_{i,j-1}, s_{i,j+1}$ of monomer $s_{i,j}$ appear, then $s_{i,j}$ forms rigid bonds with them (at rate 1). After this, $s_{i,j}$ changes its rigid bonds to $t_{i,j}$ to flexible. The top and bottom monomers $s_{i,1}$, $s_{i,m}$ are special cases: their bonds to $t_{i,1}$, $t_{i,m}$ become flexible after they have joined to their (single) neighbors $s_{i,2}$, $s_{i,m-1}$. Changing bonds in this order guarantees that only *after all monomers* of s_i have attached, and not before, the synchronization row s_i is free to agitate up and down relative to the tooth t_i (this is the same technique for building a synchronization row as described in Section 3). The new vertical synchronization row s_i is then free to agitate up and down relative to its left-adjacent tooth t_i. When $s_{i,j}$ is "down" relative to $t_{i,j}$ the horizontal bonds between $s_{i,j}$ and $t_{i,j}$ become rigid, at rate 1 per bond (Figure 6(6)–(7)). When the vertical synchronization of s_i is done, a message is sent from the top monomer $t_{i,m}$ of t_i (after its bond to $s_{i,m}$ becomes rigid) to the adjacent monomer at the top of the comb. This results in the formation of a horizontal synchronization row at the top of the structure. Using a similar technique, a horizontal synchronization row grows at the bottom of the structure. After all $2\lfloor 0.5m \rfloor$ such messages have arrived, and not before, the horizontal synchronization rows at the top and bottom of the (now) $\lfloor 1.5m \rfloor \times m$ comb change the last of their rigid (vertical) bonds to flexible and those synchronization rows are free to agitate left/right and then lock into position, signaling to all monomers along their backbone that the first of the four half-doublings of the comb has finished.

The system prepares for the next horizontal half-doubling which will grow the $\lfloor 1.5m \rfloor \times m$ comb to be an $2m \times m$ comb. The bonds at the top and bottom horizontal synchronization rows reconfigure themselves (preserving connectivity of the overall structure—see the description of reconfiguration below) in such a way as to build the gadgets needed for the next half-doubling. (Specifically, we want to now double teeth t_i for odd $i \leq m$.) The construction proceeds similarly to the first half-doubling, except for the following change. After tooth synchronization row s_1 has synchronized, tooth t_1 grows a vertical synchronization row to its left, and after s_m has synchronized, tooth t_m grows a vertical synchronization row to its right (Figure 5(4)). These two synchronization rows are used to set-up the bond structure for the next stage of the construction (where we will reconfigure the entire comb so that the teeth are horizontal).

This covers the case of the input bit being 0. Otherwise, if the input bit is 1, adding an extra tooth can be done using the single vertical synchronization row on the right—it reconfigures itself to have the bond structure of a tooth and then grows a new vertical synchronization row.

Reconfiguration Construction. Next we describe how the comb with vertical teeth is reconfigured to have horizontal teeth, as in Figure 5(4)–(5). After synchronization row s_i has synchronized, each monomer $s_{i,j}$ in s_i already has a rigid horizontal bond to monomer $t_{i,j}$. After both s_i and s_{i+1} have synchronized, for all j, monomers $s_{i,j}$ and $t_{i+1,j}$ bond using a horizontal rigid bond (at rate 1) for each pair $(s_{i,j}, t_{i+1,j})$. Monomers t_i and s_i then delete their vertical rigid bonds in such a way that preserves the overall connectivity of the structure. (For these bond reconfigurations we are simply using local—asynchronous cellular automaton style—rules that preserves connectivity. This trick has been used in previous nubots constructions in [11,2].) This leads to a bond structure similar to that in Figure 6(10) both with roughly twice the number of horizontal purple bonds: i.e. for each j, $1 \leq j \leq m$, there is now a horizontal straight line of purple bonds from the jth monomer on the leftmost vertical line to the jth monomer on the rightmost vertical line. While this reconfiguration is taking place, the leftmost and rightmost vertical synchronization rows synchronize and delete themselves, leaving appropriate gadgets to connect the horizontal teeth: this signals the beginning of the next two half-doubling steps.

Expected Time, Space and States Analysis. Lemma 5 states that the expected time to perform a half-doubling is $O(\log m)$ for an $m \times m$ comb, and since $n \leq m$, the slowest half-doubling takes expected time $O(\log n)$. Each doubling involves 2 horizontal half-doubling phases, and 2 vertical half-doubling phases, and the 4 phases are separated by discrete synchronization events. Reconfiguration involves $O(n^2)$ bond and state change events, that take place independently and in parallel ($O(\log n)$ expected time) as well as a constant number of synchronizations that each take $O(\log n)$ expected time. Hence for $4(\lfloor \log_2 n \rfloor + 1)$ such half-doublings, plus $\lfloor \log_2 n \rfloor + 1$ reconfigurations, we get an overall expected time of $O(\log^2 n)$.

We've sketched how to make an $n \times n$ structure in $(n+2) \times (n+2)$ space. To make the construction work in $n \times n$ space, we first subtract 2 from the input, and build an $(n-2) \times (n-2)$ structure, and then at the final step have the leftmost and rightmost horizontal, and topmost and bottommost vertical, synchronization rows become rigid and be the border of the final $n \times n$ structure. A final monomer is added on the top left corner and we are done. By stepping through the construction it can be seen that $O(1)$ monomer states are sufficient. □

Intuitively, the following lemma holds because the long (length m) teeth allow for rapid, $O(1)$ time per tooth, and parallel insertion of monomers to expand the width of the comb. This intuition is complicated by the fact that teeth agitating open and closed may temporarily block other teeth inserting a new monomer. However, after an insertion actually happens further growth occurs independently and in parallel, taking logarithmic expected time overall.

Lemma 5. *A comb with m teeth where each tooth is of height m, can be horizontally half-doubled to length $\lfloor 1.5m \rfloor$ in expected time $O(\log m)$ using agitation nubots.*

Proof. Consider tooth i, where $1 \leq i \leq m$ for i even. A tooth can be `open`, `closed` or `initially filled` (one new monomer inserted). Although the remaining structure can affect the transition probabilities relevant to tooth i, in any state, the rate at which the tooth transitions from `closed` to `open` is at least m, the rate that it transitions from `open` to `closed` is at most m^2, and the rate at which it transitions from `open` to `initially filled` is exactly m. We define a new Markov process, with states `open`, `closed`, and `initially filled` and the transition probabilities just described, which is easier to analyze. Clearly, the random variable representing the time for this process to transition from `closed` to `initially filled` upper bounds the random variable representing the time for the real nubots process to do the same for a single tooth. We now show that this new random variable has expected value $O(1)$.

Let T_{cf} be the random variable representing the time to go from `closed` to `initially filled`. Let T_{co} be the random variable representing the time to go from `closed` to `open`. Let T_{oc} be the random variable representing the time to go from `open` to `closed`, conditioned on that transition happening, and define T_{of} similarly for going from `open` to `initially filled`. Note that $\mathrm{E}[T_{\text{co}}] \leq \frac{1}{m}$, $\mathrm{E}[T_{\text{oc}}] \geq \frac{1}{m^2}$, and $\mathrm{E}[T_{\text{of}}] = \frac{1}{m}$. Let E_i represent the event that the process revisits state `closed` exactly i times after being in state `open` (and before reaching state `initially filled`). Let T_i be the random variable representing the time to take exactly i cycles between the states `open` and `closed`. Let C be the random variable representing the number of cycles taken between the states `open` and `closed` before transitioning to state `initially filled`. $\mathrm{E}[C] = m$ since the process goes from `open` to `initially filled` with probability $\frac{1}{m}$. Then

$$\begin{aligned}
\mathrm{E}[T_{\text{cf}}] &= \mathrm{E}[T_{\text{co}}] + \mathrm{E}[T_{\text{of}}] + \sum_{i=0}^{\infty} \Pr[E_i] \cdot \mathrm{E}[T_i] \\
&\leq \frac{2}{m} + \sum_{i=0}^{\infty} \Pr[E_i] \cdot i \cdot \left(\frac{1}{m^2} + \frac{1}{m}\right) \\
&= \frac{2}{m} + \left(\frac{1}{m^2} + \frac{1}{m}\right) \sum_{i=0}^{\infty} \Pr[E_i] \cdot i \\
&= \frac{2}{m} + \left(\frac{1}{m^2} + \frac{1}{m}\right) \mathrm{E}[C] \\
&= \frac{2}{m} + \left(\frac{1}{m^2} + \frac{1}{m}\right) m \leq 2.
\end{aligned}$$

By Markov's inequality, the probability is at most $\frac{1}{2}$ that it will take more than time 4 to reach from `closed` to `initially filled`. Because of the memoryless property of the Markov process, conditioned on the fact that time t has elapsed without reaching state `initially filled`, the probability is at most $\frac{1}{2}$ that it will take more than $t + 4$ time to reach state `initially filled`. Hence for any $t > 0$, the probability that it will take more than than $4t$ time to reach from state `closed` to `initially filled` is at most 2^{-t}.

Since this tail probability decreases exponentially, it follows that for $m/2$ teeth, the expected time for all of them to reach state `initially filled` is $O(\log m)$. □

5 Building Lines via Agitation

In this section we build a line in sublinear time while using merely $O(n)$ space. We prove this, our second main theorem (Theorem 2), by giving a line construction that works in merely $n \times 5 = O(n)$ space while achieving sublinear expected time $O(n^{\epsilon+1/3})$, and $O(1)$ monomer states.

Proof (Theorem 2). **Overview of Construction.** The binary expansion of $n \in \mathbb{N}$ is encoded as a horizontal line, denoted x, of $\ell = \lfloor \log_2 n \rfloor + 1$ adjacent binary

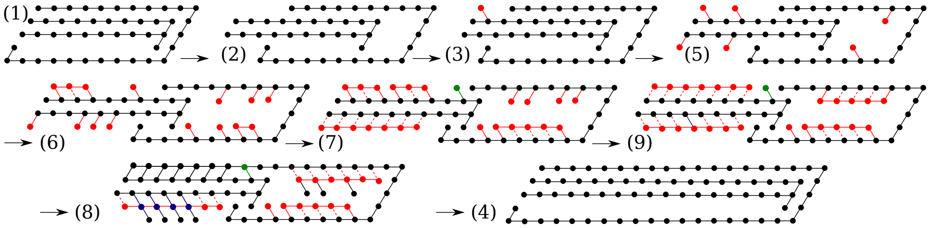


Fig. 7. Line doubling construction. The inner component is called the *sword*, which agitates left/right relative to the outer component called the scabbard (both are in black). The black sword-and-scabbard are doubled from length $m = 8$ to length $2m = 16$. Other monomers (red, green, blue) serve to both ratchet the movement, and to quickly in parallel build up the mass of the doubled sword-scabbard.

nubot monomers (each in state 0 or 1) with neighbouring monomers bound by rigid bonds, placed somewhere on the hexagonal grid. The leftmost of these monomers triggers the growth of a constant sized (length 1) sword and scabbard structure. Then an iterated doubling process begins, that happens exactly ℓ times and will result in a sword-and-scabbard of length n (and height 5). At step i of doubling, $1 \leq i \leq \ell$, the leftmost of the input monomers x_i (from x) is read, and then deleted. If $x_i = 0$ then there will be a doubling of the length of the sword-and-scabbard, else if $x_i = 1$ there will be a doubling of the length of the sword-and-scabbard with the addition of one extra monomer. It is straightforward to check that this doubling algorithm finishes with a length n object after ℓ rounds. After the final doubling step, a synchronization occurs, and then $\leq 4n$ of the monomers are deleted (in parallel) in such a way that an $n \times 1$ line remains. All that remains is to show the details of how each doubling step works.

Doubling Construction. Figure 7 describes the doubling process in detail: at iteration i of doubling assume that (a) we read an input bit 0, and that (b) we have a sword-and-scabbard structure of length m (and height 5). Since the input bit is 0 we want to double the length to $2m$. As shown in Figure 7(1), we begin with the sword sheathed in the scabbard. We next describe a biased (or ratcheted) random walk process that will ultimately result in the sword being withdrawn all the way to the *hook*, giving a structure of length $2m$. Via agitation, the sword may be unsheathed by moving out (to the left) of the scabbard, or by the scabbard moving (to the right) from the sword, although, because of the hook the sword can never be completely withdrawn and hence the two components will never drift apart.[1] The withdrawing of the sword is a random walk process with both the sword and scabbard agitating left-right. While this is happening, each monomer—at unit rate, conditioned on that monomer being unsheathed—on the top row of the sword tries to attach a new monomer above. Any such attachment event that succeeds acts as a *ratchet* that biases the ran-

[1] Besides preserving correctness of the construction, the hook is a safety feature, and hence the sword is merely decorative.

dom walk process in the forward direction. Also, as the sword is unsheathed each unsheathed sword monomer at the bottom of the sword attaches—at unit rate, conditioned on that monomer being unsheathed—a monomer below, and each monomer on the top (respectively, bottom) horizontal row of the scabbard tries to attach a monomer below (respectively, above) it. These monomers can also serve as ratchets (although in our time analysis below we ignore them which serves only to slow down the analysis). Eventually the sword is completely withdrawn to the hook, and ratcheted at that position, so further agitations do not change the structure.

At this point we are done with the doubling step, and the sword and scabbard reconfigure themselves to prepare for the next doubling (or deletion of monomers if we are done). Figure 7(6)–(9) gives the details. The attachment of new monomers results in 4 new horizontal line segments, each of length $m-1$. Each segment is built in the same way as used for the synchronization technique shown in Section 3, Figure 4; specifically the bonds are initially formed as rigid, and then transition to flexible in such a way that the line segment (or "synchronization" row) is free to agitate relative to its "backbone" row only when exactly all m bonds have formed. The line agitates left and right and is then synchronized (or locked into place, see Figure 4) causing all m monomers on the line to change state to "done". When the two new line segments that attached to the bottom and top of the sword are both done their rightmost monomers each bind to the scabbard with a rigid bond (as shown in Figure 7(8)) and delete their bonds to the sword (Figure 7(9)) (note that the rightmost of the latter kind of bonds is not deleted until after binding to the scabbard which ensures the entire structure remains connected at all times; also before the leftmost bond on the bottom is deleted a new hook is formed which prevents the new sword leaving the new scabbard prematurely). In a similar process, the two new line segments that are attached to the scabbard form a new hook, bind themselves to the sword, and then release themselves from the scabbard. We are new ready for the next stage of doubling.

The previous description assumed that the input bit is 0. If the input bit is instead 1 then after doubling both the sword and scabbard are increased in length by 1 monomer (immediately before forming the hook on the new scabbard).

After the final doubling stage then $O(n)$ monomers need to be deleted to leave an $n \times 1$ line of rigidly bound monomers (the goal is to build a line) without having monomers drift away (so as not to violate the space bound). This is relatively straightforward to achieve with two synchronizations, and subsequent deletion of monomers.

Expected Time Analysis. Lemma 6 states the expected time for a single doubling event: a length m sword is fully withdrawn to the hook, and locked into place, from a length m scabbard in expected time $O(m^{1/3+\epsilon})$.

Between each doubling event there is a reconfiguration of the sword and scabbard. Each reconfiguration invokes a constant number of synchronizations which, via Lemma 4, take expected time $O(\log m)$ each. Changing of the bond structure also takes place in $O(\log m)$ expected time since each of the four new line segments change their bonds independently, and within a line segment all bond changes (expect for a constant number) occur independently and in parallel.

There are $\ell = \lfloor \log_2 n \rfloor + 1$ doubling plus reconfiguration events, each taking time $c((2^k)^{1/3+\epsilon})$ on the k'th event for some constant c (by Lemma 6, since the size of the structure during the k'th event is $\Theta(2^k)$), the total expected time is bounded by the geometric series

$$\sum_{k=0}^{\ell-1} c(2^k)^{1/3+\epsilon} = c\sum_{k=0}^{\ell-1}(2^{1/3+\epsilon})^k = c\frac{1-(2^{1/3+\epsilon})^\ell}{1-2^{1/3+\epsilon}} = O((2^{1/3+\epsilon})^\ell) = O(n^{1/3+\epsilon}).$$

□

The next lemma states that, starting from length m, one "length-doubling" stage of the 1D line construction completes in expected time $O(m^{1/3+\epsilon})$. Intuitively, the proof (see full paper) shows that the rapid agitation process is a random walk that quickly exposes a large portion of the sword, to which a monomer quickly attaches. This attachment irreversibly "ratchets" the random walk forward, preventing it from walking backwards by very much.

Lemma 6. *For any $\epsilon > 0$, for sufficiently large m, the expected time for one line-doubling stage (doubling the length of the sword and scabbard) is $O(m^{1/3+\epsilon})$.*

Acknowledgments. A special thanks to Erik Winfree for many insightful and helpful discussions on the model and constructions. We also thank Robert Schweller, Matthew Cook and Andrew Winslow for discussions on the model and problems studied in this paper.

References

1. Bath, J., Turberfield, A.: DNA nanomachines. Nature Nanotechnology 2, 275–284 (2007)
2. Chen, M., Xin, D., Woods, D.: Parallel computation using active self-assembly. In: Soloveichik, D., Yurke, B. (eds.) DNA 2013. LNCS, vol. 8141, pp. 16–30. Springer, Heidelberg (2013); Full version: `arXiv:1405.0527`
3. Dabby, N., Chen, H.-L.: Active self-assembly of simple units using an insertion primitive. In: SODA: Proceedings of the Twenty-fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1526–1536 (January 2012)
4. Dawes-Hoang, R.E., Parmar, K.M., Christiansen, A.E., Phelps, C.B., Brand, A.H., Wieschaus, E.F.: Folded gastrulation, cell shape change and the control of myosin localization. Development 132(18), 4165–4178 (2005)
5. Doty, D.: Theory of algorithmic self-assembly. Communications of the ACM 55, 78–88 (2012)
6. Doty, D., Lutz, J.H., Patitz, M.J., Schweller, R.T., Summers, S.M., Woods, D.: The tile assembly model is intrinsically universal. In: FOCS: Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, pp. 439–446 (October 2012)

7. Martin, A.C., Kaschube, M., Wieschaus, E.F.: Pulsed contractions of an actin–myosin network drive apical constriction. Nature 457(7228), 495–499 (2008)
8. Patitz, M.J.: An introduction to tile-based self-assembly. In: Durand-Lose, J., Jonoska, N. (eds.) UCNC 2012. LNCS, vol. 7445, pp. 34–62. Springer, Heidelberg (2012)
9. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Institute of Technology (June 1998)
10. Woods, D.: Intrinsic universality and the computational power of self-assembly. In: MCU: Proceedings of Machines, Computations and Universality, September 9-12. Electronic Proceedings in Theoretical Computer Science, vol. 128, pp. 16–22 (2013)
11. Woods, D., Chen, H.-L., Goodfriend, S., Dabby, N., Winfree, E., Yin, P.: Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In ITCS 2013: Proceedings of the 4th conference on Innovations in Theoretical Computer Science, pp. 353–354. ACM (2013) Full version: `arXiv:1301.2626` [cs.DS]

Probability 1 Computation with Chemical Reaction Networks*

Rachel Cummings[1], David Doty[2], and David Soloveichik[3]

[1] Northwestern University, Evanston, Illinois, USA
rachelc@u.northwestern.edu
[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. The computational power of stochastic chemical reaction networks (CRNs) varies significantly with the output convention and whether or not error is permitted. Focusing on probability 1 computation, we demonstrate a striking difference between *stable* computation that converges to a state where the output cannot change, and the notion of *limit-stable* computation where the output eventually stops changing with probability 1. While stable computation is known to be restricted to semilinear predicates (essentially piecewise linear), we show that limit-stable computation encompasses the set of predicates in Δ_2^0 in the arithmetical hierarchy (a superset of Turing-computable). In finite time, our construction achieves an error-correction scheme for Turing universal computation. This work refines our understanding of the tradeoffs between error and computational power in CRNs.

1 Introduction

Recent advances in the engineering of complex artificial molecular systems have stimulated new interest in models of chemical computation. How can chemical reactions process information, make decisions, and solve problems? A natural model 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 $A + B \to A + C$. Subject to discrete semantics (integer number of molecules) the model corresponds to a continuous time, discrete state, Markov process. A state of the system is a vector of non-negative integers specifying the molecular counts of the species (e.g., A, B, C), a reaction can occur only when its reactants are present, and transitions between states correspond to reactions (i.e., when the above reaction occurs the count of B is decreased by 1 and the count of C increased by 1). CRNs are used extensively to describe natural biochemical systems in the cellular context. Other natural sciences use the

* The first author was supported by NSF grants CCF-1049899 and CCF-1217770, the second author was supported by NSF grants CCF-1219274 and CCF-1162589 and the Molecular Programming Project under NSF grant 1317694, and the third author was supported by NIGMS Systems Biology Center grant P50 GM081879.

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 37–52, 2014.

model as well: for example in ecology an equivalent model is widely employed to describe population dynamics [16]. Recently, CRNs began serving as a programming language for engineering artificial chemical systems. In particular, DNA strand displacement systems have the flexibility to realize arbitrarily complex interaction rules [4, 6, 15], demonstrating that arbitrary CRNs have a chemical implementation. Outside of chemistry, engineering of sensor networks and robot swarms often uses CRN-like specification rules to prescribe behavior [2].

The exploration of the computational power of CRNs rests on a strong theoretical foundation, with extensive connections to other areas of computing. Similar models have arisen repeatedly in theoretical computer science: Petri nets [11], vector addition systems [10], population protocols [2], etc. They share the fundamental feature of severely limited "agents" (molecules), with complex computation arising only through repeated interactions between multiple agents. In population protocols it is usually assumed that the population size is constant, while in CRNs molecules could be created or destroyed[1] — and thus different questions are sometimes natural in the two settings.

Informally speaking, we can identify two general kinds of computation in CRNs. In *non-uniform* computation, a single CRN computes a function over a finite domain. This is analogous to Boolean circuits in the sense that any given circuit computes only on inputs of a particular size (number of bits), and to compute on larger inputs a different circuit is needed. Conversely, in *uniform* computation, a single CRN computes on all possible input vectors. This is analogous to Turing machines that are expected to handle inputs of arbitrary size placed on their (unbounded) input tape. In this work we focus entirely on uniform computation.[2]

We focus on the question of the distinguishability of initial states by a CRN. We view CRNs as computing a Boolean valued predicate on input vectors in $\mathbb{N}^k$ that are the counts of certain input species $X_1, \ldots, X_k$. We believe that a similar characterization holds for more general function computation where the output is represented in counts of output species (e.g. $f : \mathbb{N}^k \to \mathbb{N}^l$), but that remains to be shown in future work.

Previous research on uniform computation has emphasized the difference in computational power between paradigms intended to capture the intuitive notions of error-free and error-prone computation [7]. In contrast to many other models of computing, a large difference was identified between the two settings for CRNs. We now review the previously studied error-free (probability 1) and error-prone (probability < 1) settings. In this paper we develop an error correction scheme that reduces the error of the output with time and achieves probability 1 computation in the limit. Thus, surprisingly the large distinction

[1] Having an external (implicit) supply of fuel molecules avoids violating the conservation of mass.

[2] However, it is important to keep in mind that settings with extremely weak uniform computational power may nonetheless achieve complex computation from the non-uniform perspective. For example, probability 1 committing computation (see below), while restricted to constant predicates in our setting, can simulate arbitrary Boolean circuits with a proper encoding of input and output (e.g. using so called "dual-rail" encoding with two species per input bit).

Table 1. Categorizing the computational power of CRNs by output convention and allowed error probability. In all cases we consider the class of total predicates $\phi : \mathbb{N}^k \to \{\text{no}, \text{yes}\}$ ("total" means ϕ must be defined on all input values). The input consists of the initial molecular counts of input species $X_1, \ldots, X_k$. *Committing*: In this output convention, producing any molecules of N indicates that the output of the whole computation is "no", and producing any molecules of Y indicates "yes". *Stable*: Let the *output of a state* be "no" if there are some N molecules and no Y molecules (and vice versa for "yes"). In the stable output convention, the output of the whole computation is $b \in \{\text{no}, \text{yes}\}$ when the CRN reaches a state with output value b from which every reachable state also has output value b. *Limit-stable*: The output of the computation is considered $b \in \{\text{no}, \text{yes}\}$ when the CRN reaches a state with output b and never changes it again (even though states with different output may remain reachable). The parenthetical settings have not been explicitly formalized; however the computational power indicated naturally follows by extending the results from other settings.[3]

	committing	**stable**	**limit-stable**
prob correct $= 1$	(constant)	semilinear [1]	Δ_2^0 [this work]
prob correct < 1	computable [14]	(computable)	(Δ_2^0)

between probability 1 and probability $<$ 1 computation disappears in a "limit computing" setting.

The best studied type of probability 1 computation incorporates the stable output criterion (Table 1, prob correct = 1, stable) and was shown to be limited to semilinear predicates [1] (later extended to functions [5]). For example, consider the following CRN computing the parity predicate (a semilinear predicate). The input is encoded in the initial number of molecules of X, and the output is indicated by species Y (yes) and N (no):

$$X + Y \to N, \quad X + N \to Y$$

Starting with the input count $n \in \mathbb{N}$ of X, as well as $1Y$, the CRN converges to a state where a molecule of the correct output species is present (Y if n is even and N if n is odd) and the incorrect output species is absent. From that point on, no reaction can change the output.

Motivated by such examples, probability 1 *stable* computation admits the changing of output as long as the system converges with probability 1 to an *output stable state* — a state from which no sequence of reactions can change the output. In the above example, the states with X absent are output stable.

[3] Probability 1 committing computation: Suppose there are two inputs $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{N}^k$ such that $\phi(\mathbf{x}_1) = \text{no}$ and $\phi(\mathbf{x}_2) = \text{yes}$. Consider any input $\mathbf{x}_3$ such that $\mathbf{x}_3 \geq \mathbf{x}_1$ and $\mathbf{x}_3 \geq \mathbf{x}_2$. From state $\mathbf{x}_3$ we can produce N by the sequence of reactions from $\phi(\mathbf{x}_1)$, and Y by the sequence of reactions from $\phi(\mathbf{x}_2)$. Both sequences occur with some non-zero probability, and so the probability of error is non-zero for non-constant predicates. Probability $<$ 1 stable computation: The question of whether there is a sequence of reactions to change the output is computable and thus the stable output convention cannot give more computational power than the committing output convention. Probability $<$ 1 limit-stable computation: Our negative result for probability 1 limit-stable computation can be modified to apply.

(Although the limits on stable computation were proven in the population protocols model [1] they also apply to general CRNs where infinitely many states may be reachable.)

A more stringent output convention requires irreversibly producing N or Y to indicate the output (i.e., "Y is producible from initial state $\mathbf{x}$" $\iff$ "N is not producible from $\mathbf{x}$" $\iff$ $\phi(\mathbf{x}) = \text{yes}$). We term such output convention *committing*. However, probability 1 committing computation is restricted to constant predicates[4] (Table 1, prob correct = 1, committing).

Intuitively, committing computation "knows" when it is done, while stable computation does not. Nonetheless, stable computation is not necessarily impractical. All semilinear predicates can be stably computed such that checking for output stability is equivalent to simply inspecting whether any further reaction is possible — so an outside observer can easily recognize the completion of computation [3]. While stable CRNs do not know when they are done computing, different downstream processes can be catalyzed by the N and Y species. As long as these processes can be undone by the presence of the opposite species, the overall computation can be in the sense stable as well. A canonical downstream process is signal amplification in which a much larger population of $\hat{N}, \hat{Y}$ is interconverted by reactions $\hat{N} + Y \to \hat{Y} + Y$ and $\hat{Y} + N \to \hat{N} + N$. Finally all semilinear predicates can be stably computed quickly (polylogarithmic in the input molecular counts).

In contrast to the limited computational abilities of the probability 1 settings just now discussed, tolerating a positive probability of error significantly expands computational power. Indeed arbitrary computable functions (Turing universal computation) can be computed with the committing output convention [14]. Turing universal computation can also be fast — the CRN simulation incurs only a polynomial slowdown. However, error is unavoidable and is due fundamentally to the inability of CRNs to deterministically detect the absence of species. (Note that Turing universal computation is only possible in CRNs when the reachable state space, i.e., molecular count, is unbounded — and is thus not meaningful in population protocols.)

When a CRN is simulating a Turing machine, errors in simulation cannot be avoided. However, can they be *corrected* later? In this work we develop an error correction scheme that can be applied to Turing universal computation that ensures overall output error decreases the longer the CRN runs. Indeed, in the limit of time going to infinity, with probability 1 the answer is correct.

To capture Turing-universal probability 1 computation with such an error correction process a new output convention is needed, since the committing and stable conventions are limited to much weaker forms of probability 1 computation (Table 1). *Limit-stability* subtly relaxes the stability requirement: instead of there being no path to change the output, we require the system to eventually stop taking such paths with probability 1. To illustrate the difference between the original notion of stability and our notion of *limit-stability*, consider the

[4] Particulars of input encoding generally make a difference for the weaker computational settings. Indeed, probability 1 committing computation can compute more than constant predicates if they are not required to be total. For example, a committing CRN can certainly distinguish between two $\leq$-incomparable inputs.

reachability of the empty state (without any molecules of Y, in which the output is undefined) in the CRN:

$$\varnothing \xrightarrow{1} Y, \quad Y \xrightarrow{1} \varnothing, \quad Y \xrightarrow{2} 2Y$$

From any reachable state, the empty state is reachable (just execute the second reaction enough times). However, with probability 1, the empty state is visited only a finite number of times.[5] If, as before, we think of the presence of Y indicating a yes output, then the yes output is never stable (the empty state is always reachable), but with probability 1 a yes output will be produced and never change — i.e., it is limit-stable.

We show that with the limit-stable output convention, errors in Turing universal computation can be rectified eventually with probability 1. Our construction is based on simulating a register machine (a.k.a. Minsky counter machine) over and over in an infinite loop, increasing the number of simulated steps each time (dovetailing). Each time the CRN updates its answer to the answer given by the most recently terminated simulation. While errors occur during these simulations, our construction is designed such that with probability 1 only a finite number of errors occur (by the Borel-Cantelli Lemma), and then after some point the output will stay correct forever. The main difficulty is ensuring that errors "fail gracefully": they are allowed to cause the wrong answer to appear for a finite time, but they cannot, for instance, interfere with the dovetailing itself. Thus, although errors are unavoidable in a CRN simulation of a register machine, they can subsequently be corrected for with probability 1. We also show that the expected time to stabilize to the correct output is polynomial in the running time of the register machine (which, however, is exponentially slower than a Turing machine).

It is natural to wonder if limit-stable probability 1 computation in CRNs characterizes exactly the computable languages. However, we show that the class of predicates limit-stable computable by CRNs with probability 1 is exactly the so-called Δ^0_2 predicates (at the second level of the arithmetical hierarchy [12]). The class Δ^0_2 can be characterized in terms of "limit computing" of Turing machines that can change their output a finite but unbounded number of times [13]. Thus it is not surprising that relaxing the output convention from committing to limit-stable for probability < 1 computation increases the computational power from the computable predicates to Δ^0_2. However, the key contribution of this paper is that the gap between probability < 1 and probability 1 computation completely disappears with limit-stable output.

Note that in no way should our result be interpreted to mean that chemistry violates the Church-Turing thesis. Since we do not know when the CRN will stop changing its answer, the output of a limit-stable computation cannot be practically read out in finite time.

[5] The second and third reactions are equivalent to the gambler's ruin problem [8], which tells us that, because the probability of increasing the count of Y is twice that of decreasing its count, there is a positive probability that Y's count never reaches 0. The first reaction can only increase this probability. Since whenever Y reaches 0, we have another try, eventually with probability 1 we will not reach 0.

Relaxing the definition of probability 1 computation further does not increase computational power. An alternative definition of probability 1 limit-stable computation is to require that as time goes to infinity, the probability of expressing the correct output approaches 1. In contrast to limit-stability, the output may change infinitely often. Note that this is exactly the distinction between almost sure convergence and convergence in probability. Our proof that probability 1 limit-stable computation is limited to Δ_2^0 predicates applies to this weaker sense of convergence as well, bolstering the generality of our result. Interestingly, it is still unclear whether in CRNs there is a natural definition of probability 1 computation that exactly corresponds to the class of Turing-computable functions.

In the remainder of this paper we focus on the type of probability 1 computation captured by the notion of limit-stability, reserving the term *probability* 1 *computation* to refer specifically to probability 1 limit-stable computation.

To our knowledge, the first hints in the CRN literature of probability 1 Turing universal computation occur in ref. [17], where Zavattaro and Cardelli showed that the following question is uncomputable: Will a given CRN with probability 1 reach a state where no further reactions are possible? Although their construction relied on repeated simulations of a Turing machine, it did not use the Borel-Cantelli Lemma, and could not be directly applied to computation with output.

2 Preliminaries

Computability theory. We use the term *predicate* (a.k.a. *language*, *decision problem*) interchangeably to mean a subset $L \subseteq \mathbb{N}^k$, or equivalently a function $\phi : \mathbb{N}^k \to \{0, 1\}$, such that $\phi(\mathbf{x}) = 1 \iff \mathbf{x} \in L$. The class Δ_2^0 of predicates at the second level of the arithmetical hierarchy (cf. [12]) has many equivalent definitions. In this paper we use the following characterization (see e.g. [13]). A predicate $\phi : \mathbb{N}^k \to \{0, 1\}$ is *limit computable*, and we write $\phi \in \Delta_2^0$, if there is a computable function $r : \mathbb{N}^k \times \mathbb{N} \to \{0, 1\}$ such that, for all $\mathbf{x} \in \mathbb{N}^k$, $\lim\limits_{t\to\infty} r(\mathbf{x}, t) = \phi(\mathbf{x})$.

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 $\mathbf{c} \in \mathbb{N}^\Lambda$ as a vector $\mathbf{c} \in \mathbb{N}^{|\Lambda|}$ of $|\Lambda|$ nonnegative integers, with each coordinate "labeled" by an element of Λ. Given $S \in \Lambda$ and $\mathbf{c} \in \mathbb{N}^\Lambda$, we refer to $\mathbf{c}(S)$ as the *count of S in* $\mathbf{c}$. We write $\mathbf{c} \leq \mathbf{c}'$ if $\mathbf{c}(S) \leq \mathbf{c}'(S)$ for all $S \in \Lambda$, and $\mathbf{c} < \mathbf{c}'$ if $\mathbf{c} \leq \mathbf{c}'$ and $\mathbf{c} \neq \mathbf{c}'$. Given $\mathbf{c}, \mathbf{c}' \in \mathbb{N}^\Lambda$, we define the vector component-wise operations of addition $\mathbf{c} + \mathbf{c}'$ and subtraction $\mathbf{c} - \mathbf{c}'$. For a set $\Delta \subset \Lambda$, we view a vector $\mathbf{c} \in \mathbb{N}^\Delta$ equivalently as a vector $\mathbf{c} \in \mathbb{N}^\Lambda$ by assuming $\mathbf{c}(S) = 0$ for all $S \in \Lambda \setminus \Delta$.

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 (amount consumed/ produced) of the reactants and products, respectively, and the *rate constant* k. For instance, given $\Lambda = \{A, B, C\}$, the reaction $A + 2B \overset{7.5}{\to} A + 3C$ is represented by the triple $\langle (1, 2, 0), (1, 0, 3), 7.5 \rangle$. If not specified, assume that the rate constant $k = 1$. A *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 *state* of a CRN $N = (\Lambda, R)$ is a vector $\mathbf{c} \in \mathbb{N}^\Lambda$.

Given a state $\mathbf{c}$ and reaction $\alpha = \langle \mathbf{r}, \mathbf{p} \rangle$, we say that α is *applicable* to $\mathbf{c}$ if $\mathbf{r} \leq \mathbf{c}$ (i.e., $\mathbf{c}$ contains enough of each of the reactants for the reaction to occur). If α is applicable to $\mathbf{c}$, then write $\alpha(\mathbf{c})$ to denote the state $\mathbf{c} + \mathbf{p} - \mathbf{r}$ (i.e., the state that results from applying reaction α to $\mathbf{c}$). If $\mathbf{c}' = \alpha(\mathbf{c})$ for some reaction $\alpha \in R$, we write $\mathbf{c} \to^1 \mathbf{c}'$. An *execution sequence* $\mathcal{E}$ is a finite or infinite sequence of states $\mathcal{E} = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \ldots)$ such that, for all $i \in \{1, \ldots, |\mathcal{E}| - 1\}$, $\mathbf{c}_{i-1} \to^1 \mathbf{c}_i$. If a finite execution sequence starts with $\mathbf{c}$ and ends with $\mathbf{c}'$, we write $\mathbf{c} \to \mathbf{c}'$, and we say that $\mathbf{c}'$ is *reachable from* $\mathbf{c}$.

Stable decidability by CRNs. We now review the definition of stable decidability of predicates introduced by Angluin, Aspnes, and Eisenstat [1]. Although our main theorem concerns probability 1 decidability, not stable decidability, many of the definitions of this section will be required, so it it useful to review. Intuitively, some species "vote" for a yes/no answer, and a CRN is a stable decider if it is guaranteed to reach a consensus vote that cannot change.

A *chemical reaction decider* (CRD) is a tuple $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon, \phi, \mathbf{s})$, where (Λ, R) is a CRN, $\Sigma \subseteq \Lambda$ is the *set of input species*,[6] $\Upsilon \subseteq \Lambda$ is the set of *voters*, $\phi : \Upsilon \to \{0, 1\}$ is the *(Boolean) output function*, and $\mathbf{s} \in \mathbb{N}^{\Lambda \setminus \Sigma}$ is the *initial context*. An input to $\mathcal{D}$ is a vector $\mathbf{x} \in \mathbb{N}^\Sigma$, or equivalently $\mathbf{x} \in \mathbb{N}^k$ if $|\Sigma| = k$; $\mathcal{D}$ and $\mathbf{x}$ define an *initial state* $\mathbf{i} \in \mathbb{N}^\Lambda$ as $\mathbf{i} = \mathbf{s} + \mathbf{x}$ (when $\mathbf{i}$ and $\mathbf{x}$ are considered as elements of $\mathbb{N}^\Lambda$).[7] When $\mathbf{i}$ is clear from context, we say that a state $\mathbf{c}$ is *reachable* if $\mathbf{i} \to \mathbf{c}$.

We extend ϕ to a partial function $\Phi : \mathbb{N}^\Lambda \dashrightarrow \{0, 1\}$ as follows. $\Phi(\mathbf{c})$ is undefined if either $\mathbf{c}(V) = 0$ for all $V \in \Upsilon$, or if there exist $N, Y \in \Upsilon$ such that $\mathbf{c}(N) > 0$, $\mathbf{c}(Y) > 0$, $\phi(N) = 0$ and $\phi(Y) = 1$. Otherwise, $(\exists b \in \{0, 1\})(\forall V \in \Upsilon)(\mathbf{c}(V) > 0 \implies \phi(V) = b)$; in this case, the *output* $\Phi(\mathbf{c})$ of state $\mathbf{c}$ is b. In other words $\Phi(\mathbf{c}) = b$ if some voters are present and they all vote b.

If $\Phi(\mathbf{y})$ is defined and every state $\mathbf{y}'$ reachable from $\mathbf{y}$ satisfies $\Phi(\mathbf{y}) = \Phi(\mathbf{y}')$, then we say that $\mathbf{y}$ is *output stable*, i.e., if $\mathbf{y}$ is ever reached, then no reactions can ever change the output. We say that $\mathcal{D}$ *stably decides* the predicate $\phi : \mathbb{N}^k \to \{0, 1\}$ if, for all input states $\mathbf{x} \in \mathbb{N}^k$, for every state $\mathbf{c}$ reachable from $\mathbf{x}$, there is an output stable state $\mathbf{y}$ reachable from $\mathbf{c}$ such that $\Phi(\mathbf{y}) = \phi(\mathbf{x})$. In other words, no sequence of reactions (reaching state $\mathbf{c}$) can *prevent* the CRN from being able to reach the correct answer (since $\mathbf{c} \to \mathbf{y}$ and $\Phi(\mathbf{y}) = \phi(\mathbf{x})$) and staying there if reached (since $\mathbf{y}$ is output stable).[8]

[6] In Section 3 and beyond, we restrict attention to the case that $|\Sigma| = 1$, i.e., single-integer inputs.

[7] In other words, species in $\Lambda \setminus \Sigma$ must always start with the same counts, and counts of species in Σ are varied to represent different inputs to $\mathcal{D}$, similarly to a Turing machine that starts with different binary string inputs, but the Turing machine must always start with the same initial state and tape head position.

[8] At first glance, this definition appears too weak to claim that the CRN is "guaranteed" to reach the correct answer. However, if the set of reachable states is finite, it implies that a correct output stable state is actually reached with probability 1, assuming the model of stochastic kinetics defined later. Conversely, if a correct output-stable state is reached with probability 1, then this implies a correct output-stable state is always reachable, even with an infinite reachable state space.

Probability 1 decidability by CRNs. In order to define probability 1 computation with CRNs, we first review the model of stochastic chemical kinetics. It is widely used in quantitative biology and other fields dealing with chemical reactions between species present in small counts [9]. It ascribes probabilities to execution sequences, and also defines the time of reactions.

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.

The kinetics of a CRN is described by a continuous-time Markov process as follows. The system has some volume $v \in \mathbb{R}^+$ that affects the transition rates, which may be fixed or allowed to vary over time; in this paper we assume a constant volume of 1.[9] In state $\mathbf{c}$, the *propensity* of a unimolecular reaction $\alpha : X \overset{k}{\to} \ldots$ in state $\mathbf{c}$ is $\rho(\mathbf{c}, \alpha) = k \cdot \mathbf{c}(X)$. The propensity of a bimolecular reaction $\alpha : X + Y \overset{k}{\to} \ldots$, where $X \neq Y$, is $\rho(\mathbf{c}, \alpha) = k \cdot \frac{\mathbf{c}(X) \cdot \mathbf{c}(Y)}{v}$. The propensity of a bimolecular reaction $\alpha : X + X \overset{k}{\to} \ldots$ is $\rho(\mathbf{c}, \alpha) = \frac{k}{2} \cdot \frac{\mathbf{c}(X) \cdot (\mathbf{c}(X) - 1)}{v}$. The propensity function governs the evolution of the system as follows. The time until the next reaction occurs is an exponential random variable with rate $\rho(\mathbf{c}) = \sum_{\alpha \in R} \rho(\mathbf{c}, \alpha)$ (note $\rho(\mathbf{c}) = 0$ if and only if no reactions are applicable to $\mathbf{c}$). The probability that next reaction will be a particular α_{next} is $\frac{\rho(\mathbf{c}, \alpha_{\text{next}})}{\rho(\mathbf{c})}$.

We now define probability 1 computation by CRDs. Our definition is based on the *limit-stable* output convention as discussed in the introduction. Let $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon, \phi, \mathbf{s})$ be a CRD. Let $\mathcal{E} = (\mathbf{c}_0, \mathbf{c}_1, \ldots)$ be an execution sequence of $\mathcal{D}$. In general $\mathcal{E}$ could be finite or infinite, depending on whether $\mathcal{D}$ can reach a terminal state (one in which no reaction is applicable); however, in this paper all CRDs will have no reachable terminal states, so assume $\mathcal{E}$ is infinite. We say that $\mathcal{E}$ has a defined output if there exists $b \in \{0, 1\}$ such that, for all but finitely many $i \in \mathbb{N}$, $\Phi(\mathbf{c}_i) = b$. In other words, $\mathcal{E}$ eventually stabilizes to a certain answer. In this case, write $\Phi(\mathcal{E}) = b$; otherwise, let $\Phi(\mathcal{E})$ be undefined.

If $\mathbf{x} \in \mathbb{N}^k$, write $\mathcal{E}(\mathcal{D}, \mathbf{x})$ to denote the random variable representing an execution sequence of $\mathcal{D}$ on input $\mathbf{x}$, resulting from the Markov process described previously. We say that $\mathcal{D}$ *decides* $\phi : \mathbb{N}^k \to \{0, 1\}$ *with probability 1* if, for all $\mathbf{x} \in \mathbb{N}^k$, $\Pr[\Phi(\mathcal{E}(\mathcal{D}, \mathbf{x})) = \phi(\mathbf{x})] = 1$.

3 Turing-Decidable Predicates

This section describes how a CRD can decide an arbitrary Turing-decidable predicate with probability 1. This construction also contains most of the technical details needed to prove our positive result that CRDs can decide arbitrary Δ_2^0 predicates with probability 1. The proof is via simulation of register machines,

[9] A common restriction is to assume the *finite density constraint*, which stipulates that arbitrarily large mass cannot occupy a fixed volume, and thus the volume must grow proportionally with the total molecular count. With some minor modifications to ensure *relative* rates of reactions stay the same (even though all bimolecular reactions would be slowed down in absolute terms), our construction would work under this assumption, although the time analysis would change. For the sake of conceptual clarity, we present the construction assuming a constant volume.

which are able to simulate arbitrary Turing machines if at least 3 registers are used. This section describes the simulation and gives intuition for how it works. Section 4 proves its correctness (proof sketch in this version of the paper). Section 5 shows how to extend the construction to handle Δ_2^0 predicates and prove that no more predicates can be decided with probability 1 by a CRD.

3.1 Register Machines

A *register machine* M has m *registers* $r_1, \ldots, r_m$ that can each hold a nonnegative integer. M is programmed by a finite sequence (*lines*) of *instructions.* There are four types of instructions: `accept`, `reject`, `inc(`r_j`)`, and `dec(`r_j`,`k`)`. For simplicity, we describe our construction for single-input register machines and thus predicates $\phi : \mathbb{N} \to \{0, 1\}$, but it can be easily extended to more inputs. The input $n \in \mathbb{N}$ to M is the initial value of register r_1, and the remaining $m-1$ registers are used to perform a computation on the input, which by convention are set to 0 initially. The semantics of execution of M is as follows. The *initial line* is the first instruction in the sequence. If the current line is `accept` or `reject`, then M halts and accepts or rejects, respectively. If the current line is `inc(`r_j`)`, then register r_j is incremented, and the next line in the sequence becomes the current line. If the current line is `dec(`r_j`,`k`)`, then register r_j is decremented, and the next line in the sequence becomes the current line, unless $r_j = 0$, in which case it is left at 0 and line k becomes the current line. In other words, M executes a straight-line program, with a "conditional jump" that occurs when attempting to decrement a 0-valued register. For convenience we assume there is a fifth type of instruction `goto(`k`)`, meaning "unconditionally jump to line k". This can be indirectly implemented by decrementing a special register r_0 that always has value 0, or easily implemented in a CRN directly. The set of input values n that cause the machine to accept is then the language/predicate decided by the machine. For example, the following register machine decides the parity of the initial value of register r_1:

```
1: dec(r1,5)
2: dec(r1,4)
3: goto(1)
4: accept
5: reject
```

Chemical reaction networks can be used to simulate any register machine through a simple yet error-prone construction, which is similar to the simulation described in [14]. We now describe the simulation and highlight the source of error. Although this simulation may be error-prone, the effect of the errors has a special structure, and our main construction will take advantage of this structure to keep errors from invalidating the entire computation. Specifically, there is a possibility of an error precisely when the register machine performs a conditional jump. We highlight this fact in the construction below to motivate the modifications we make to the register machine in Section 3.2.

For a register machine with l lines of instructions and m registers, create molecular species $L_1, \ldots, L_l$ and $R_1, \ldots, R_m$. The presence of molecule L_i is used to indicate that the register machine is in line i. Since the register machine can only be in one line at a time, there will be exactly one molecule of the form L_i present in the solution at any time. The count of species R_j represents the current value of register r_j. The following table shows the reactions to simulate an instruction of the register machine, assuming the instruction occurs on line i:

`accept`	$L_i \to H_Y$
`reject`	$L_i \to H_N$
`goto(`k`)`	$L_i \to L_k$
`inc(`r_j`)`	$L_i \to L_{i+1} + R_j$
`dec(`r_j`,`k`)`	$L_i + R_j \to L_{i+1}$ $L_i \to L_k$

The first four reactions are error-free simulations of the corresponding instructions. The final two reactions are an error-prone way to decrement register r_j. If $r_j = 0$, then only the latter reaction is possible, and when it occurs it is a correct simulation of the instruction. However, if $r_j > 0$ (hence there are a positive number R_j molecules in solution), then either reaction is possible. While only the former reaction is correct, the latter reaction could still occur. The semantic effect this has on the register machine being simulated is that, when a decrement `dec(`r_j`,`k`)` is possible because $r_j > 0$, the machine may nondeterministically jump to line k anyway. Our two goals in the subsequently described construction are 1) to reduce sufficiently the probability of this error occurring each time a decrement instruction is executed so that with probability 1 errors eventually stop occurring, and 2) to set up the simulation carefully so that it may recover from any finite number of errors.

3.2 Simulating Register Machine

We will first describe how to modify M to obtain another register machine S which is easier for the CRD to simulate repeatedly to correct errors. There are two general modifications we make to M to generate S. The first consists of adding several instructions before M's first line and at the end (denoted line h below). The second consists of adding a pair of instructions before every decrement of M. We now describe these modifications in more detail.

Intuitively, S maintains a bound $b \in \mathbb{N}$ on the total number of decrements M is allowed to perform, and S halts if M exceeds this bound.[10] Although M halts if no errors occur, an erroneous jump may take M to a configuration unreachable from the initial configuration, so that M would not halt even if simulated correctly from that point on. To ensure that S always halts, it stores b in such a way that errors cannot corrupt its value. S similarly stores M's input n in an incorruptible way. Since we know in advance that M halts on input n but do not know how many steps it will take, the CRD will simulate S over

[10] It is sufficient to bound the number of decrements, rather than total instructions, since we may assume without loss of generality that M contains no "all-increment" cycles. (If it does then either these lines are not reachable or M enters an infinite loop.) Thus any infinite computation of M must decrement infinitely often.

and over again on the same input, each time incrementing the bound b, so that eventually b is large enough to allow a complete simulation of M on input n, assuming no errors.

If the original register machine M has m registers $r_1, \ldots, r_m$, then the simulating register machine S will have $m+4$ registers: $r_1, \ldots, r_m, r_{\text{in}}, r_{\text{in}'}, r_t, r_{t'}$. The first m registers behave exactly as in M, and the additional 4 registers will be used to help S maintain the input n and bound b.

The registers r_{in} and $r_{\text{in}'}$ are used to store the input value n. The input value n is given to S, and initially stored in r_{in}, and passed to $r_{\text{in}'}$ and r_1 before any other commands are executed. This additional step still allows the input n to be used in register r_1, while retaining the input value n in $r_{\text{in}'}$ for the next run of S. We want to enforce the invariant that, even if errors occur, $r_{\text{in}} + r_{\text{in}'} = n$, so that the value n can be restored to register r_{in} when S is restarted. To ensure that this invariant is maintained, the values of registers r_{in} and $r_{\text{in}'}$ change only with one of the two following sequences: $\texttt{dec(}r_{\text{in}}\texttt{,}k\texttt{);inc(}r_{\text{in}'}\texttt{)}$ or $\texttt{dec(}r_{\text{in}'}\texttt{,}k\texttt{);inc(}r_{\text{in}}\texttt{)}$ (it is crucial that the decrement comes before the increment, so that the increment is performed only if the decrement is successful). The only time the invariant $r_{\text{in}} + r_{\text{in}'} = n$ is not maintained is between the decrement and the subsequent increment. However, once the decrement is successfully performed, there is no possibility of error in the increment, so the invariant is guaranteed to be restored.

Registers r_t and $r_{t'}$ are used to record and bound the number of decrements performed by M, and their values are modified similarly to r_{in} and $r_{\text{in}'}$ so that S maintains the invariant $r_t + r_{t'} = b$ throughout each execution. The following lines are inserted before every decrement of M (assuming the decrement of M occurs on line $i+2$):

$$\begin{aligned} i&: \texttt{dec(}r_t\texttt{,}h\texttt{)} \\ i+1&: \texttt{inc(}r_{t'}\texttt{)} \end{aligned}$$

S uses lines h and $h+1$ added at the end to halt if M exceeds the decrement bound:

$$\begin{aligned} h&: \texttt{inc(}r_t\texttt{)} \\ h+1&: \texttt{reject} \end{aligned}$$

As S performs each decrement command in M, r_t is decremented and $r_{t'}$ is incremented. If the value of r_t is zero, then this means M has exceeded the allowable number of decrements, and computation halts (with a `reject`, chosen merely as a convention) immediately after incrementing r_t on line h. This final increment ensures that when the CRD simulates S again, the bound $b = r_t + r_{t'}$ will be 1 larger than it was during the previous run of S.

S is simulated multiple times by the CRD. To make it easier for the CRD to "reset" S by simply setting the current instruction to be the first line, S assumes that the work registers $r_2, \ldots, r_m$ are initially positive and must be set to 0. It sets r_1 to 0 before using r_{in} and $r_{\text{in}'}$ to initialize r_1 to n. In both pairs of registers $r_{\text{in}}, r_{\text{in}'}$ and $r_t, r_{t'}$, S sometimes needs to transfer the entire quantity stored in one register to the other. We describe below a "macro" for this operation, which we call `flush`. The following three commands will constitute the $\texttt{flush(}r_j\texttt{,}r_{j'}\texttt{)}$ command for any registers r_j and $r_{j'}$.

flush(r_j,$r_{j'}$)	i: dec(r_j,$i+3$)
	$i+1$: inc($r_{j'}$)
	$i+2$: goto(i)

While r_j has a non-zero value, it will be decremented and $r_{j'}$ will be incremented. To flush into more than one register, we can increment multiple registers in the place of line $i+1$ above. We denote this as `flush(`r_j`,`$r_{j'}$` and `$r_{j''}$`)`.

To set register r_j to 0, we use the following macro, denoted `empty(`r_j`)`.

empty(r_j)	i: dec(r_j,$i+2$)
	$i+1$: goto(i)

If an error occurs in a `flush`, the invariant on the sum of the two registers will still be maintained. If an error occurs on an `empty`, the effect will be that the register maintains a positive value. Both effects can be remedied by an error-free repeat of the same macro.

Combining all techniques described above, the first instructions of S when simulating M will be as follows:

```
      1: empty($r_1$)
      2: empty($r_2$)
         ⋮
    $m$: empty($r_m$)
$m+1$: flush($r_{\text{in}'}$,$r_{\text{in}}$)
$m+2$: flush($r_{\text{in}}$,$r_{\text{in}'}$ and $r_1$)
$m+3$: flush($r_{t'}$,$r_t$)
$m+4$: first line of $M$
```

The first m lines ensure that registers $r_1, \dots, r_m$ are initialized to zero. Lines $m+1$ and $m+2$ pass the input value n (stored as $r_{\text{in}} + r_{\text{in}'}$) to register r_1 (input register of M) while still saving the input value in $r_{\text{in}'}$ to be reused the next time S is run. Line $m+3$ passes the full value of $r_{t'}$ to r_t, so that the value of register r_t can be used to bound the number of jumps in the register machine. After line $m+3$, S executes the commands of M, starting with either the initial line of M (or decrementing r_t if the initial line of M is a decrement command as explained above).

Let d_n be the number of decrements M makes on input n without errors. If S is run without errors from an initial configuration (starting on line 1) with $r_{\text{in}} + r'_{\text{in}'} = n$ and $r_t + r_{t'} \geq d_n$, then it will successfully simulate M. Since the invariants on $r_{\text{in}} + r_{\text{in}'} = n$ and $r_t + r_{t'}$ are maintained throughout the computation — even in the face of errors — the only thing required to reset S after it halts is to set the current line to 1.

3.3 CRD Simulation of the Modified Register Machine

We now construct a CRD $\mathcal{D}$ to simulate S, while reducing the probability of an error each time an error could potentially occur. Besides the species described in Section 3.1, we introduce several new species: voting species N and Y, an "accuracy" species A, and four "clock" species C_1, C_2, C_3, and C_4. The accuracy and clock species will be used to reduce the probability of an error occurring. The initial state of $\mathcal{D}$ on input $n \in \mathbb{N}$ is $\{n\ R_{\text{in}}, 1\ L_1, 1\ Y, 1\ C_1\}$ — i.e., start with register $r_{\text{in}} = n$, initialize the register machine S at line 1, have initial vote "yes" (arbitrary choice), and start the "clock module" in the first stage.

Recall that the only source of error in the CRD simulation is from the decrement command dec(r_j,h), when R_j is present, but the jump reaction $L_i \to L_k$ occurs instead of the decrement reaction $L_i + R_j \to L_{i+1}$. This would cause the CRD to erroneously perform a jump when it should instead decrement register r_j. To decrease the probability of this occurring, we can slow down the jump reaction, thus decreasing the probability of it occurring before the decrement reaction when R_j is present.

The following reactions we call the "clock module," which implement a random walk that is biased in the reverse direction, so that C_4 is present sporadically, with the bias controlling the frequency of time C_4 is present. The count of "accuracy species" A controls the bias:

$$\begin{aligned} C_1 &\to C_2, & C_2 + A &\to C_1 + A,\\ C_2 &\to C_3, & C_3 + A &\to C_2 + A,\\ C_3 &\to C_4, & C_4 + A &\to C_3 + A. \end{aligned}$$

We modify the conditional jump reaction to require a molecule of C_4 as a reactant, as shown below. Increasing the count of species A decreases the expected time until the reaction $C_{i+1} + A \to C_i + A$ occurs, while leaving the expected time until reaction $C_i \to C_{i+1}$ constant. This has the effect that C_4 is present less frequently (hence the conditional jump reaction is slowed). Intuitively, with an ℓ-stage clock, if there are a molecules of A, the frequency of time that C_ℓ is present is less than $\frac{1}{a^{\ell-1}}$. A stage $\ell = 4$ clock is used to ensure that the error decreases quickly enough that with probability 1 a finite number of errors are ever made, and the last error occurs in finite expected time. A more complete analysis of the clock module is contained in the full version of this paper.

To use the clock module to make decrement instructions unlikely to incur errors, we change the CRD simulation of a decrement command to be the following two reactions:

dec(r_j,k)	$L_i + R_j \to L_{i+1} + A$
	$L_i + C_4 \to L_k + C_1 + A$

The jump reaction produces a C_1 molecule so the clock can be restarted for the next decrement command. Both reactions produce an additional A molecule to increase the accuracy of the next decrement command. As we continue to perform decrements, the random walk from C_1 to C_4 acquires a stronger reverse bias, so the conditional jump becomes less likely to occur erroneously.

The accept, reject, goto, and inc commands cannot result in errors for the CRD simulation, so we keep their reactions unchanged from Section 3.1.

After the CRD has completed the simulation and stabilizes, we would like the CRD to store the output of computation (either H_Y or H_N) and restart. At any time, there is precisely one molecule of either of the two voting species Y or N and none of the other, representing the CRD's "current vote." The vote is updated after each simulation of S, and S is reset to its initial configuration, via these reactions: $H_Y + Y \to L_1 + Y$, $H_Y + N \to L_1 + Y$, $H_N + Y \to L_1 + N$, and $H_N + N \to L_1 + N$.

4 Correctness of Simulation

Theorem 4.1. *Let $\mathcal{D}$ be the CRD described in Section 3, simulating an arbitrary register machine M deciding predicate $\phi : \mathbb{N} \to \{0,1\}$. Then $\mathcal{D}$ decides ϕ with probability 1. Furthermore, the expected time until $\mathcal{D}$ stabilizes to the correct answer is $O(t_n^8)$, where t_n is the number of steps M takes to halt on input n.*

Proof (sketch). Theorem 4.1 is proven formally in the full version of this paper. Intuitively, the proof is as follows. We use the clock module to reduce the probability of an incorrect decrement (a conditional jump on the instruction `dec(`r_j`,`k`)` to line k even when $r_j > 0$) each time a decrement occurs. For an error to occur, the two reactions $L_i + R_j \to L_{i+1} + A$ and $L_i + C_4 \to L_k + A + C_1$ compete. After d decrement instructions have occurred, the count of A is d. The stage $\ell = 4$ clock module ensures that the frequency of time in which C_4 is present is bounded above by $\frac{1}{d^3}$. Thus if an error is possible (i.e., if at least one R_j is present), then the probability of error on that step — i.e., that L_i encounters C_4 before R_j — is bounded by $\frac{1}{d^3}$. Since $\sum_{d=1}^{\infty} \frac{1}{d^3} < \infty$, by the Borel-Cantelli lemma, with probability 1, only finitely many errors occur. Our construction of S is designed so that it can recover from any finite number of errors. Once the final error occurs and S is reset to run from line 1, assuming the bound b on the number of decrements of M that S has is sufficiently large (and it grows by 1 on each subsequent simulation of S), then M will be simulated correctly from then on. After the first such simulation, the voter species will stabilize its vote forever. The bound on expected time until convergence follows from the fact that the error probability bound $\frac{1}{d^3}$ ensures a constant expected number of errors. (Borel-Cantelli follows even for an $\ell = 3$ stage clock which results in error probability bound $\frac{1}{d^2}$; but then the expected number of errors may diverge.) Hence the time is limited by the time required for the decrement bound b to grow large enough to simulate M all the way until it halts, roughly quadratic in the number of steps taken by M, followed by accounting for the slowdown of the CRD on each simulated register machine step due to the fact that *correct* conditional jumps due to the reaction $L_i + C_4 \to L_k + A + C_1$ get slower with each additional decrement. □

5 Δ_2^0 Predicates

In this section we extend the technique of Section 3 to show that every Δ_2^0 predicate is decidable with probability 1 by a CRD (Theorem 5.1). We also show the converse result (Theorem 5.2) that every predicate decidable with probability 1 by a CRD is in Δ_2^0. Theorems 5.1 and 5.2 give our main result, that probability 1 decidability by CRDs is exactly characterized by the class Δ_2^0.

Theorem 5.1. *Every Δ_2^0 predicate is decidable with probability 1 by a CRD.*

Proof. For every Δ_2^0 predicate $\phi : \mathbb{N} \to \{0,1\}$, there is a computable function $r : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ such that, for all $n \in \mathbb{N}$, $\lim_{t\to\infty} r(n,t) = \phi(n)$. Therefore there is a register machine M that computes r. As in Section 3, we first construct a register machine S that simulates M in a controlled fashion that ensures errors in simulation by a CRD will be handled gracefully.

It is routine to show that the definition of Δ_2^0 does not change if we require the register machine M computing $r(n,t)$ to halt in exactly $2t$ steps. Similar to Section 3, S uses a "timer" to simulate $M(n,t)$ for at most $2t$ steps. Unlike in Section 3, S decrements the timer after every step (not just the decrement steps) and the timer is incremented by 2 after each execution (in the previous construction, the timer is incremented by 1 and only if M exceeds the time bound). Note that no matter what errors occur, no execution can go for longer than $2t$ steps by the timer construction in Section 3. So S will dovetail the computation as before, running $M(n,1)$ for 2 steps, $M(n,2)$ for 4 steps, etc., and in between each execution of $M(n,t)$, update its voting species with the most recent answer.

As in the construction of Section 3, so long as the d'th decrement has error probability at most $\frac{1}{d^3}$, then by the Borel-Cantelli lemma, with probability 1 a finite number of errors occur. Errors in the CRD simulating S maintain the input without error and increment the timer value without error. Thus after the last error occurs, and after t is sufficiently large that $r(n,t) = \phi(n)$, the CRD will stop updating its voter, and the CRD's output will be correct. □

The next theorem shows that *only* Δ_2^0 predicates are decidable with probability 1 by a CRD.

Theorem 5.2. *Let the CRD $\mathcal{D}$ decide predicate $\phi : \mathbb{N} \to \{0,1\}$ with probability 1. Then $\phi \in \Delta_2^0$.*

Theorem 5.2 is proven in the full version of this paper. Intuitively, it follows because on input n, t, a Turing machine M can conduct a breadth-first search of the graph of reachable states and compute each of their probabilities of being the CRD's state after exactly t reactions. M uses this information to compute the probability that the output is 0 or 1 (or undefined) after exactly t reactions and outputs whichever bit has higher probability. Since $\mathcal{D}$ decides ϕ with probability 1, for all sufficiently large t, after t reactions the bit $\phi(n)$ has higher probability than its negation, hence is output by M.

Acknowledgements. We thank Shinnosuke Seki, Chris Thachuk, and Luca Cardelli for many useful and insightful discussions.

References

[1] Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: PODC 2006: Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, pp. 292–299. ACM Press, New York (2006)

[2] Aspnes, J., Ruppert, E.: An introduction to population protocols. Bulletin of the European Association for Theoretical Computer Science 93, 98–117 (2007)

[3] Brijder, R.: Output stability and semilinear sets in chemical reaction networks and deciders. In: Murata, S., Kobayashi, S. (eds.) DNA 2014. LNCS, vol. 8727, Springer, Heidelberg (2014)

[4] Cardelli, L.: Strand algebras for DNA computing. Natural Computing 10(1), 407–428 (2011)

[5] Chen, H.-L., Doty, D., Soloveichik, D.: Deterministic function computation with chemical reaction networks. Natural Computing (2013) (to appear), Preliminary version appeared in Stefanovic, D., Turberfield, A. (eds.) DNA 18. LNCS, vol. 7433, pp. 25–42. Springer, Heidelberg (2012)

[6] Chen, Y.-J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from DNA. Nature Nanotechnology 8(10), 755–762 (2013)

[7] Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses, pp. 543–584. Springer, Heidelberg (2009)

[8] Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1. Wiley (January 1968)

[9] Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)

[10] Karp, R.M., Miller, R.E.: Parallel program schemata. Journal of Computer and System Sciences 3(2), 147–195 (1969)

[11] Petri, C.A.: Communication with automata. Technical report, DTIC Document (1966)

[12] Rogers Jr., H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill Series in Higher Mathematics. McGraw-Hill Book Company, New York (1967)

[13] Soare, R.I.: Interactive computing and relativized computability. In: Copeland, B.J., Posy, C.J., Shagrir, O. (eds.) Computability: Turing, Gödel, Church, and Beyond, pp. 203–260. MIT Press, Cambridge (2013)

[14] Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing 7(4), 615–633 (2008)

[15] 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), Preliminary version appeared in Goel, A., Simmel, F.C., Sosík, P. (eds.) DNA 14. LNCS, vol. 5347, pp. 57–69. Springer, Heidelberg (2009)

[16] Volterra, V.: Variazioni e fluttuazioni del numero dindividui in specie animali conviventi. Mem. Acad. Lincei Roma 2, 31–113 (1926)

[17] 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)

The Computational Capability of Chemical Reaction Automata

Fumiya Okubo[1,⋆,⋆⋆] and Takashi Yokomori[2,⋆⋆⋆]

[1] Faculty of Arts and Science, Kyushu University 744 Motooka,
Nishi-ku, Fukuoka 819-0395, Japan
fokubo@artsci.kyushu-u.ac.jp

[2] Department of Mathematics,
Faculty of Education and Integrated Arts and Sciences,
Waseda University 1-6-1 Nishiwaseda, Shinjuku-ku, Tokyo 169-8050, Japan
yokomori@waseda.jp

Abstract. We propose a new computing model called *chemical reaction automata* (CRAs) as a simplified variant of reaction automata (RAs) studied in recent literature ([7–9]).

We show that CRAs in maximally parallel manner are computationally equivalent to Turing machines, while the computational power of CRAs in sequential manner coincides with that of the class of Petri nets, which is in marked contrast to the result that RAs (in both maximally parallel and sequential manners) have the computing power of Turing universality ([7, 8]). Intuitively, CRAs are defined as RAs without inhibitor functioning in each reaction, providing an offline model of computing by chemical reaction networks (CRNs).

Thus, the main results in this paper not only strengthen the previous result on Turing computability of RAs but also clarify the computing powers of inhibitors in RA computation.

Keywords: Chemical reaction automata, Reaction automata, Chemical reaction networks, Turing computability.

1 Introduction

For the last few decades, the notion of a multiset has frequently appeared and been investigated in many disciplines such as mathematics, computer science, linguistics, and so forth. In fact, during the last decade, a multiset has received more and more attention, particularly in the areas of biochemical computing

[⋆] Corresponding author.

[⋆⋆] The work of F. Okubo was in part supported by Grants-in-Aid for JSPS Fellows No.25.3528, Japan Society for the Promotion of Science.

[⋆⋆⋆] The work of T.Yokomori was in part supported by a Grant-in-Aid for Scientific Research on Innovative Areas "Molecular Robotics"(No.24104003) of The Ministry of Education, Culture, Sports, Science, and Technology, Japan, and by Waseda University grant for Special Research Projects: 2013B-063 and 2013C-159.

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 53–66, 2014.

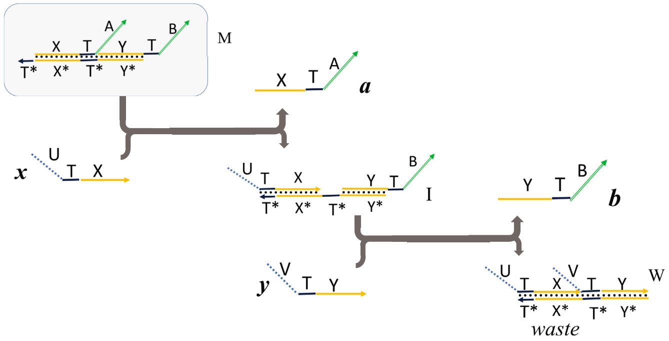


Fig. 1. The DSD implementation of a bimolecular reaction $X + Y \to A + B$: (A simplified modification of Fig.1 in [11] is depicted here.) Strands $\boldsymbol{x}$: $\mathtt{U} \cdot \mathtt{T} \cdot \mathtt{X}$ and $\boldsymbol{y}$: $\mathtt{V} \cdot \mathtt{T} \cdot \mathtt{Y}$ convey the information on the reactants X and Y, respectively, where $\mathtt{U}$ and $\mathtt{V}$ can be arbitrarily designed, while $\mathtt{T}$ is the toehold domain used universally in the implementation. The molecular structure M is designed uniquely to this reaction. For given $\boldsymbol{x}$ and $\boldsymbol{y}$, the series of the reactions $\boldsymbol{x} + \mathrm{M} \to \mathrm{I} + \boldsymbol{a}$ followed by $\boldsymbol{y} + \mathrm{I} \to \mathrm{W} + \boldsymbol{b}$ eventually produces $\boldsymbol{a}$ and $\boldsymbol{b}$ for the products A and B, respectively.

and molecular computing (e.g., [1]). There are two major research approaches in the area of biochemical computing: one is the formal computing systems based on multiset rewriting, and the other is the abstract models for chemical reaction machines.

Among many models proposed from the viewpoint of the former approach, inspired by the seminal work on reaction systems proposed in [4], *reaction automata* (RAs) have been first introduced in [8] as computing devices for accepting string languages. In the theory of RAs, a biochemical reaction is formulated as a triple $\mathbf{a} = (R_{\mathbf{a}}, I_{\mathbf{a}}, P_{\mathbf{a}})$, where $R_{\mathbf{a}}$ is a multiset of molecules called *reactants*, $I_{\mathbf{a}}$ is a set of molecules called *inhibitors*, and $P_{\mathbf{a}}$ is a multiset of molecules called *products*. Let T be a multiset of molecules, then the result of applying a reaction $\mathbf{a}$ to T, denoted by $Res_{\mathbf{a}}(T)$, is given by $T'(= T - R_{\mathbf{a}} + P_{\mathbf{a}})$ if $\mathbf{a}$ is enabled by T (i.e., if T completely includes $R_{\mathbf{a}}$ and excludes $I_{\mathbf{a}}$). A computation step in RAs is performed in such a way that each time receiving an input molecule a_{i+1} ($i = 0, 1, 2, \ldots$) provided from the environment, the system incorporates it to the current multiset T_i and changes its state into $T_{i+1}(= Res_{\mathbf{a}}(T_i \cup \{a_{i+1}\}))$. When starting with the initial state T_0 (of a multiset), an external input string $a_1 \cdots a_n$ is accepted if it induces a sequence of configurations T_is that ends in the final state predesignated. In RA computation process, two manners of applying reactions are considered: maximally parallel manner and sequential manner.

In the latter approach mentioned above, chemical reaction networks (CRNs) are formal models for molecular programming and defined as a finite set of chemical reactions with a multiset of signal molecules. An advantage of CRNs is that they can be in principle implemented by a molecular reaction primitive called DNA strand displacement systems (DSDs). For example, the DSD implementation of a bimolecular reaction $X + Y \to A + B$ is illustrated in Figure 1. Note that a bimolecular reaction $X + Y \to A + B$ in CRN formulation is represented by a reaction $\mathbf{a} = (XY, \emptyset, AB)$ in RA models. Thus, a formulation by CRNs is closely-related to RA models.

From these observations, in this paper we introduce a new computing model called *chemical reaction automata* (CRAs) that is a simplified variant of RAs where *no inhibitor* is involved in each reaction $\mathbf{a}$ in RAs. This notion of CRAs provides a new model for CRNs that is an *open system* for CRNs in which input molecules are provided in sequential manner from the environment. Thus, CRAs can be characterized as an *offline model* of computing device of (irreversible) CRNs. We primarily investigate the computational powers of CRAs, which may shed new light on the computational aspects of CRNs.

2 Preliminaries

2.1 Basic Definitions

We assume that the reader is familiar with the basic notions of formal language theory. For unexplained details, refer to [6]. Let V be a finite alphabet. For a set $U \subseteq V$, the cardinality of U is denoted by $|U|$. The set of all finite-length strings over V is denoted by V^*. The empty string is denoted by λ.

We use the basic notations and definitions regarding multisets that follow [1]. A multiset over an alphabet V is a mapping $\mu : V \to \mathbf{N}$, where $\mathbf{N}$ is the set of non-negative integers and for each $a \in V$, $\mu(a)$ represents the number of occurrences of a in the multiset μ. The set of all multisets over V is denoted by $V^{\#}$, including the empty multiset denoted by μ_λ, where $\mu_\lambda(a) = 0$ for all $a \in V$. We can represent the multiset μ by any permutation of the string $w = a_1^{\mu(a_1)} \cdots a_n^{\mu(a_n)}$. Conversely, with any string $x \in V^*$ one can associate the multiset $\mu_x : V \to \mathbf{N}$ defined by $\mu_x(a) = |x|_a$ for each $a \in V$. In this sense, we often identify a multiset μ with its string representation w_μ or any permutation of w_μ. Note that the string representation of μ_λ is λ, i.e., $w_{\mu_\lambda} = \lambda$. A usual set $U \subseteq V$ is regarded as a multiset μ_U such that $\mu_U(a) = 1$ if a is in U and $\mu_U(a) = 0$ otherwise. In particular, for each symbol $a \in V$, a multiset $\mu_{\{a\}}$ is often denoted by a itself.

For two multisets μ_1, μ_2 over V, we define one relation and two operations as follows:

- Inclusion : $\mu_1 \subseteq \mu_2$ iff $\mu_1(a) \leq \mu_2(a)$, for each $a \in V$,
- Sum : $(\mu_1 + \mu_2)(a) = \mu_1(a) + \mu_2(a)$, for each $a \in V$,
- Difference : $(\mu_1 - \mu_2)(a) = \mu_1(a) - \mu_2(a)$, for each $a \in V$ (for the case $\mu_2 \subseteq \mu_1$).

The sum for a family of multisets $M = \{\mu_i\}_{i \in I}$ is denoted by $\sum_{i \in I} \mu_i$. For a multiset μ and $n \in \mathbf{N}$, μ_n is defined by $\mu_n(a) = n \cdot \mu(a)$ for each $a \in V$. The weight of a multiset μ is $|\mu| = \sum_{a \in V} \mu(a)$.

For a symbol a, a new symbol a' is introduced as the primed version of a. Similarly, for a set S, $S' = \{a' \mid a \in S\}$ is introduced as the primed version of S.

Finally, for a multiset $x = a_1^{n_1} a_2^{n_2} \cdots a_k^{n_k} \in S^{\#}$, x' denotes the primed version of a multiset x such that $x' = a_1'^{n_1} a_2'^{n_2} \cdots a_k'^{n_k} \in S'^{\#}$.

2.2 Reaction Automata

Inspired by the works of reaction systems ([4]), we have introduced the notion of reaction automata in [8] by extending sets in each reaction to multisets. Here, we start by recalling basic notions concerning reaction automata.

Definition 1. For a set S, a *reaction rule* (or *reaction*) in S is a 3-tuple $\mathbf{a} = (R_{\mathbf{a}}, I_{\mathbf{a}}, P_{\mathbf{a}})$ of finite multisets, such that $R_{\mathbf{a}}, P_{\mathbf{a}} \in S^{\#}$, $I_{\mathbf{a}} \subseteq \mu_S$ and $R_{\mathbf{a}} \cap I_{\mathbf{a}} = \emptyset$.

The multisets $R_{\mathbf{a}}$, $I_{\mathbf{a}}$ and $P_{\mathbf{a}}$ are called the *reactant* of $\mathbf{a}$, the *inhibitor* of $\mathbf{a}$ and the *product* of $\mathbf{a}$, respectively. These notations are extended to a multiset of reactions as follows: For a set of reactions A and a multiset α over A,

$$R_\alpha = \sum_{\mathbf{a} \in A} R_{\mathbf{a}}^{\alpha(\mathbf{a})}, \ I_\alpha = \bigcup_{\mathbf{a} \subseteq \alpha} I_{\mathbf{a}}, \ P_\alpha = \sum_{\mathbf{a} \in A} P_{\mathbf{a}}^{\alpha(\mathbf{a})}.$$

In this paper, we consider two ways for applying reactions, i.e., sequential manner and maximally parallel manner. Intuitively, one reaction is applied to a multiset of objects in each step in sequential manner, while a multiset of reactions is exhaustively applied to a multiset in maximally parallel manner.

Definition 2. Let A be a set of reactions in S and $\alpha \in A^{\#}$ be a multiset of reactions over A. Then, for a finite multiset $T \in S^{\#}$, we say that
(1) α is *enabled by* T if $R_\alpha \subseteq T$ and $I_\alpha \cap T = \emptyset$,
(2) α is *enabled by* T *in sequential manner* if α is enabled by T with $|\alpha| = 1$.
(3) α is *enabled by* T *in maximally parallel manner* if there is no $\beta \in A^{\#}$ such that $\alpha \subset \beta$, and α and β are enabled by T.
(4) By $En_A^{sq}(T)$ and $En_A^{mp}(T)$, we denote the sets of all multisets of reactions $\alpha \in A^{\#}$ which are enabled by T in sequential manner and in maximally parallel manner, respectively.
(5) The *results of* A *on* T, denoted by $Res_A^X(T)$ with $X \in \{sq, mp\}$, is defined as follows:

$$Res_A^X(T) = \{T - R_\alpha + P_\alpha \,|\, \alpha \in En_A^X(T)\},$$

We note that $Res_A^{sq}(T) = \{T\}$ if $En_A^{sq}(T) = \emptyset$. Thus, if no multiset of reactions $\alpha \in A^{\#}$ is enabled by T, then T remains unchanged. In the case of maximaly parallel manner, $En_A^{mp}(T)$ at least contains one element. Specifically, if no multiset of reactions over A is enabled by T, then $En_A^{mp}(T) = \{\lambda\}$, and therefore $Res_A^{mp}(T) = \{T\}$.

Example 1. Let $S = \{a, b, c, d, e\}$ and consider the following set $A = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ of reactions in S:

$$\mathbf{a} = (b^2, a, c), \quad \mathbf{b} = (c, d, b), \quad \mathbf{c} = (bc, \emptyset, e).$$

Consider a finite multiset $T = \{b^4 cd\}$.

- In the case of sequential manner, $\mathbf{a}$ and $\mathbf{c}$ are enabled by T, while $\mathbf{b}$ is not enabled by T, because $I_{\mathbf{b}} \cap T \neq \emptyset$. Since $En_A^{sq}(T) = \{\mathbf{a}, \mathbf{c}\}$, we have

$$Res_A^{sq}(T) = \{b^2c^2d,\, b^3de\}.$$

- In the case of maximally parallel manner, $\alpha_2 = \mathbf{a}^2$ is enabled by T in maximally parallel manner, because no β with $\alpha_2 \subset \beta$ is enabled by T. Similarly, $\alpha_3 = \mathbf{ac}$ is also enabled by T in maximally parallel manner. Since $R_{\alpha_2} = b^4$, $P_{\alpha_2} = c^2$, $R_{\alpha_3} = b^3c$, $P_{\alpha_3} = ce$ and $En_A^{mp}(T) = \{\alpha_2, \alpha_3\}$, we have

$$Res_A^{mp}(T) = \{c^3d,\, bcde\}.$$

It is obvious from the above example that reactions are applied to a multiset in *nondeterministic way* both in sequential manner and maximally parallel manner.

We are now in a position to introduce the notion of reaction automata.

Definition 3. A *reaction automaton* (RA) $\mathcal{A}$ is a 5-tuple $\mathcal{A} = (S, \Sigma, A, D_0, f)$, where

- S is a finite set, called the *background set of* $\mathcal{A}$,
- $\Sigma (\subseteq S)$ is called the *input alphabet of* $\mathcal{A}$,
- A is a finite set of reactions in S,
- $D_0 \in S^\#$ is an *initial multiset*,
- $f \in S$ is a special symbol which indicates the final state.

Definition 4. Let $\mathcal{A} = (S, \Sigma, A, D_0, f)$ be an RA, $w = a_1 \cdots a_n \in \Sigma^*$ and $X \in \{sq, mp\}$. An *interactive process in* $\mathcal{A}$ *with input* w in X manner is an infinite sequence $\pi = D_0, \ldots, D_i, \ldots$, where

$$\begin{cases} D_{i+1} \in Res_A^X(a_{i+1} + D_i) & (\text{for } 0 \leq i \leq n-1), \text{ and} \\ D_{i+1} \in Res_A^X(D_i) & (\text{for all } i \geq n). \end{cases}$$

In order to represent an interactive process π, we also use the "arrow notation" for $\pi : D_0 \to^{a_1} D_1 \to^{a_2} D_2 \to^{a_3} \cdots \to^{a_{n-1}} D_{n-1} \to^{a_n} D_n \to D_{n+1} \to \cdots$. By $IP_X(\mathcal{A}, w)$ we denote the set of all interactive processes in $\mathcal{A}$ with input w in X manner.

Let $\Sigma_\lambda = \Sigma \cup \{\lambda\}$. When $a_i = \lambda$ for some several $1 \leq i \leq n$ in an input string $w = a_1 \cdots a_n$, an interactive process is said to be with λ-input mode. By $IP_X^\lambda(\mathcal{A}, w)$ we denote the set of all interactive processes in $\mathcal{A}$ with λ-input mode in X manner for the input $w \in \Sigma_\lambda^*$.

For an interactive process π in $\mathcal{A}$ with input w, if $En_A^X(D_m) = \emptyset$ for some $m \geq |w|$, then we have that $Res_A(D_m) = \{D_m\}$ and $D_m = D_{m+1} = \cdots$. In this case, considering the smallest m, we say that π *converges on* D_m (at the m-th step). If an interactive process π converges on D_m, then D_m is called the *converging state* of π and each D_i of π is omitted for $i \geq m+1$.

Definition 5. Let $\mathcal{A} = (S, \Sigma, A, D_0, f)$ be an RA and $X = \{sq, mp\}$. Then, the set of accepting interactive processes is defined as follows:

$$AIP_X^\lambda(\mathcal{A}, w) = \{\pi \in IP_X^\lambda(\mathcal{A}, w) \mid \ \pi \text{ converges on } D_m \text{ at the } m\text{-th step for some } m \geq |w| \text{ and } f \subseteq D_m\}.$$

The *language accepted by* $\mathcal{A}$ is defined as follows:

$$L_X^\lambda(\mathcal{A}) = \{w \in \Sigma_\lambda^* \mid AIP_X^\lambda(\mathcal{A}, w) \neq \emptyset\}.$$

Example 2. Let us consider a reaction automaton $\mathcal{A} = (S, \Sigma, A, D_0, f)$ defined as follows:

$$\begin{aligned}
&S = \{a, b, c, d, e, f\} \text{ with } \Sigma = \{a\},\\
&A = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_5, \mathbf{a}_6\}, \text{ where}\\
&\quad \mathbf{a}_1 = (a^2, \emptyset, b),\ \ \mathbf{a}_2 = (b^2, ac, c),\ \mathbf{a}_3 = (c^2, b, b),\\
&\quad \mathbf{a}_4 = (bd, ac, e),\ \mathbf{a}_5 = (cd, b, e),\ \ \mathbf{a}_6 = (e, abc, f),\\
&D_0 = d.
\end{aligned}$$

Let $w = aaaaaaaa \in S^*$ be the input string. Consider an interactive process π such that

$$\begin{aligned}
\pi : d \rightarrow^a ad \rightarrow^a bd \rightarrow^a abd \rightarrow^a b^2d \rightarrow^a ab^2d\\
\rightarrow^a b^3d \rightarrow^a ab^3d \rightarrow^a b^4d \rightarrow c^2d \rightarrow bd \rightarrow e \rightarrow f.
\end{aligned}$$

For instance, in the 9th step, since $\mathbf{a}_2^2 \in En_A^{mp}(b^4d)$, it holds that $c^2d \in Res_A^{mp}$ (b^4d). Hence, the step $b^4d \rightarrow c^2d$ is valid. Similarly, we can confirm that $\pi \in IP_{mp}^\lambda(\mathcal{A}, w)$. Since π converges on f, it holds that $\pi \in AIP_{mp}^\lambda(\mathcal{A}, w)$. Hence, it also holds that $w \in L(\mathcal{A})$.

We can also see that $L_{mp}^\lambda(\mathcal{A}) = \{a^{2^n} \mid n \geq 1\}$ which is context-sensitive.

2.3 Chemical Reaction Automata: CRAs

We define a *chemical reaction automaton* (CRA) as a special version of a reaction automaton. That is, a CRA is a 5-tuple (S, Σ, A, D_0, F), where each reaction in A is of the form $(R, \emptyset, P)$ (each reaction in CRA has no *inhibitor*), and F is a set of final multisets. For convenience, each reaction in A is denoted by $R \rightarrow P$. In an interactive process of CRA, if $En_A^X(D) = \emptyset$, $Res_A^X(D)$ is undefined. A language accepted by a CRA $\mathcal{A} = (S, \Sigma, A, D_0, F)$ is defined by

$$L_X^\lambda(\mathcal{A}) = \{w \in \Sigma_\lambda^* \mid \pi : D_0 \rightarrow^{a_1} D_1 \rightarrow^{a_2} \cdots \rightarrow^{a_n} D \in IP_X^\lambda(A, w),\ D \in F\}.$$

Remark. The acceptance condition of CRA computation is slightly different from that of RA computation. A CRA accepts a string if the final multiset *coincides* with an element of F, while an RA accepts a string if the final multiset *includes* a particular symbol f. This deference is significant to obtain our results in the paper.

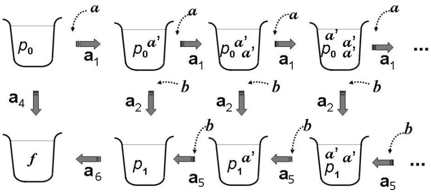


Fig. 2. A graphic illustration of interactive biochemical reaction processes for accepting strings in the language $L = \{a^n b^n \mid n \geq 0\}$ in terms of $\mathcal{A}$ in Example 3

Example 3. Let $\mathcal{A} = (S, \{a, b\}, A, p_0, \{f\})$ be a CRA which works in sequential manner, where

$$
\begin{aligned}
&S = \{a, b\} \cup \{a', p_0, p_1, f, \natural\}\\
&A = \{\mathbf{a_1} : p_0 + a \to p_0 + a', \mathbf{a_2} : p_0 + a' + b \to p_1,\ \mathbf{a_3} : p_0 + b \to \natural,\\
&\qquad \mathbf{a_4} : p_0 \to f, \qquad\qquad \mathbf{a_5} : p_1 + a' + b \to p_1,\ \mathbf{a_6} : p_1 \to f,\\
&\qquad \mathbf{a_7} : p_1 + a \to \natural,\ \ \mathbf{a_8} : f + a \to \natural,\ \mathbf{a_9} : f + a' \to \natural,\ \mathbf{a_{10}} : f + b \to \natural\}
\end{aligned}
$$

In the graphic drawing in Figure 2, each reaction $\mathbf{a}_i$ is applied to a multiset (a test tube) after receiving an input symbol (if any is provided) from the environment. In particular, applying $\mathbf{a}_4$ to $\{p_0\}$ leads to that the empty string is accepted by $\mathcal{A}$. It is seen, for example, that reactions $\mathbf{a}_1$ and $\mathbf{a}_2$ are enabled by the multiset $T = \{p_0, a', a'\}$ only when inputs a and b, respectively, are received, which result in producing $R_1 = \{p_0, a', a', a'\}$ and $R_2 = \{p_1, a'\}$, respectively. Thus, we have that $Res^{sq}_{\mathbf{a}_1}(T \cup \{a\}) = R_1$ and $Res^{sq}_{\mathbf{a}_2}(T \cup \{b\}) = R_2$. Once applying $\mathbf{a}_2$ has brought about a change of p_0 into p_1, $\mathcal{A}$ has no possibility of accepting further inputs a's. Otherwise any possible application of rules leads to introducing the symbol $\natural$, eventually resulting in the failure of computations. (Thus, in the construction of A, the symbol $\natural$ plays a role of *trapdoor* for unsuccessful computations.) It is easily seen that

$$L^{\lambda}_{sq}(\mathcal{A}) = \{a^n b^n \mid n \geq 0\}.$$

We remark that this language is also accepted by a RA in both maximally parallel and sequential manners, where the RA has fewer reaction rules than the above CRA has.

3 Main Results

3.1 The Computation Power of CRAs in Maximally Parallel Manner

In this section, we show that the computational power of CRAs working in maximally parallel manner is equivalent to that of Turing machines. To this aim, we utilize the notion of a multicounter machine: a variant of a Turing machine with a one-way read only input tape and several counters. It is known that a two-counter machine is equivalent to a Turing machine as a language accepting device ([5], [6]).

Multicounter Machines. For a non-negative integer $k \geq 1$, a k-counter machine is represented by a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F),$$

where, Q is a set of states, Σ is an alphabet of inputs, q_0 is an initial state, F is a set of final states, and δ is a set of transition rules defined by a mapping from $Q \times (\Sigma \cup \{\lambda\}) \times \{0,1\}^k$ into $Q \times \{0,+1\} \times \{-1,0,+1\}^k$. A configuration of M on an input $w\,(\in \Sigma^*)$ is given by a $(k+3)$-tuple $(q, w, i, c_1, \dots, c_k)$, where M is in a state q with the input head reading the i-th symbol of w, and $c_1, c_2, \dots, c_k (\in \mathbf{N})$ stored in the k counters. We write

$$(q, w, i, c_1, \dots, c_k) \vdash (p, w, i+d, c_1+d_1, \dots, c_k+d_k),$$

if a is the i-th symbol of w and $\delta(q, a, h(c_1), \dots, h(c_k))$ contains $(p, d, d_1, \dots, d_k)$, where $a = \lambda$ implies $d = 0$, and h is the logical mapping such that, for each $1 \leq j \leq k$,

$$h(c_j) = \begin{cases} 0 & \text{(if } c_j = 0\text{; the content of the } j\text{-th counter is zero)} \\ 1 & \text{(if } c_j \neq 0\text{; the content of the } j\text{-th counter is nonzero).} \end{cases}$$

(Note that, thus, M has the ability of distinguishing between zero and nonzero for the content c_j of each counter.)

The reflexive-transitive closure of $\vdash$ is written by $\vdash^*$. A language accepted by M is defined as

$$L(M) = \{w \in \Sigma^* \mid (q_0, w, 1, \overbrace{0, \dots, 0}^{k}) \vdash^* (f, w, i, c_1, \dots, c_k),\ f \in F\}.$$

We introduce the label for each transition rule in δ. A transition rule with a label is denoted as $r : (p, d, d_1, \dots, d_k) \in \delta(q, a, e_1, \dots, e_k)$. The set of labels of δ is denoted by $Lab(\delta)$.

The following lemma plays a crucial rule for our purpose.

Lemma 1. *For any k-counter machine M, there exists a CRA $\mathcal{A}$ such that $L(M) = L^{\lambda}_{mp}(\mathcal{A})$.*

Proof. For a given k-counter machine $M = (Q, \Sigma, \delta, q_0, F)$, let $T = Q \cup \Sigma \cup \{t_1, \dots, t_k\}$, where each t_i is a new symbol (corresponding to the i-th counter of M for $1 \leq i \leq k$). Consider the set of reaction rules Δ as follows: for each transition rule $r : (p, d, d_1, \dots, d_k) \in \delta(q, a, e_1, \dots, e_k)$, let

$$\begin{array}{ll} U_r = \{t_i \in T \mid 1 \leq i \leq k,\ e_i = 1\}, & X_r = \{t_i \in T \mid 1 \leq i \leq k,\ d_i = +1\}, \\ V_r = \{t_i \in T \mid 1 \leq i \leq k,\ e_i = 0\}, & Y_r = \{t_i \in T \mid 1 \leq i \leq k,\ d_i = -1\}. \end{array}$$

Then, we define

$$\Delta = \{r : (qa{+}U_r,\ V_r,\ p{+}U_r{+}X_r{-}Y_r) \mid r : (p, d, d_1, \ldots, d_k) \in \delta(q, a, e_1, \ldots, e_k)\}.$$

(Note that an inhibitor V_r of r in Δ plays a role of testing for zero for each counter of M[1].)

Using Δ, we construct a CRA $\mathcal{A} = (S, \Sigma, A, q_0\$, \{\#\})$, where

$$S = T \cup T' \cup T'' \cup \{\$, \#, \natural, f\} \cup \{\$_r \mid r \in \mathit{Lab}(\delta)\},$$
$$A = A' \cup \{a \to \natural \mid a \in \Sigma\} \cup \{a + \# \to \# \mid a \in S\},$$

and A' is composed of all the reaction rules from the union of the following three groups shown in Table 1 for *every* $r : (R, I, P)$ in Δ. We note that a set of reactions A is designed for simulating Δ *without inhibitor.*

Table 1. Reaction rules in A'

period 0	period 1	period 2	conditions
$R + \$ \to R' + \$_r$	$R' \to R''$	$R'' + \$_r \to P + \$$ (if $f \not\subseteq P$)	for $f \in F$,
	$\$_r + i \to \natural$	$R'' + \$_r \to P + \#$ (if $f \subseteq P$)	for $i \in I$

[The sketch of simulation of M]

Basically, one step of M is simulated by three steps of $\mathcal{A}$ (period 0, 1 and 2). On the $(i+1)$-th step of $\mathcal{A}$, only the reaction rules of period j can be used with $i \equiv j \bmod 3$. Reaction rules of each group in A' work as follows:

- Reaction rules $\{a \to \natural \mid a \in \Sigma\}$ are used to control the timing of an input. If the symbol a is inputted in the timing when rules in period 1 or period 2 are used, then the reaction $a \to \natural$ must be used in maximally parallel manner. This means that $\mathcal{A}$ cannot reach the final multiset. Hence, in an accepting interactive process, each symbol in Σ may be inputted only in period 0.
- Using reaction rules in A', one transition of M is simulated by three steps of $\mathcal{A}$ (period 0, period 1, period 2), where the "zero-test" of multicounter machine is performed by taking advantage of maximally parallel manner. If a reaction rule r is applied in a wrong way, then the reaction rule $\$_r + i \to \natural$ must be used in the next step. This means that $\mathcal{A}$ cannot reach the final multiset.
- The reaction rule $R'' + \$_r \to P + \#$ is used when M is in the final step. After this step, by using reaction rules $\{a + \# \to \# \mid a \in S\}$, the final multiset $\#$ is derived in $\mathcal{A}$, if M reaches the final state.

The simulation proceeds as follows: Let $(q, w, j, c_1, \ldots, c_k) \vdash (p, w, j', c'_1, \ldots, c'_k)$ be the $(i + 1)$-th step of M by the transition $r : (p, d, d_1, \ldots, d_k) \in$

[1] This trick has been used for simulating M by an RA (with inhibitors) in sequential manner in [7].

$\delta(q, a, e_1, \ldots, e_k)$ corresponding to the reaction $r : (R, I, P)$ in $\mathcal{A}$. Then, it holds that $D_{3i} \to D_{3i+1} \to D_{3i+2} \to D_{3(i+1)}$ in $\mathcal{A}$, where

$$\begin{aligned} D_{3i} &= q + t_1^{c_1} t_2^{c_2} \cdots t_k^{c_k} + \$, \\ D_{3i+1} &= (D_{3i} - R) + R' + \$_r, \\ D_{3i+2} &= (D_{3i} - R) + R'' + \$_r, \\ D_{3(i+1)} &= \begin{cases} D_{3i} - R + P + \$ = p + t_1^{c_1'} t_2^{c_2'} \cdots t_k^{c_k'} + \$, & (\text{if } p \notin F) \\ p + t_1^{c_1'} t_2^{c_2'} \cdots t_k^{c_k'} + \# & (\text{if } p \in F) \end{cases} \end{aligned}$$

This process is realized by the reaction rules in A'. Note that when the multi-counter machine M reads a_i in $(i+1)$-th step, a_i is inputted into D_{3i} of period 0 (in $(3i+1)$-th step) in $\mathcal{A}$.

After appearing #, only reaction rules in $\{a + \# \to \# \mid a \in S\}$ can be used, which deletes all objects but #. As a result of computation of $\mathcal{A}$, if M accepts w, then only # remains in the multiset of $\mathcal{A}$, hence $\mathcal{A}$ accepts w. Conversely, from the manner of constructing A, it is easily seen that $\mathcal{A}$ can accept no input other than w that is accepted by M. Thus, it holds that $L(M) = L^{\lambda}_{mp}(\mathcal{A})$. □

Note. As in mentioned in Remark in section 2.3, the modification of the acceptance condition in RA is essentially required for CRA in this proof. If the acceptance condition of CRA remains unchanged, then a multiset ♮# may be a final multiset, since it contains #. However, this implies that a symbol is input into the system in period 1 or period 2, leading to the failure of simulating a k-counter machine.

Thus, by Lemma 1, we have already proved the following:

Theorem 1. *The computational power of CRAs with λ-input mode in maximally parallel manner is equivalent to that of Turing machines.*

Corollary 1. *Every recursively enumerable language is accepted by a CRA with λ-input mode in maximally parallel manner.*

3.2 The Computation Power of CRAs in Sequential Manner

We now consider a naive question on the computing powers of CRAs whether or not there exists a real gap between working in maximally parallel manner and in sequential manner. We shall show that the class of CRAs in sequential manner is computationally less powerful than the other. In fact, it is shown that the class of CRAs in sequential manner is computationally equivalent to the class of Petri nets (of a certain type). Our question is thus solved as a corollary of the next theorem.

Petri Nets. A Petri net is a 4-tuple $N = (P, T, A, \mu_0)$, where P is a finite alphabet of places, T is a finite alphabet of transitions, $A : T \to P^{\#} \times P^{\#}$ is an arc function and $\mu_0 \in P^{\#}$ is an initial marking. For t in T, $A(t) = (\alpha, \beta)$ is

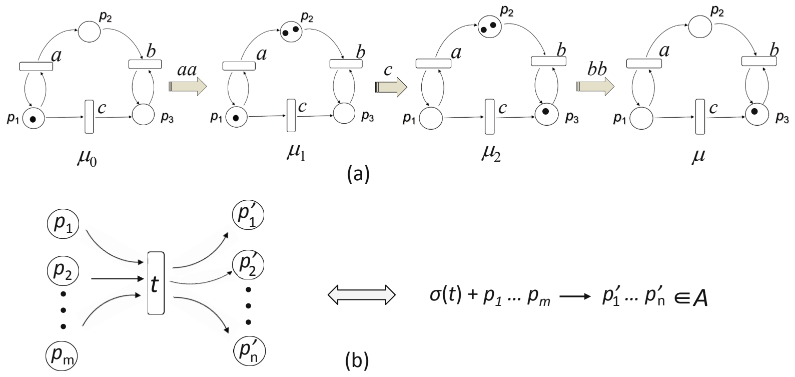


Fig. 3. (a). A graphic drawing of a transition sequence in Petri net N. **(b)**. A correspondence between transitions on t in N and a reaction rule involving t in A. (Note that p_i and p'_j are not necessarily distinct each other.)

also denoted by $\alpha \rightarrow^t \beta$ (in A). When a transition t with $\alpha \rightarrow^t \beta$ is applied to a multiset μ in $P^\#$ with $\alpha \subseteq \mu$, we write it as $\mu \Rightarrow^t \mu - \alpha + \beta$. For $t \in T$ and $w \in T^*$, $\mu \Rightarrow^{tw} \mu'$ is defined by $\mu \Rightarrow^\lambda \mu$ and $\mu \Rightarrow^t \mu'' \Rightarrow^w \mu'$.

There are several ways to define a language generated by a Petri net in [10]. Here, we adopt the following definition. A Petri net system is a 6-tuple $NS = (P, T, A, \mu_0, \sigma, F)$, where (P, T, A, μ_0) is a Petri net, $\sigma : T \rightarrow \Sigma_\lambda$ is a labeling of transitions, and F is a finite set of final markings. Then, a language generated by a Petri net system NS is defined as

$$L(NS) = \{\sigma(w) \in \Sigma^* \mid w \in T^*, \mu_0 \Rightarrow^w \mu, \mu \in F\}.$$

Example 4. Consider a Petri net system $NS = (\{p_1, p_2, p_3\}, \{a, b, c\}, A, p_1, \sigma, \{p_3\})$, where $A(a) = (p_1, p_1p_2)$, $A(b) = (p_2p_3, p_3)$, $A(c) = (p_1, p_3)$, $\sigma(a) = a, \sigma(b) = b$, and $\sigma(c) = \lambda$. Figure 3 (a) illustrates an example of the computation process in NS: $\mu_0(= p_0) \Rightarrow^{aa} \mu_1 \Rightarrow^c \mu_2 \Rightarrow^{bb} \mu$. Thus, the sequence of transitions $w = aacbb$ leads μ_0 to μ. Since $\mu = p_3$ is designated as a final marking, a string $\sigma(w) = aabb$ is in $L(NS)$. In general, N together with the labeling function σ and the final marking p_3 generates the language $\{a^n b^n \mid n \geq 0\}$. Thus, we have that $L(NS) = \{a^n b^n \mid n \geq 0\}$.

Theorem 2. *A language L is generated by a Petri net system if and only if L is accepted by a CRA in sequential manner.*

Proof. Here, we only outline the proof for both directions.

(only if part) For a Petri net system $NS = (P, T, A, \mu_0, \sigma, F)$, construct a CRA $\mathcal{A} = (P \cup \sigma(T), \sigma(T), A, \mu_0, F)$, where $\sigma(t) + \alpha \rightarrow \beta \in A$ if $\alpha \rightarrow^t \beta$. Let $w = a_1 \cdots a_n$ in T^*, then it holds that $\mu_0 \Rightarrow^w \mu$ if and only if $\pi : \mu_0 \rightarrow^{\sigma(a_1)} \cdots \rightarrow^{\sigma(a_n)} \mu \in IP^\lambda_{sq}(\mathcal{A}, \sigma(w))$. Thus, we have $L(NS) = L^\lambda_{sq}(\mathcal{A})$.

(if part) For a CRA $\mathcal{A} = (S, \Sigma, A, D_0, F)$, construct a Petri net system $NS = (S, T, A', D_0, \sigma, F)$ as follows:

$$T = \{r_a \mid a \in \Sigma, a \subseteq R, r : R \to P \in A\} \cup \{r'_a \mid a \in \Sigma, a \not\subseteq R, r : R \to P \in A\}$$
$$\cup \{r \mid r : R \to P\}$$

A' and σ are constructed as below:

- (i) $R - a \to^{r_a} P$ and $\sigma(r_a) = a$, for all r_a in T
- (ii) $R \to^{r'_a} P + a$ and $\sigma(r'_a) = a$, for all r'_ain T
- (iii) $R \to^{r} P$ and $\sigma(r) = \lambda$, for all r in T.

Each rule in (i) is used in the case when the input $\sigma(r_a)$ is consumed by the corresponding rule r of A, while each rule in (ii) is used in the case when the input $\sigma(r'_a)$ is not consumed by the corresponding rule r of A. Each rule in (iii) is used in the case when the corresponding rule r of A is used in the λ-input mode.

Then, for $w = a_1 \cdots a_n$ in T^*, it holds that $\pi : D_0 \to^{\sigma(a_1)} \cdots \to^{\sigma(a_n)} D \in IP^{\lambda}_{sq}(\mathcal{A}, \sigma(w))$ if and only if $D_0 \Rightarrow^{w} D$. In this way, $L^{\lambda}_{sq}(\mathcal{A}) = L(NS)$ is obtained. (Note that the one-to-one correspondence between a graph structure of transitions in N and a reaction rule in A is illustrated in Figure 3 (b).) □

It is known that the class of Petri net languages is strictly included in the class of context-sensitive languages and is incomparable to the class of context-free languages ([10]). Hence, we have:

Corollary 2. *The computational power of CRAs with λ-input mode in sequential manner is less powerful than that of CRAs with λ-input mode in maximally parallel manner.*

4 Discussion

4.1 Related Work

Apart from our previous works on reaction automata (RAs) (in [7–9]), here we only refer to the following two themes of work that share subjects and issues with this paper.

- **Turing computation and CRNs/DSDs**: Chemical reaction networks (CRNs) have been introduced as a descriptive formal language to analyze the behaviors of chemical reactions in nature, while DNA strand displacement systems (DSDs) is known as a powerful method for implementing CRNs.

  In the stochastic setting of CRNs, Soloveichik et al. ([12]) shows that CRNs can perform Turing-universal computation with arbitrary small error probability, in which register machines, as well as Turing machines (TMs), are used to prove the result. By providing an energy efficient molecular implementation of stack machines, Qian et al. ([11]) shows that Turing computability

can be achieved by using DSDs together with polymers. It was shown in [14] that reversible CRNs (and DSDs) can simulate in a space and energy efficient manner polynomially space-bounded TM computation. The latter work also proves that the reachability problem for CRNs is PSPACE-complete for some restricted classes of CRNs.
- **Multiset-based computing models**: Suzuki et al. ([13]) propose an abstract chemical reaction model based on multiset rewriting that first attracted our attention. With the CRN framework called ARMS, they primarily investigates the modeling (and simulation) capability of ARMS from the artificial life point of view. Among many in the literature on language acceptors based on multiset rewriting, a variant of P systems called P automata has been intensively investigated (e.g., [2],[3]). In a P automaton, a configuration comprises a tuple of multisets each of which consists of objects from each membrane region. On receiving an input (a multiset) from the environment it changes its configuration by making region-wise applications of the equipped rules. In this sense, RAs (and CRAs) may also be regarded as a simplified variant of P automata *without* any membrane structure.

4.2 Conclusion and Future Work

We have investigated the computational capability of a new class of automata called chemical reaction automata (CRAs) that was a simplified version of the class of reaction automata (RAs) in [7–9]. CRAs are *offline* computational models based on the chemical programming languages known as chemical reaction networks (CRNs) in which an input sequence (i.e., a sequence of molecular species) is provided one by one from the external environment to the solution (i.e., a multiset of molecules) during a nondeterministic computation process. Hence, the class of CRAs offers a good device of models to explore the computational aspects of CRNs. In this paper, we have shown the following:

1. CRAs with λ-input mode in *maximally parallel* manner can achieve the Turing universal computability. Further, it was shown that, in turn,
2. the computational power of CRAs with λ-input mode in *sequential* manner exactly coincides with that of the class of Petri Nets, therefore, strictly less powerful than Turing machines. From these results, it turned out that
3. within the RA computing models, the use of inhibitors of reaction rules makes *no* effect on the power of computing in maximally parallel manner, while it does in sequence manner.
4. Considering that the class of RAs with λ-input mode in sequential manner can achieve the Turing computability (Theorem 3.1 in [7]), it is strongly suggested that as for the computational power, there exists a trade-off relation between the role of inhibitors in reaction rules and that of maximally parallelism in computation process of RAs.

There remain many subjects to be investigated along the research direction suggested by CRAs and the related. Among others,

- it is of importance to clarify the relationships between subclasses of CRAs and others defined by computing devices based on the multiset rewriting, such as a variety of P-automata and their variants of dP automata ([3]).
- It is also valuable to explore the computational power of deterministic CRAs. Specifically, it is open whether deterministic CRAs in maximally parallel manner are Turing universal.
- In the contexts of computational complexity of CRAs subclasses, no effort has been made yet for investigating the *time* complexity of any subclass.
- CRAs can be modified in an obvious manner to define a class of transducers. Such a variant of CRAs that can compute functions (rather than languages) remains left open to be studied.

References

1. Calude, C.S., Pun, G., Rozenberg, G., Salomaa, A. (eds.): Multiset Processing. LNCS, vol. 2235. Springer, Heidelberg (2001)
2. Csuhaj-Varju, E., Ibarra, O.H., Vaszil, G.: On the computational complexity of P automata. Natural Computing 5, 109–126 (2006)
3. Csuhaj-Varju, E., Vaszil, G.: P automata. In: The Oxford Handbook of Membrane Computing, pp. 145–167 (2010)
4. Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fundamenta Informaticae 75, 263–280 (2007)
5. Fischer, P.C.: Turing Machines with Restricted Memory Access. Inform. and Contr. 9(4), 364–379 (1966)
6. Hopcroft, J.E., Motwani, T., Ullman, J.D.: Introduction to automata theory, language and computation, 2nd edn. Addison-Wesley (2003)
7. Okubo, F.: Reaction automata working in sequential manner. RAIRO Theoretical Informatics and Applications 48, 23–38 (2014)
8. Okubo, F., Kobayashi, S., Yokomori, T.: Reaction automata. Theoretical Computer Science 429, 247–257 (2012)
9. Okubo, F., Kobayashi, S., Yokomori, T.: On the properties of language classes defined by bounded reaction automata. Theoretical Computer Science 454, 206–221 (2012)
10. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, Englewood Cliffs (1981)
11. Qian, L., Soloveichik, D., Winfree, E.: Efficient Turing-Universal Computation with DNA Polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
12. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing 7(4), 615–633 (2008)
13. Suzuki, Y., Fujiwara, Y., Takabayashi, J., Tanaka, H.: Artificial Life Applications of a Class of P Systems: Abstract Rewriting Systems on Multisets. In: Calude, C.S., Pun, G., Rozenberg, G., Salomaa, A. (eds.) Multiset Processing. LNCS, vol. 2235, pp. 299–346. Springer, Heidelberg (2001)
14. Thachuk, C., Condon, A.: Space and energy efficient computation with DNA strand displacement systems. In: Stefanovic, D., Turberfield, A. (eds.) DNA 18. LNCS, vol. 7433, pp. 135–149. Springer, Heidelberg (2012)

Emulating Cellular Automata in Chemical Reaction-Diffusion Networks

Dominic Scalise[1] and Rebecca Schulman[1,2]

[1] Chemical and Biomolecular Engineering, Johns Hopkins University
[2] Computer Science, Johns Hopkins University

Abstract. Chemical reactions and diffusion can produce a wide variety of static or transient spatial patterns in the concentrations of chemical species. Little is known, however, about what dynamical patterns of concentrations can be reliably *programmed* into such reaction-diffusion systems. Here we show that given simple, periodic inputs, chemical reactions and diffusion can reliably emulate the dynamics of a deterministic cellular automaton, and can therefore be programmed to produce a wide range of complex, discrete dynamics. We describe a modular reaction-diffusion program that orchestrates each of the fundamental operations of a cellular automaton: storage of cell state, communication between neighboring cells, and calculation of cells' subsequent states. Starting from a pattern that encodes an automaton's initial state, the concentration of a "state" species evolves in space and time according to the automaton's specified rules. To show that the reaction-diffusion program we describe produces the target dynamics, we simulate the reaction-diffusion network for two simple 1-dimensional cellular automata using coupled partial differential equations. Reaction-diffusion based cellular automata could potentially be built *in vitro* using networks of DNA molecules that interact *via* branch migration processes and could in principle perform universal computation, storing their state as a pattern of molecular concentrations, or deliver spatiotemporal instructions encoded in concentrations to direct the behavior of intelligent materials.

1 Introduction

A fundamental question in materials design is how we might program materials to sense and respond to dynamic signals across time and space. Biological materials routinely exhibit this capacity, as cells and tissues sense and respond to a complex array of spatial and temporal cues. For example, during chemotaxis, many cells can detect gradients of chemoattractants and move in the direction of increasing chemoattractant concentration. In a mechanism like chemotaxis[1–3], cells use spatiotemporal chemical reaction networks to process information collected by distributed chemical sensors to decide on and execute responses to changing environmental conditions. In this paper we discuss the design of analogous synthetic chemical reaction networks that have robust, programmable spatiotemporal dynamics. The ability to engineer such systems could have wide-ranging applications for the design of smart, responsive, programmable materials.

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 67–83, 2014.

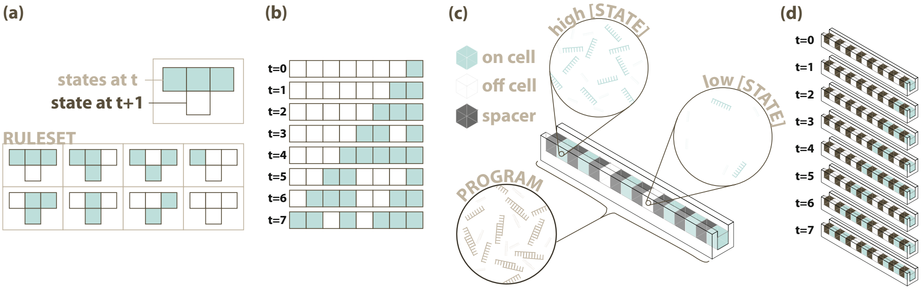


Fig. 1. ***A cellular automaton*** consists of a lattice of cells. At a given time, each cell is in one of a finite number of states, shown by color (blue or white). Cell states change over time as the result of the application of local rules - finite functions that take as inputs the states of the current cell and a finite set of neighbors and produce the cells' new state as output. Here we consider a 1-dimensional automaton where each cell is either *on* or *off*, and where update rules take the cell's own state and those of its left and right neighbors as inputs. (a) An example rule set. (b) Example dynamics for the rule set in (a). (c) Schematic of the chemical reaction-diffusion cellular automaton described in this paper. A 1-dimensional channel contains cells separated by spacers. The state in each cell is encoded by either a *high* or *low* concentration of a 'state' species within that cell. Spacers between cells, which do not contain any state information, are shown in black. During the computation, the program and state species react and diffuse. This reaction-diffusion process maintains and updates cell state according to the rules of the desired cellular automaton. (d) Target dynamics of the state species for the example cellular automaton rule in (a).

To design a generic set of mechanisms that can process a wide range of input signals and invoke a wide range of responses, we consider a framework for distributed spatial computation that has been studied extensively – the cellular automaton (Fig. 1). A cellular automaton (CA) is a model of computation consisting of a rectangular lattice of domains, or 'cells'. At a given time a cell can be in one of a finite number of states, such as an *on* or *off*. In a synchronous CA, cells update their state once per time step based on their current state and the current states of a finite set of nearby cells. Although each cell update is relatively simple, groups of cells can together perform elaborate spatial computation. CA can execute any computable algorithm, a trait known as universal computation [4–7]. Specific automata also exist that can programmably construct any structure[8, 9], self-replicate[8–10], mutate and evolve[11].

In this paper we propose a strategy for building synchronous CA using chemical reaction-diffusion networks[1]. We begin by breaking down CA into their fundamental operations: storage of cell states, communication between nearby cells, and calculation of new cell states. We demonstrate how existing chemical computing mechanisms could implement these operations. We then combine these chemical mechanisms to emulate two specific automata, known as 'Rule 110' and 'Rule 60'. These chemical CA can be viewed as a proof of concept that synthetic materials could sense signals across space and time and execute a broad class of dynamic programmed responses.

[1] A more complex design for an asynchronous CA can be constructed along similar lines.

2 Background: Reaction-Diffusion Processes for Computation

Reaction-diffusion (RD) networks are sets of chemically reactive species that diffuse within a continuous substrate. In contrast to a well-mixed chemical reaction system, reaction-diffusion (RD) networks can produce spatial patterns, where some species are more abundant in some parts of the substrate and less abundant in others. The interplay of reactions and diffusion can lead to sustained spatial patterns and even the emergence of patterns from a homogeneous initial substrate which experiences transient concentration fluctuations [12]. Transient waves within these patterns can emulate Turing machines and perform basic computations[13–15].

Recently it has been shown that arbitrary chemical reaction networks can be readily designed and implemented *in vitro* using short strands of synthetic DNA [16, 17]. Because DNA binding under many conditions is largely determined by the Watson-Crick sequence complementarity (A-T, G-C), reactive species can be designed to minimize unintended crosstalk between species that should not interact. These techniques separate the design of new reaction networks from the discovery of naturally occurring chemicals that perform the intended reactions. In support of this idea, large reaction networks involving up to 130 different sequences of DNA have been demonstrated *in vitro* without substantial crosstalk[18], and have been used to implement Boolean logic[18–20]. Further, the rates of the emulated reactions can be controlled[17].

It also appears plausible to extend this mechanism of DNA reaction design to the design of large reaction-diffusion networks, as the diffusion rates of different DNA strands can be programmed. Because the diffusion coefficient of DNA scales polynomially with the length of the strand[21], the diffusion rate of each species in a DNA reaction network can be independently tuned by adding or removing bases from a sequence, and such changes can be done so that the reactive propensity of a species is largely unchanged. Further, within a polymer gel, species attached to the gel substrate do not diffuse, but can continue to react. Together, the capacities to design arbitrary chemical reactions and tune the diffusion coefficient of each species in principle enable us to implement *de novo* simple RD networks that perform pattern transformations[22, 23].Here we ask how we might design an RD network that could be implemented by DNA molecules, given what is known about designing DNA-based reactions and diffusion processes. To focus on this question, here we ignore experimental nonidealities and the challenges of building large molecular networks, including unintended crosstalk between species.

By designing RD network modules that perform simple, predictable, repeatable transformations to a pattern of chemical concentrations, circuits of modules can be combined to perform elaborate patterning operations[24]. Pattern transformation modules take specific species as inputs, perform reactions potentially involving some intermediate species within the module, and produce an output species (Fig. 2). Modules can be connected together with the output of upstream modules serving as the input to downstream modules. If these modules are designed such that the intermediate species of one module do not react with the intermediates of other modules, then many modules can operate simultaneously in the same substrate without interfering with each other. Further, by imposing the design requirement that modules must not significantly deplete (or "load") the concentrations of their inputs, it is possible to ensure that a module's reactions affect only other modules that lie downstream within the network. Thus, modules

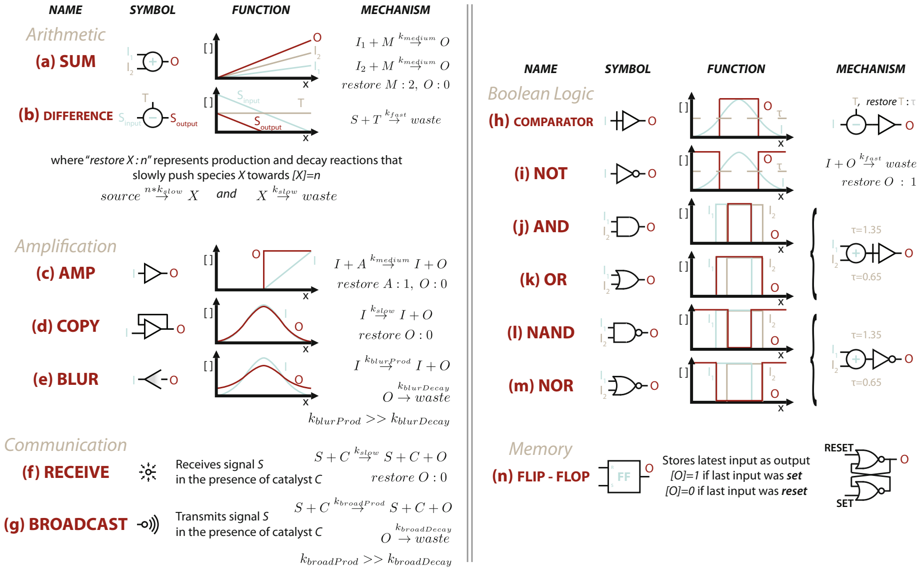

NAME
SYMBOL
FUNCTION
MECHANISM
Arithmetic
(a) SUM
$I_1 + M \xrightarrow{k_{medium}} O$
$I_2 + M \xrightarrow{k_{medium}} O$
$restore\ M : 2,\ O : 0$
(b) DIFFERENCE
$S + T \xrightarrow{k_{fast}} waste$
where "restore X : n" represents production and decay reactions that slowly push species X towards [X]=n
$source \xrightarrow{n*k_{slow}} X$ and $X \xrightarrow{k_{slow}} waste$
Amplification
(c) AMP
$I + A \xrightarrow{k_{medium}} I + O$
$restore\ A : 1,\ O : 0$
(d) COPY
$I \xrightarrow{k_{slow}} I + O$
$restore\ O : 0$
(e) BLUR
$I \xrightarrow{k_{blurProd}} I + O$
$O \xrightarrow{k_{blurDecay}} waste$
$k_{blurProd} >> k_{blurDecay}$
Communication
(f) RECEIVE
Receives signal S in the presence of catalyst C
$S + C \xrightarrow{k_{slow}} S + C + O$
$restore\ O : 0$
(g) BROADCAST
Transmits signal S in the presence of catalyst C
$S + C \xrightarrow{k_{broadProd}} S + C + O$
$O \xrightarrow{k_{broadDecay}} waste$
$k_{broadProd} >> k_{broadDecay}$
Boolean Logic
(h) COMPARATOR
T, restore T : τ
(i) NOT
$I + O \xrightarrow{k_{fast}} waste$
$restore\ O\ :\ 1$
(j) AND
(k) OR
τ=1.35
τ=0.65
(l) NAND
(m) NOR
τ=1.35
τ=0.65
Memory
(n) FLIP - FLOP
FF
Stores latest input as output
[O]=1 if last input was set
[O]=0 if last input was reset
RESET
SET

Fig. 2. ***Reaction-Diffusion Modules.*** Each module is a set of simpler modules or of abstract chemical reactions that could in principle be emulated *in vitro* using a DNA strand-displacement network. In an emulation of these reactions using DNA strand displacement processes, species are represented as DNA strands or hybridized complexes of strands with particular sequences. Other strands or complexes are also required for the emulation, and act as "intermediates" in the reaction process [16, 24]. More details on these modules and their operation can be found in [24].

can be added one at a time to a system such that each addition of a module results in a simple, predictable change to the patterning process.

Here we extend existing pattern transformation techniques to emulate a discrete, synchronous, 1-dimensional CA, generating spatial patterns of chemical concentrations with controlled dynamics. We design a network of reaction-diffusion pattern transformation modules (defined in detail in Fig. 2) in combination with a simple, static initial pattern and an external "clock" signal whose concentrations change periodically. This network forms the target CA structure, and controllably transforms the state of that structure over time. In principle, our abstract chemical reaction-diffusion network could be translated into a DNA strand displacement network for *in vitro* implementation.

One challenge in the design of pattern transformations is that the second law of dynamics implies that without the continual input of energy, purely diffusive patterns are unstable and tend to become well mixed over time. Thus, to prevent spatial patterns of soluble molecules from dissipating, reaction-diffusion networks will require a constant energy flux. One way to achieve this flux is to develop reactions that slowly release and degrade high-energy species. These reactions produce a sustained flux of molecules in the environment, and maintain a pattern such that only sporadic replenishment of some high-energy precursors are required to sustain the pattern formation process. *Production* reactions take the form $source \rightarrow species$, and continuously produce reactants that are depleted by converting a high-energy precursor into the desired species. Degradation reactions take the form $species \rightarrow waste$, and convert species that are produced into low-energy waste to stabilize the system.

3 Anatomy of Our Reaction-Diffusion Cellular Automaton

Reaction-diffusion systems emulating a CA must be able to store the current state of the system as a tape or grid of cells and execute the update rules as a function of the states of the cell and the cell's left and right neighbors (Fig. 3). For the class of CA we consider here, the state of each cell is either *on* or *off*.

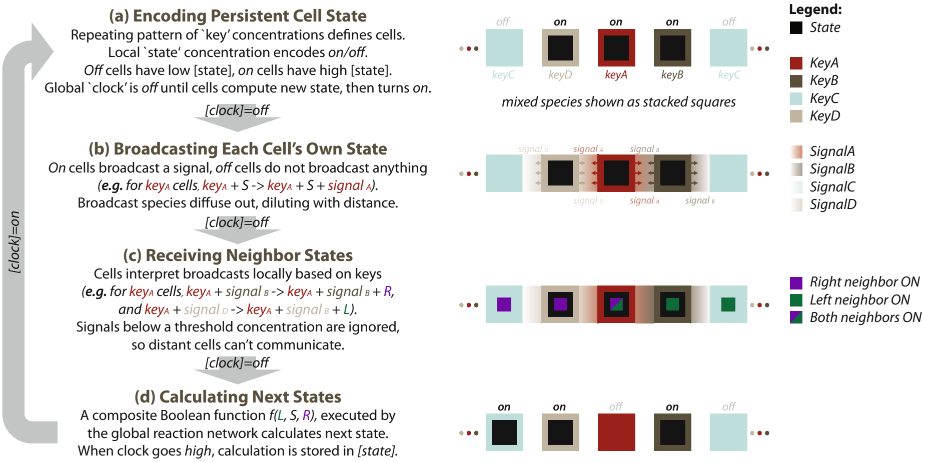

(a) Encoding Persistent Cell State
Repeating pattern of `key' concentrations defines cells.
Local `state' concentration encodes on/off.
Off cells have low [state], on cells have high [state].
Global `clock' is off until cells compute new state, then turns on.
[clock]=off
(b) Broadcasting Each Cell's Own State
On cells broadcast a signal, off cells do not broadcast anything
(e.g. for keyA cells, keyA + S -> keyA + S + signal A).
Broadcast species diffuse out, diluting with distance.
[clock]=off
(c) Receiving Neighbor States
Cells interpret broadcasts locally based on keys
(e.g. for keyA cells, keyA + signal B -> keyA + signal B + R,
and keyA + signal D -> keyA + signal B + L).
Signals below a threshold concentration are ignored,
so distant cells can't communicate.
[clock]=off
(d) Calculating Next States
A composite Boolean function f(L, S, R), executed by
the global reaction network calculates next state.
When clock goes high, calculation is stored in [state].
[clock]=on
off
on
on
on
off
keyC
keyD
keyA
keyB
keyC
mixed species shown as stacked squares
signal D
signal A
signal B
signal D
signal A
signal B
on
on
off
on
off
Legend:
State
KeyA
KeyB
KeyC
KeyD
SignalA
SignalB
SignalC
SignalD
Right neighbor ON
Left neighbor ON
Both neighbors ON

Fig. 3. ***The chemical CA we describe*** performs three types of operations. (a) The state of each cell is stored locally. (b & c) Cells communicate their state to their immediate neighbor cells. (d) A Boolean logic circuit calculates each cell's next state as a function of the cell's own state and the state of its neighbors, and stores this new state in (a), completing one cycle of automaton dynamics. A global clock signal synchronizes these three operations. The clock is *off* for the communication and calculation stages, and turns *on* to allow the calculated new state to be stored in memory for the next cycle.

In our construction, each cell is a region of the substrate with a static, uniformly high concentration of a particular catalyst molecule. Catalyst molecules are attached to the substrate, so they do not diffuse. We call these molecules 'keys.' In our one-dimensional grid, cells can have one of four different types of keys (Key_A, Key_B, Key_C and Key_D) with key types repeating as one proceeds down the channel so that cells can identify neighbors based on local key type (Fig. 3a). For instance, cells defined by Key_A are always neighbored on the right by Key_B cells, and the left by Key_D cells. Cells are separated by 'spacer' regions that do not contain keys.

Since key molecules only participate in reactions as catalysts, it is assumed that they are not depleted over time[2]. In this paper we assume that this pattern of key catalysts is present at the start of the reaction as a set of initial conditions. Such a grid pattern could either be manually patterned into the substrate by top-down techniques, such as microcontact printing[25] or directed hydrogel assembly[26], or generated by a separate bottom-up RD patterning program[24].

In addition to the static pattern of key catalysts, a mix of many other freely diffusing "program" species is supplied across the substrate. These program molecules interact with the key molecules to emulate the dynamics of a CA in a cycle of three discrete conceptual stages (Fig. 3b-c). In the first stage, cells share their states with their neighbors and receive the states of other neighbors. Next, cells use information about their local state and the states of their neighbors to determine their state at the next time step. Finally, the calculated state is stored as the current state. Cycles of communication, calculation of new states, and storage of the new state in a cell's memory emulate the dynamics of a CA. In the construction below, an externally provided clock signal synchronizes the three cycles. While a global clocking mechanism isn't strictly required to implement a CA[27], we chose to construct a CA with a clock, because such a system can compute niversally when each cell can take one of only two possible states.

3.1 Communication: Sending and Receiving Addressable Signals

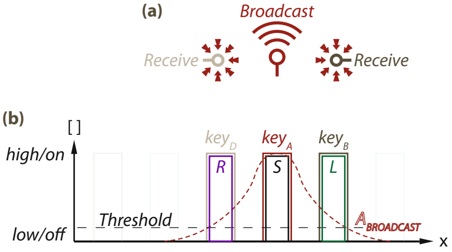


Fig. 4. ***Communication Stage.*** (a) Using a broadcast module, each *on* cell produces a gradient encoding its current state and key. Receive modules interpret broadcasts based on the identity of their local key. (b) Plot of $[species]$ $vs.$ x for a single *on* A-type cell, with local $[S]$ = $high$. A broadcast module generates a stable gradient of $A_{Broadcast}$ that decays with distance from Key_A. Receive modules at D-type cells interpret $A_{Broadcast}$ as R, while receive modules at B-type cells interpret the same broadcast as L. Broadcast signals below a threshold are ignored, so cells only communicate with their neighbors.

We begin the description of a CA cycle in the Communication Stage, right after a set of cell states for the last time step have been stably stored. At this point, each cell's current *on/off* state is represented, respectively, by the local *high* or *low* concentration of a 'state' species. During the communication stage of a computation cycle, cells must communicate their current state to their immediate neighbors.

Communication is managed by Broadcast and Receive modules (Fig. 4). Each *on* cell broadcasts information about its current state by producing a signal within the cell that can diffuse away from the cell. Broadcast modules (Fig. 2g) execute this function. In order for neighboring cells to interpret these broadcasts as coming from the left or right neighbor cell, these broadcasts must contain information about which cell is sending them.

[2] In practice, catalysts have a fixed turnover number. Reactions that cyclically produce and degrade catalysts could enable the periodic replacement of key catalysts (and other catalysts in the system) that are no longer functional.

The identity of the cell broadcasting information about its state is encoded using the key types of cells: Cells that are defined by ‘key A’ species broadcast ‘signal A’, those defined by ‘key B’ broadcast ‘signal B’, and so on. The distance that broadcast signals propagate is controlled by the production and degradation rates of the Broadcast module, such that a cell’s broadcasts only reach its neighbors. Each key has its own separate broadcast module.

The counterpart to a Broadcast module, a Receive module (Fig. 2f), receives signals transmitted by a cell’s neighbors and translates them into local information about the state of a neighboring cell. This conversion is also done in a cell-specific manner, such that each cell converts particular broadcasts into information about particular types of neighbors. For example, within cells defined by Key_A, the key species catalyzes the conversion of the broadcast signal from Key_B into a species that encodes the $right$ neighbor’s state as *on* and catalyzes the conversion of the broadcast signal from Key_D into a species that encodes the $left$ neighbor’s state as on. Key species B through D catalyze a corresponding set of reactions to produce signals that encode whether their $right$ and $left$ neighbors are on. Each conversion reaction of a broadcast to a type of neighbor information is managed by a separate receive module. Because there are four key types and each cell has two types of neighbors, eight Receive modules are required.

Receive modules convert broadcasts into “preliminary neighbor signals.” These preliminary neighbor signals are at different concentrations throughout a cell because they are copies of the broadcast signals, which decay in concentration with the distance from the neighbor. To produce uniform *on*/*off* neighbor signals throughout a cell, comparators (Fig. 2h) rectify preliminary neighbor signals, producing digital “processed neighbor signals” whose *on* levels are the same across a cell.

Together, the broadcast and receive modules ensure that after some period of broadcasting, each cell contains species that encode its own state and those of its neighbors.

3.2 Calculation Stage: Calculating New State Changes

Neighbor broadcasts that are received and processed by each cell are used to calculate the next cell state. Each update rule can be encoded as a Boolean circuit with the neighbors and the cell’s own state as inputs. Such circuits can be implemented as a set of reaction-diffusion program modules (Fig. 5). For instance, in a Rule 60 CA, a cell’s next generation state is *on* if its own current state is *on* OR its left-hand neighbor is *on*, but NOT if both of these states are *on*. Because the state of the right-hand neighbor is irrelevant, Rule 60 cells do not need to listen to their right-hand neighbor’s broadcast. The logic for a Rule 110 local update is performed by the sub-circuit in Figure 5d. The output signal produced by this circuit determines the target state of the cell at the next time step.

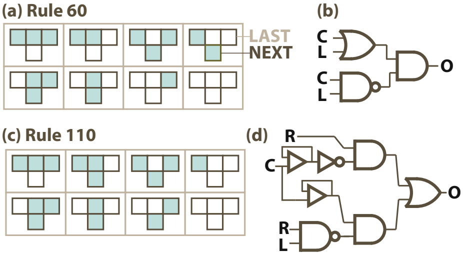


Fig. 5. ***Calculation Stage***. Every CA update rule has a corresponding Boolean logic implementation. (a) Rule 60. (b) Rule 60 converted into Boolean logic. (c) Rule 110. (d) Boolean logic for Rule 110.

3.3 Memory: Storing and Stably Encoding a New Cell State

During the Memory stage the computed next state of the cell is stored using a "flip-flop" module (Fig. 6). Flip-flops have "set" and a "reset" input signal and one output signal. When the set input is *on*, the output signal also turns *on*. The output remains *on* even after the set signal turns *off*. When the reset signal turns *on* the output turns *off*, and remains *off* until the set signal again turns *on*. This module encodes the cell's state, providing a persistent state signal used by the communication and calculation stages.

The reactions that communicate a cell's state to its neighbors and compute its next state occur without control over timing. Different cells (or different regions within a cell) may take different amounts of time to compute their new state. To ensure that all cells finish computing their next states before any other cell commits its new state to memory, calculated next states are not stored in memory until a global clock signal turns *on*. For a given CA, the clock period must be designed so that all cells finish communicating and calculating before the clock signal turns *on*. The next state must be committed to memory before the clock turns *off*.

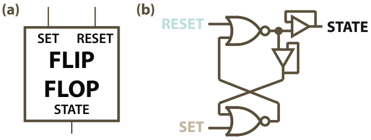


Fig. 6. ***The Memory stage*** stores cell state in a flip-flop. (a) A flip-flop's output does not change when its inputs are *off*. In our design, these inputs are *off* when the clock is *off*. When the clock is *on*, the set signal is *on* if the calculation stage outputs *on*, setting the flip-flop *on*. The reset signal is *on* if the calculation stage outputs *off*, resetting the flip-flop *off*. Memory is required so the inputs to the other stages are not affected by the calculation of new local or neighbor states. (b) Circuit for our reaction-diffusion flip-flop using modules from Fig. 2. Copy modules ensure output is not depleted by internal feedback or downstream load.

To ensure that calculated next states are not stored in memory unless the clock signal is *on*, an AND gate takes the clock signal and the calculated next state from the Calculation stage as inputs, and sends a set signal to the flip-flop module only when both the clock and the calculated next state are *on*. Another AND gate takes the clock signal and the inverse of the calculated next state as inputs, and produces a reset signal when the clock is *on* but the next state is *off*. The process of storing the new state in memory ends when the clock signal returns to a low value at the beginning of the next stage of computation.

4 Simulation of a Reaction-Diffusion Cellular Automaton

The complete automaton circuit is shown in Fig. 7. We simulated the simultaneous mass action reaction kinetics and diffusion for the entire system, using Rules 60 and 110 in the logic stage, and observed the intended cell updating for both rules (Fig. 8). The complete set of chemical reactions and the corresponding coupled partial differential equations describing these systems are provided in the Appendix A, along with all other simulation details. Concentrations within $[0, 0.3]$ μM were considered *off*, while concentrations within $[0.7, 1]$ μM were considered *on*. One irregularity that appears in our system is that the cells have blurred boundaries, an artifact that arises when chemical species produced inside of a cell diffuse across the cell boundary. This blurring effect is the reason that we included short spacer regions to separate adjacent cells, so that the logic inside of one cell does not interfere with the logic inside of its neighbors.

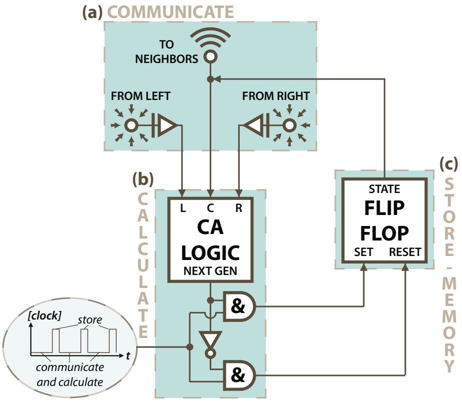


Fig. 7. ***CA Circuit Diagram*** (a) The ‘Communication’ stage. Current cell states are broadcast to neighbors, while neighbor states are received. (b) The ‘Calculation’ stage. The states of a cell and its neighbors are passed through a subcircuit that performs the update logic. The output from this subcircuit is *on* if the cell should change into an *on* state in the next generation. This next state is prevented from affecting the Memory stage by AND gates when the clock is *off*. (c) When the clock turns *on*, the next state is stored in the ‘Memory’ stage.

Two important parameters can break the reaction-diffusion program if not tuned carefully: the *on* time or ‘duty cycle’ of the clock signal, and the kinetic rates for the broadcast module. If the duty cycle is too short, then the flip-flop does not have enough time to store the intended next-generation state. In our simulations, this occurs for duty cycles shorter than 15-20 minutes. However, for particularly long duty cycles, some cell states can become desynchronized because cells can erroneously update their state multiple times within a single cell cycle. In our simulations, duty cycles longer than about an hour and a half led cells to become desynchronized.

The second critical parameter is the production rate constant for the broadcast module. When a cell is *on*, this constant must be high enough to saturate its neighbors with signaling molecule. In the worst case, where a cell is at the minimum *on* concentration of 0.7 μM, it must maintain a broadcast signal above the receive module’s threshold concentration at the farthest edge of its neighboring cell regions. On the other hand, when a cell is *off*, this constant must be low enough to avoid broadcasting any signal to its neighbors. Specifically, in the worst case where a cell is at the maximum *off* concentration of 0.3 μM, it must maintain a broadcast signal below the receive module’s threshold concentration at the closest edge of its neighboring cell regions. If either of these conditions are not met, then erroneous signals can be sent between cells.

5 Discussion

In this work we develop a method for building a CA using a reaction-diffusion program. This program consists of a set of molecules that everywhere can react and diffuse in the same way, along with a small set of molecules that are patterned into a grid of cells. The collective actions of these molecules cause the pattern of molecules that encode an “*on*” state to change over time to display a series of patterns that are the successive states of a one-dimensional binary CA. While the construction we propose is for a 1-dimensional binary CA, straightforward extensions to the system could be used to produce 2- or 3-dimensional CA, or CA in which cells can be in more than two states.

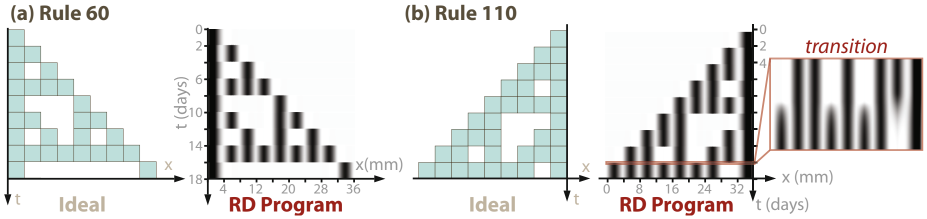


Fig. 8. ***Results of Chemical CA Simulations*** Ideal CA (left) compared to our simulated reaction-diffusion program (right). Every three-length binary input state is contained in each pattern, demonstrating correct updating for all eight possible local states. (a) Rule 60. (b) Rule 110. The dynamics shown here were computed using the set of coupled partial differential equations in the Appendix A. The detail of the rapid dynamics of a state transition are shown on the far right.

This construction thus suggests two important new capabilities for systems that are driven by designed chemical reaction networks. First, this system provides a way to generate dynamic spatial patterns, where the concentrations of species vary continuously over time, by encoding these dynamics within a CA. Second, this system makes it possible to perform computations using molecules in which the same molecular species simultaneously perform different computations within different regions of a substrate.

The capacity for this kind of spatial computation is likely to be an important part of scaling the capacity for molecular systems to process information. Because the number of independent molecular interfaces is inherently limited, it is not possible to arbitrarily increase the number of interacting molecules within a well-mixed system without introducing crosstalk. The importance of spatial computation with molecules is underscored by its prevalence in living systems. Reaction-diffusion processes are used for signaling within cells, and across tissues, where different cells (which each share the same genome) collectively coordinate tissue behavior.

While other molecular processes can perform Turing universal computation with only a limited number of molecular species, *i.e.* they are uniform, these constructions require that the state of a computation be encoded either within a single molecular assembly [28] or in the precise number of molecules [29]. As such, these constructions are susceptible to errors that can destroy the computation. In contrast, computation by the CA that we describe involves the collective action of many molecules, so it is not susceptible to errors caused by a small number of microscopic events.

However, the designs presented in this paper require the construction of large chemical reaction networks, a clock signal at regular intervals and a printed grid of different "key" molecules. Our reaction network uses 65 species to emulate a "Rule 60" CA, and 76 species to emulate a "Rule 110" CA. Further emulating these abstract chemical networks using DNA strand-displacement reactions could increase the network size by an order of magnitude, because multiple intermediate DNA strands are generally required when emulating reactions. Likely there are simplifications that could be made to our circuit, as our goal was to demonstrate that such an implementation is theoretically possible instead of designing the smallest possible circuit. For instance, it may be possible to condense some sections or our system into smaller special case circuits for particular CA updating rules. Additionally, our four-key system that provides unique identities to cells in a local group is expensive in terms of number of species, requiring four separate sets of transmitter modules and eight separate sets of receiver modules in 1-dimensional space, and a more clever means for identifying neighboring cells may exist. However, it is unclear how to reduce the number of strands in our system by an order of magnitude.

Generally, the complexity of our circuits suggests that implementing even a simple 1-dimensional automaton would be challenging with current chemical computers. Constructing CA as complex as von Neumann's self-replicating automata is likely to be infeasible for the foreseeable future. It will therefore be important to ask whether there are more efficient models for spatial-computing in which complex behaviors such as self-replication or healing can be designed as simply as possible. One starting point is to consider computation systems that do not require an explicit regular grid, such as Petri nets[30], Lindenmayer systems[31], or graph automata[32, 33], and un-clocked systems such as asynchronous CA[27].

More generally, we might ask not only how to perform molecular computation using space as a medium, but how to construct a scalable architecture for computing appropriate responses of a material to stimuli that are presented across space and time. Patterns generated by CA could act as blueprints, encoding dynamic information spatially. By constructing CA in chemical networks, it may be possible to use this information to coordinate the behavior of intelligent programmable materials. Biological cells, tissues, and synthetic nanostructures could potentially respond to local instructions released by an embedded chemical automaton. CA could endow these physical systems with unique properties, creating artificial structures that heal, self-replicate and evolve.

Acknowledgements. The authors would like to thank Paul Rothemund, Damien Woods, Josh Fern, John Zenk, and the anonymous referees for insightful reading and comments. This work was supported by NSF-CCF-1161941 and a grant to the Turing Centenary Project by the John Templeton Foundation.

References

1. Murray, J.D.: Mathematical Biology II: Spatial Models and Biomedical Applications, 3rd edn. Springer, New York (2003)
2. Greenfield, D., et al.: Self-organization of the Escherichia coli chemotaxis network imaged with super-resolution light microscopy. PLoS Biol. 7 (2009)
3. Baker, M.D., Wolanin, P.M., Stock, J.B.: Signal transduction in bacterial chemotaxis. Bioessay 28(1), 9–22 (2006)
4. Gács, P.: Reliable cellular automata with self-organization. J. Stat. Phys. 103, 45–267 (2001)
5. Gács, P., Reif, J.: A simple three-dimensional real-time reliable cellular array. J. Comput. Syst. Sci. 36, 125–147 (1988)
6. Cook, M.: Universality in elementary cellular automata. Complex Systems 15, 1–40 (2004)
7. Neary, T., Woods, D.: P-completeness of cellular automaton rule 110. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 132–143. Springer, Heidelberg (2006)
8. von Neumann, J.A.W., Burks, E.: The Theory of Self-Reproducing Automata. University of Illinois Press, Urbana (1966)
9. Codd, E.F.: Cellular automata. Academic Press, Inc., San Diego (1968)
10. Langton, C.G.: Self-reproduction in cellular automata. Physica D 10(1), 135–144 (1984)

11. Sayama, H.: A new structurally dissolvable self-reproducing loop evolving in a simple cellular automata space. Artificial Life 5(4), 343–365 (1999)
12. Turing, A.M.: The chemical basis of morphogenesis. Phil. T. Roy. Soc. B 237, 37–72 (1952)
13. Tóth, Á., Showalter, K.: Logic gates in excitable media. The Journal of Chemical Physics 103, 2058–2066 (1995)
14. Steinbock, O., Kettunen, P., Showalter, K.: Chemical wave logic gates. The Journal of Physical Chemistry 100, 18970–18975 (1996)
15. Bánsági, T., Vanag, V.K., Epstein, I.R.: Tomography of reaction-diffusion microemulsions reveals three-dimensional Turing patterns. Science 331, 1309–1312 (2011)
16. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. P. Natl. Acad. Sci. 107, 5393–5398 (2010)
17. Chen, Y., et al.: Programmable chemical controllers made from DNA. Nat. Nanotechnol. 8, 755–762 (2013)
18. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement. Science 332, 1196–1201 (2011)
19. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science 314, 1585–1588 (2006)
20. Qian, L., Winfree, E.: A simple DNA gate motif for synthesizing large-scale circuits. J. R. Soc. Interface 8, 1281–1297 (2011)
21. Smith, D.E., Perkins, T.T., Chu, S.: Dynamical scaling of DNA diffusion coefficients. Macromolecules 29, 1372–1373 (1996)
22. Allen, P.B., Chen, X., Ellington, A.D.: Spatial control of DNA reaction networks by DNA sequence. Molecules 17, 13390–13402 (2012)
23. Chirieleison, S.M., Allen, P.B., Simpson, Z.B., Ellington, A.D., Chen, X.: Pattern transformation with DNA circuits. Nature Chem. 5, 1000–1005 (2013)
24. Scalise, D., Schulman, R.: Designing modular reaction-diffusion programs for complex pattern formation. Technology 2, 55–66 (2014)
25. Ruiza, S.A., Chen, C.S.: Microcontact printing: A tool to pattern. Soft Matter 3, 168–177 (2007)
26. Du, Y., Lo, E., Ali, S., Khademhosseini, A.: Directed assembly of cell-laden microgels for fabrication of 3D tissue constructs. P. Natl. Acad. Sci. 105, 9522–9527 (2008)
27. Nehaniv, C.L.: Asynchronous automata networks can emulate any synchronous automata network. International Journal of Algebra and Computation 14, 719–739 (2004)
28. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
29. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing 7, 615–633 (2008)
30. Peterson, J.L.: Petri net theory and the modeling of systems. Prentice Hall, Englewood Cliffs (1981)
31. Lindenmayer, A.: Mathematical models for cellular interactions in development I. filaments with one-sided inputs. J. Theor. Biol. 18, 280–299 (1968)
32. Wu, A., Rosenfeld, A.: Cellular graph automata. I. basic concepts, graph property measurement, closure properties. Information and Control 42, 305–329 (1979)
33. Tomita, K., Kurokawa, H., Murata, S.: Graph automata: natural expression of self-reproduction. Physica D: Nonlinear Phenomena 171, 197–210 (2002)
34. Zhang, D.Y., Winfree, E.: Control of DNA strand displacement kinetics using toehold exchange. J. Am. Chem. Soc. 131, 17303–17314 (2009)
35. Lukacs, G.L., Haggie, P., Seksek, O., Lechardeur, D., Verkman, N.F.A.: Size-dependent DNA mobility in cytoplasm and nucleus. J. Biol. Chem. 275 (2000)
36. Stellwagen, E., Lu, Y., Stellwagen, N.: Unified description of electrophoresis and diffusion for DNA and other polyions. Biochemistry 42 (2003)

A Rule 110 Partial Differential Equations

"I hope to say something about a 'continuous' rather than 'crystalline' model [of automata]. There, as far as I can now see, a system of nonlinear partial differential equations, essentially of the diffusion type, will be used."

- John von Neumann (Oct. 28th, 1952) discussing unfinished Theory of Automata.

von Neumann papers, Library of Congress, Box 28 "Theory of Automata".

This section describes the set of coupled partial differential equations that govern our Rule 110 chemical automaton. These equations use standard mass-action equations and the diffusion equation. Figure 8(b) contains a plot of the solution to these equations. One equation is devoted to each of the 76 species in our network. Figure 9 shows a detail of the circuit from Figure 7 for a Rule 110 automaton, with each species labeled.

Unless otherwise specified, all species start with zero initial concentration. Absorbing boundary conditions apply to all species whose concentrations change over time. Our reaction rate constants and diffusion coefficients are selected to be realistically attainable values for DNA-based reaction-diffusion networks, on the same order of magnitude as experimentally derived data in the literature[34–36]. The *Mathematica* code we used to numerically solve these equations is available upon request.

Constants:

$$xMax = 64 \qquad k_p = 0.002\,\mu\mathrm{Ms}^{-1}$$
$$D = 0.00015\,\mathrm{mm}^2\mathrm{s}^{-1} \qquad k_x = 0.002\,\mu\mathrm{M}^{-1}\mathrm{s}^{-1} \qquad c_{recTh} = 0.5\,\mu\mathrm{M}$$
$$k_T = 20\,\mu\mathrm{M}^{-1}\mathrm{s}^{-1} \qquad k_B = 0.0002\,\mu\mathrm{M}^{-2}\mathrm{s}^{-1} \qquad clkPeriod = 2*24*3600\,\mathrm{s}$$
$$k_L = 0.2\,\mu\mathrm{M}^{-1}\mathrm{s}^{-1} \qquad k_{Bd} = 0.00002\,\mathrm{s}^{-1} \qquad clkDuty = .5*3600\,\mathrm{s}$$

1. External Signals:
   (a) Keys (in μM):

$$Key_A(t,x) = \begin{cases} 1/(1+Exp[-25*(Mod(x,16)-1)]) & : Mod(x,16) \leq 2 \\ 1-1/(1+Exp[-25*(Mod(x,16)-3)]) & : otherwise. \end{cases} \quad (1)$$

$$Key_B(t,x) = \begin{cases} 1/(1+Exp[-25*(Mod(x,16)-5)]) & : Mod(x,16) \leq 6 \\ 1-1/(1+Exp[-25*(Mod(x,16)-7)]) & : otherwise. \end{cases} \quad (2)$$

$$Key_C(t,x) = \begin{cases} 1/(1+Exp[-25(Mod(x,16)-9)]) & : mod(x,16) \leq 10 \\ 1-1/(1+Exp[-25(Mod(x,16)-11)]) & : otherwise. \end{cases} \quad (3)$$

$$Key_D(t,x) = \begin{cases} 1/(1+Exp[-25(Mod(x,16)-13)]) & : mod(x,16) \leq 14 \\ 1-1/(1+Exp[-25(Mod(x,16)-15)]) & : otherwise. \end{cases} \quad (4)$$

   (b) Clock:

$$clk(t,x) = \begin{cases} 1\,\mu\mathrm{M} & : t < 4000 \\ 1\,\mu\mathrm{M} & : Mod(t, clkPeriod) < clkDuty \\ 0 & : otherwise. \end{cases} \quad (5)$$

$$\frac{\partial clk_{On}(t,x)}{\partial t} = D\nabla^2 clk_{On}(t,x) - k_d * clk_{On}(t,x) + k_d * clk(t,x) - k_L * clk_{On}(t,x) * Sg_{nxOn}(t,x) \quad (6)$$

$$\frac{\partial clk_{Off}(t,x)}{\partial t} = D\nabla^2 clk_{Off}(t,x) - k_d * clk_{Off}(t,x) + k_d * clk(t,x) - k_L * clk_{Off}(t,x) * Sg_{OffNx}(t,x) \quad (7)$$

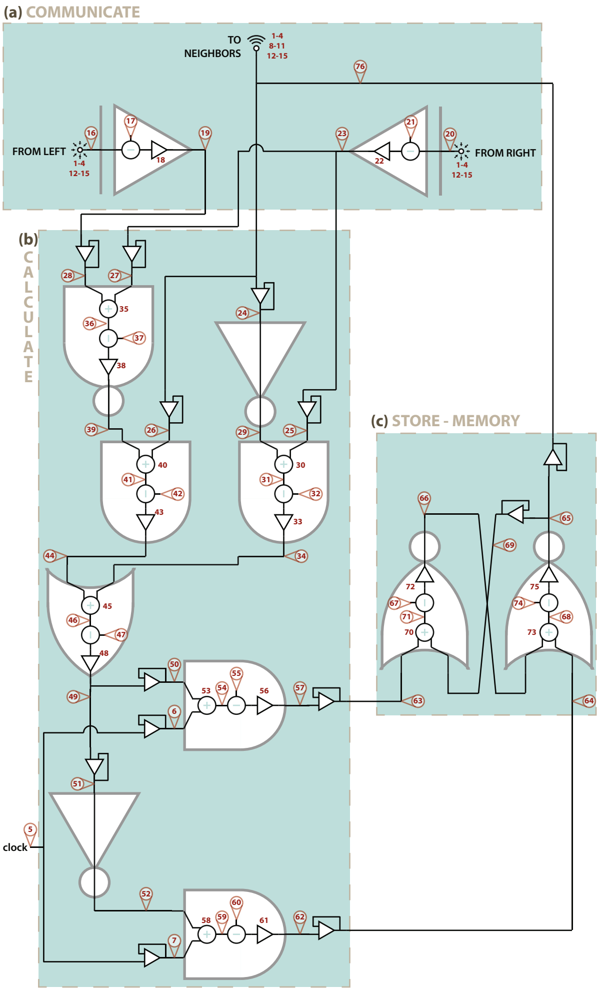
2. Communication Stage:
   (a) Broadcast Modules (in μM):

$$Src_A(t,x) = 1 \quad (8) \qquad Src_B(t,x) = 1 \quad (9) \qquad Src_C(t,x) = 1 \quad (10) \qquad Src_D(t,x) = 1 \quad (11)$$

   (b) Broadcast signals, note initial concentration of Sig_A triggers initial *on* cell:

$$\frac{\partial Sig_A(t,x)}{\partial t} = D\nabla^2 Sig_A(t,x) + k_B Src_A(t,x) Key_A(t,x) Last(t,x) - k_{Bd} Sig_A(t,x) \quad (12)$$

$$Sig_A(t=0,x) = \begin{cases} 2\,\mu\mathrm{M} & : x_{max} - 8 < x < x_{max} - 0.1 \\ 0 & : otherwise. \end{cases}$$



1 - key A
2 - key B
3 - key C
4 - key D
5 - clock (clk)
6 - clock On (clkOn)
7 - clock Off (clkOff)
8 - source A (SrcA)
9 - source B (SrcB)
10 - source C (SrcC)
11 - source D (SrcD)
12 - signal A (SigA)
13 - signal B (SigB)
14 - signal C (SigC)
15 - signal D (SigD)
16 - left raw (Lraw)
17 - threshold left (ThL)
18 - amp left (AmpL)
19 - left (Lft)
20 - right raw (Rraw)
21 - threshold right (Thr)
22 - amp right (Ampr)
23 - right (Rght)
24 - Previous for Birth (BrPr)
25 - Right for Birth (RBr)
26 - Previous for On (OnPr)
27 - Right for Crowded (RCr)
28 - Left for Crowded (LCr)
29 - Previous for Off (OffPr)
30 - `birth' sum gate (SgBr)
31 - `birth' sum signal (SumBr)
32 - `birth' threshold (ThBr)
33 - `birth' amp (AmpBr)
34 - `birth' signal (Br)
35 - neighbor off sum gate (SgnbOff)
36 - neighbor off sum signal (SumnbOff)
37 - neighbor off threshold (ThnbOff)
38 - neighbor off amp (AmpnbOff)
39 - neighbor off signal (nbrOff)
40 - `crowded' sum gate (SgCr)
41 - `crowded' sum signal (SumCr)
42 - `crowded' threshold (ThCr)
43 - `crowded' amp (AmpCr)
44 - still alive signal (Live)
45 - next state on sum gate (SgNx)
46 - next state on sum signal (SgNx)
47 - next state on threshold (ThNx)
48 - next state on amp (AmpNx)
49 - next state on signal (Nx)
50 - next state on copy (OnNx)
51 - next state not on copy (NtNx)
52 - next state off signal (OffNx)
53 - on sum gate (SgNxOn)
54 - on sum signal (OnNxSum)
55 - on threshold (OnNxTh)
56 - on amp (OnNxAmp)
57 - buffered set signal (SetBfr)
58 - off sum gate (SgOffNx)
59 - off sum signal (SumOffNx)
60 - off threshold (ThOffNx)
61 - off amp (AmpOffNx)
62 - buffered reset signal (ResBfr)
63 - set signal (Set)
64 - reset signal (Res)
65 - buffered flipflop output (ffBfrd)
66 - flipflop NOT feedback (ffFbackNot)
67 - NOR gate 1 threshold (N1Th)
68 - NOR gate 2 sum signal (N2Sum)
69 - flipflop feedback (ffFback)
70 - NOR gate 1 sum gate (N1Sg)
71 - NOR gate 1 sum signal (N1Sum)
72 - NOR gate 1 amp (N1)
73 - NOR gate 2 sum gate (N2Sg)
74 - NOR gate 2 threshold (N2Th)
75 - NOR gate 2 amp (N2)
76 - Stored state (Last)

Fig. 9. ***Chemical reaction-diffusion circuit for Rule 110.*** *This is a detailed version of the circuit outlined in Fig. 7, using the modules defined in Fig. 2. Species are labelled in red by their equation numbers from Appendix section A.*, with species names and abbreviations to the right.

$$\frac{\partial Sig_B(t,x)}{\partial t} = D\nabla^2 Sig_B(t,x) + k_B Src_B(t,x) Key_B(t,x) Last(t,x) - k_{Bd} Sig_B(t,x) \quad (13)$$

$$\frac{\partial Sig_C(t,x)}{\partial t} = D\nabla^2 Sig_C(t,x) + k_B Src_C(t,x) Key_C(t,x) Last(t,x) - k_{Bd} Sig_C(t,x) \quad (14)$$

$$\frac{\partial Sig_D(t,x)}{\partial t} = D\nabla^2 Sig_D(t,x) + k_B Src_D(t,x) Key_D(t,x) Last(t,x) - k_{Bd} Sig_D(t,x) \quad (15)$$

(c) Receiving and processing left-hand neighbor signal:

$$\begin{aligned}\frac{\partial L_{raw}(t,x)}{\partial t} &= D\nabla^2 L_{raw}(t,x) - 4k_d L_{raw}(t,x) - k_T L_{raw}(t,x) Th_l(t,x) \\ &+ k_x Sig_D(t,x) Key_A(t,x) + k_x Sig_A(t,x) Key_B(t,x) + k_x Sig_B(t,x) Key_C(t,x) \\ &+ k_x Sig_C(t,x) Key_D(t,x) \quad (16)\end{aligned}$$

$$\frac{\partial Th_l(t,x)}{\partial t} = D\nabla^2 Th_l(t,x) + c_{recTh} k_p - k_d Th_l(t,x) - k_T L_{raw}(t,x) Th_l(t,x) \quad (17)$$

$$\frac{\partial Amp_l(t,x)}{\partial t} = D\nabla^2 Amp_l(t,x) + k_p - k_d Amp_l(t,x) - k_L L_{raw}(t,x) Amp_l(t,x) \tag{18}$$

$$\frac{\partial Lft(t,x)}{\partial t} = D\nabla^2 Lft(t,x) - k_d Lft(t,x) + k_L L_{raw}(t,x) Amp_l(t,x) \tag{19}$$

(d) Receiving and processing right-hand neighbor signal:

$$\begin{aligned}\frac{\partial R_{raw}(t,x)}{\partial t} &= D\nabla^2 R_{raw}(t,x) - 4k_d R_{raw}(t,x) - k_T R_{raw}(t,x) Th_r(t,x)\\ &+ k_x Sig_{(}t,x) Key_A(t,x) + k_x Sig_C(t,x) Key_B(t,x) + k_x Sig_D(t,x) Key_C(t,x)\\ &+ k_x Sig_A(t,x) Key_D(t,x)\end{aligned} \tag{20}$$

$$\frac{\partial Th_r(t,x)}{\partial t} = D\nabla^2 Th_r(t,x) + c_{recTh} k_p - k_d Th_r(t,x) - k_T R_{raw}(t,x) Th_r(t,x) \tag{21}$$

$$\frac{\partial Amp_{rt}(t,x)}{\partial t} = D\nabla^2 Amp_{rt}(t,x) + k_p - k_d Amp_{rt}(t,x) - k_L R_{raw}(t,x) Amp_{rt}(t,x) \tag{22}$$

$$\frac{\partial Rght(t,x)}{\partial t} = D\nabla^2 Rght(t,x) - k_d Rght(t,x) + k_L R_{raw}(t,x) Amp_{rt}(t,x) \tag{23}$$

3. Calculation Stage:

(a) Copy left, right and previous time step (pr) signals (multiple gates operate on each)

$$\frac{\partial Br_{pr}(t,x)}{\partial t} = D\nabla^2 Br_{pr}(t,x) - k_d Br_{pr}(t,x) + k_d Last(t,x) - k_T Br_{pr}(t,x) Off_{pr}(t,x) \tag{24}$$

$$\frac{\partial R_{br}(t,x)}{\partial t} = D\nabla^2 R_{br}(t,x) - k_d R_{br}(t,x) + k_d Rght(t,x) - k_L R_{br}(t,x) Sg_{br}(t,x) \tag{25}$$

$$\frac{\partial On_{pr}(t,x)}{\partial t} = D\nabla^2 On_{pr}(t,x) - k_d On_{pr}(t,x) + k_d Last(t,x) - k_L On_{pr}(t,x) Sg_{cr}(t,x) \tag{26}$$

$$\frac{\partial R_{cr}(t,x)}{\partial t} = D\nabla^2 R_{cr}(t,x) - k_d R_{cr}(t,x) + k_d Rght(t,x) - k_L R_{cr}(t,x) Sg_{nbOff}(t,x) \tag{27}$$

$$\frac{\partial L_{cr}(t,x)}{\partial t} = D\nabla^2 L_{cr}(t,x) - k_d L_{cr}(t,x) + k_d Lft(t,x) - k_L L_{cr}(t,x) Sg_{nbOff}(t,x) \tag{28}$$

$$\begin{aligned}\frac{\partial Off_{pr}(t,x)}{\partial t} &= D\nabla^2 Off_{pr}(t,x) + k_p - k_d Off_{pr}(t,x) -_T Br_{pr}(t,x) Off_{pr}(t,x)\\ &- k_L Off_{pr}(t,x) Sg_{br}(t,x)\end{aligned} \tag{29}$$

(b) Boolean logic for Br (birth) condition: *off* to *on* transition

$$\begin{aligned}\frac{\partial Sg_{br}(t,x)}{\partial t} &= D\nabla^2 Sg_{br}(t,x) + 2k_p - k_d Sg_{br}(t,x)\\ &- k_L Off_{pr}(t,x) Sg_{br}(t,x) - k_L R_{br}(t,x) Sg_{br}(t,x)\end{aligned} \tag{30}$$

$$\begin{aligned}\frac{\partial Sum_{br}(t,x)}{\partial t} &= D\nabla^2 Sum_{br}(t,x) - k_d Sum_{br}(t,x)\\ &+ k_L Off_{pr}(t,x) Sg_{br}(t,x) + k_L R_{br}(t,x) Sg_{br}(t,x) - k_T Sum_{br}(t,x) Th_{br}(t,x)\end{aligned} \tag{31}$$

$$\frac{\partial Th_{br}(t,x)}{\partial t} = D\nabla^2 Th_{br}(t,x) + 1.35k_p - k_d Th_{br}(t,x) - k_T Sum_{br}(t,x) Th_{br}(t,x) \tag{32}$$

$$\frac{\partial Amp_{br}(t,x)}{\partial t} = D\nabla^2 Amp_{br}(t,x) + k_p - k_d Amp_{br}(t,x) - k_L Sum_{br}(t,x) Amp_{br}(t,x) \tag{33}$$

$$\frac{\partial Br(t,x)}{\partial t} = D\nabla^2 Br(t,x) - k_d Br(t,x) + k_L Sum_{br}(t,x) Amp_{br}(t,x) - k_L Br(t,x) Sg_{nx}(t,x) \tag{34}$$

(c) Boolean logic for no death condition (*on* and at least one neighbor *off*): stay *on*

$$\begin{aligned}\frac{\partial Sg_{nbOff}(t,x)}{\partial t} &= D\nabla^2 Sg_{nbOff}(t,x) + 2k_p - k_d Sg_{nbOff}(t,x)\\ &- k_L R_{cr}(t,x) Sg_{nbOff}(t,x) - k_L L_{cr}(t,x) Sg_{nbOff}(t,x)\end{aligned} \tag{35}$$

$$\begin{aligned}\frac{\partial Sum_{nbOff}(t,x)}{\partial t} &= D\nabla^2 Sum_{nbOff}(t,x) - k_d Sum_{nbOff}(t,x) + k_L R_{cr}(t,x) Sg_{nbOff}(t,x)\\ &+ k_L L_{cr}(t,x) Sg_{nbOff}(t,x) - k_T Sum_{nbOff}(t,x) Th_{nbOff}(t,x)\end{aligned} \tag{36}$$

$$\begin{aligned}\frac{\partial Th_{nbOff}(t,x)}{\partial t} &= D\nabla^2 Th_{nbOff}(t,x) + 1.35k_p - k_d Th_{nbOff}(t,x)\\ &- k_T Sum_{nbOff}(t,x) Th_{nbOff}(t,x)\end{aligned} \tag{37}$$

$$\begin{aligned}\frac{\partial Amp_{nbOff}(t,x)}{\partial t} &= D\nabla^2 Amp_{nbOff}(t,x) + k_p - k_d Amp_{nbOff}(t,x)\\ &- k_L Th_{nbOff}(t,x) Amp_{nbOff}(t,x)\end{aligned} \tag{38}$$

$$\frac{\partial nbrOff(t,x)}{\partial t} = D\nabla^2 nbrOff(t,x) - k_d nbrOff(t,x) + k_L Th_{nbOff}(t,x) Amp_{nbOff} y(t,x)$$

$$-k_L nbrOff(t,x) Sg_{cr}(t,x) \tag{39}$$

$$\begin{aligned}\frac{\partial Sg_{cr}(t,x)}{\partial t} &= D\nabla^2 Sg_{cr}(t,x) + 2k_p - k_d Sg_{cr}(t,x) \\ &-k_L On_{pr}(t,x) Sg_{cr}(t,x) - k_L nbrOff(t,x) Sg_{cr}(t,x)\end{aligned} \tag{40}$$

$$\begin{aligned}\frac{\partial Sum_{cr}(t,x)}{\partial t} &= D\nabla^2 Sum_{cr}(t,x) - k_d Sum_{cr}(t,x) + k_L On_{pr}(t,x) Sg_{cr}(t,x) \\ &+k_L nbrOff(t,x) Sg_{cr}(t,x) - k_T Sum_{cr}(t,x) Th_{cr}(t,x)\end{aligned} \tag{41}$$

$$\frac{\partial Th_{cr}(t,x)}{\partial t} = D\nabla^2 Th_{cr}(t,x) + 1.35k_p - k_d Th_{cr}(t,x) - k_T Sum_{cr}(t,x) Th_{cr}(t,x) \tag{42}$$

$$\frac{\partial Amp_{cr}(t,x)}{\partial t} = D\nabla^2 Amp_{cr}(t,x) + k_p - k_d Amp_{cr}(t,x) - k_L Sum_{cr}(t,x) Amp_{cr}(t,x) \tag{43}$$

$$\begin{aligned}\frac{\partial Live(t,x)}{\partial t} &= D\nabla^2 Live(t,x) - k_d Live(t,x) \\ &+k_L Sum_{cr}(t,x) Amp_{cr}(t,x) - k_L Live(t,x) Sg_{nx}(t,x)\end{aligned} \tag{44}$$

(d) Boolean logic to determine if next state is *on*

$$\begin{aligned}\frac{\partial Sg_{nx}(t,x)}{\partial t} &= D\nabla^2 Sg_{nx}(t,x) + 2k_p - k_d Sg_{nx}(t,x) \\ &-k_L Live(t,x) Sg_{nx}(t,x) - k_L Br(t,x) Sg_{nx}(t,x)\end{aligned} \tag{45}$$

$$\begin{aligned}\frac{\partial Sum_{nx}(t,x)}{\partial t} &= D\nabla^2 Sum_{nx}(t,x) - k_d Sum_{nx}(t,x) \\ &+k_L Live(t,x) Sg_{nx}(t,x) + k_L Br(t,x) Sg_{nx}(t,x) - k_T Sum_{nx}(t,x) Th_{nx}(t,x)\end{aligned} \tag{46}$$

$$\frac{\partial Th_{nx}(t,x)}{\partial t} = D\nabla^2 Th_{nx}(t,x) + 0.65k_p - k_d Th_{nx}(t,x) - k_T Sum_{nx}(t,x) Th_{nx}(t,x) \tag{47}$$

$$\frac{\partial Amp_{nx}(t,x)}{\partial t} = D\nabla^2 Amp_{nx}(t,x) + k_p - k_d Amp_{nx}(t,x) - k_L Sum_{nx}(t,x) Amp_{nx}(t,x) \tag{48}$$

$$\frac{\partial Nx(t,x)}{\partial t} = D\nabla^2 Nx(t,x) - k_d Nx(t,x) + k_L Sum_{nx}(t,x) Amp_{nx}(t,x) \tag{49}$$

$$\frac{\partial On_{nx}(t,x)}{\partial t} = D\nabla^2 On_{nx}(t,x) - k_d On_{nx}(t,x) + k_d Nx(t,x) - k_L On_{nx}(t,x) Sg_{nxOn}(t,x) \tag{50}$$

(e) Boolean logic to determine if next state is *off*

$$\frac{\partial Nt_{nx}(t,x)}{\partial t} = D\nabla^2 Nt_{nx}(t,x) - k_d Nt_{nx}(t,x) + k_d Nx(t,x) - k_T Nt_{nx}(t,x) Off_{nx}(t,x) \tag{51}$$

$$\begin{aligned}\frac{\partial Off_{nx}(t,x)}{\partial t} &= D\nabla^2 Off_{nx}(t,x) + k_p - k_d Off_{nx}(t,x) \\ &-k_T Nt_{nx}(t,x) Off_{nx}(t,x) - k_L Off_{nx}(t,x) Sg_{OffNx}(t,x)\end{aligned} \tag{52}$$

(f) Clocked synchronization gates

$$\begin{aligned}\frac{\partial Sg_{nxOn}(t,x)}{\partial t} &= D\nabla^2 Sg_{nxOn}(t,x) + 2k_p - k_d Sg_{nxOn}(t,x) \\ &-k_L On_{nx}(t,x) Sg_{nxOn}(t,x) - k_L clk_{On}(t,x) Sg_{nxOn}(t,x)\end{aligned} \tag{53}$$

$$\begin{aligned}\frac{\partial On_{nx}Sum(t,x)}{\partial t} &= D\nabla^2 On_{nx}Sum(t,x) - k_d On_{nx}Sum(t,x) + k_L On_{nx}(t,x) Sg_{nxOn}(t,x) \\ &+k_L clk_{On}(t,x) Sg_{nxOn}(t,x) - k_T On_{nx}Sum(t,x) On_{nx}Th(t,x)\end{aligned} \tag{54}$$

$$\begin{aligned}\frac{\partial On_{nx}Th(t,x)}{\partial t} &= D\nabla^2 On_{nx}Th(t,x) + 1.35k_p - k_d On_{nx}Th(t,x) \\ &-k_T On_{nx}Sum(t,x) On_{nx}Th(t,x)\end{aligned} \tag{55}$$

$$\begin{aligned}\frac{\partial On_{nx}Amp(t,x)}{\partial t} &= D\nabla^2 On_{nx}Amp(t,x) + k_p - k_d On_{nx}Amp(t,x) \\ &-k_L On_{nx}Sum(t,x) On_{nx}Amp(t,x)\end{aligned} \tag{56}$$

$$\frac{\partial Set_{Bfr}(t,x)}{\partial t} = D\nabla^2 Set_{Bfr}(t,x) - k_d Set_{Bfr}(t,x) + k_L On_{nx}Sum(t,x) On_{nx}Amp(t,x) \tag{57}$$

$$\begin{aligned}\frac{\partial Sg_{OffNx}(t,x)}{\partial t} &= D\nabla^2 Sg_{OffNx}(t,x) + 2k_p - k_d Sg_{OffNx}(t,x) \\ &-k_L Off_{nx}(t,x) Sg_{OffNx}(t,x) - k_L clk_{Off}(t,x) Sg_{OffNx}(t,x)\end{aligned} \tag{58}$$

$$\begin{aligned}\frac{\partial Sum_{OffNx}(t,x)}{\partial t} &= D\nabla^2 Sum_{OffNx}(t,x) - k_d Sum_{OffNx}(t,x) \\ &+k_L Off_{nx}(t,x) Sg_{OffNx}(t,x) + k_L clk_{Off}(t,x) Sg_{OffNx}(t,x) - k_T Sum_{OffNx}(t,x) Th_{OffNx}(t,x)\end{aligned} \tag{59}$$

$$\begin{aligned}\frac{\partial Th_{OffNx}(t,x)}{\partial t} &= D\nabla^2 Th_{OffNx}(t,x) + 1.35k_p - k_d Th_{OffNx}(t,x) \\ &-k_T Sum_{OffNx}(t,x) Th_{OffNx}(t,x)\end{aligned} \tag{60}$$

$$\begin{aligned}\frac{\partial Amp_{OffNx}(t,x)}{\partial t} &= D\nabla^2 Amp_{OffNx}(t,x) + k_p - k_d Amp_{OffNx}(t,x) \\ &-k_L Sum_{OffNx}(t,x) Amp_{OffNx}(t,x)\end{aligned} \tag{61}$$

$$\frac{\partial Res_{Bfr}(t,x)}{\partial t} = D\nabla^2 Res_{Bfr}(t,x) - k_d Res_{Bfr}(t,x) + k_L Sum_{OffNx}(t,x) Amp_{OffNx}(t,x) \tag{62}$$

4. Storage Stage
   (a) Copies of Set/Res signals

$$\frac{\partial Set(t,x)}{\partial t} = D\nabla^2 Set(t,x) - k_d Set(t,x) + k_d Set_{Bfr}(t,x) - k_L Set(t,x) N1Sg(t,x) \tag{63}$$

$$\frac{\partial Res(t,x)}{\partial t} = D\nabla^2 Res(t,x) - k_d Res(t,x) + k_d Res_{Bfr}(t,x) - k_L Res(t,x) N2Sg(t,x) \tag{64}$$

   (b) Flip-flop module

$$\frac{\partial ffBfrd(t,x)}{\partial t} = D\nabla^2 ffBfrd(t,x) - k_d ffBfrd(t,x) + k_L N2Th(t,x) N2(t,x) \tag{65}$$

$$\begin{aligned}\frac{\partial ffFbackNot(t,x)}{\partial t} &= D\nabla^2 ffFbackNot(t,x) - k_d ffFbackNot(t,x) \\ &+ k_L N1Th(t,x) N1(t,x) - k_L ffFbackNot(t,x) N2Sg(t,x)\end{aligned} \tag{66}$$

$$\frac{\partial N1Th(t,x)}{\partial t} = D\nabla^2 N1Th(t,x) + 0.65k_p - k_d N1Th(t,x) - k_T N1Sum(t,x) N1Th(t,x) \tag{67}$$

$$\begin{aligned}\frac{\partial N2Sum(t,x)}{\partial t} &= D\nabla^2 N2Sum(t,x) - k_d N2Sum(t,x) + k_L Res(t,x) N2Sg(t,x) \\ &+ k_L ffFbackNot(t,x) N2Sg(t,x) - k_T N2Sum(t,x) N2Th(t,x)\end{aligned} \tag{68}$$

$$\begin{aligned}\frac{\partial ffFback(t,x)}{\partial t} &= D\nabla^2 ffFback(t,x) - k_d ffFback(t,x) + k_d ffBfrd(t,x) \\ &- k_L ffFback(t,x) N1Sg(t,x)\end{aligned} \tag{69}$$

$$\begin{aligned}\frac{\partial N1Sg(t,x)}{\partial t} &= D\nabla^2 N1Sg(t,x) + 2k_p - k_d N1Sg(t,x) \\ &- k_L Set(t,x) N1Sg(t,x) - k_L ffFback(t,x) N1Sg(t,x)\end{aligned} \tag{70}$$

$$\begin{aligned}\frac{\partial N1Sum(t,x)}{\partial t} &= D\nabla^2 N1Sum(t,x) - k_d N1Sum(t,x) + k_L Set(t,x) N1Sg(t,x) \\ &+ k_L ffFback(t,x) N1Sg(t,x) - k_T N1Sum(t,x) N1Th(t,x)\end{aligned} \tag{71}$$

$$\frac{\partial N1(t,x)}{\partial t} = D\nabla^2 N1(t,x) + k_p - k_d N1(t,x) - k_L N1Th(t,x) N1(t,x) \tag{72}$$

$$\begin{aligned}\frac{\partial N2Sg(t,x)}{\partial t} &= D\nabla^2 N2Sg(t,x) + 2k_p - k_d N2Sg(t,x) \\ &- k_L Res(t,x) N2Sg(t,x) - k_L ffFbackNot(t,x) N2Sg(t,x)\end{aligned} \tag{73}$$

$$\frac{\partial N2Th(t,x)}{\partial t} = D\nabla^2 N2Th(t,x) + 0.65k_p - k_d N2Th(t,x) - k_T N2Sum(t,x) N2Th(t,x) \tag{74}$$

$$\frac{\partial N2(t,x)}{\partial t} = D\nabla^2 N2(t,x) + k_p - k_d N2(t,x) - k_L N2Th(t,x) N2(t,x) \tag{75}$$

   (c) Stored State

$$\frac{\partial Last(t,x)}{\partial t} = D\nabla^2 Last(t,x) - k_d Last(t,x) + k_d ffBfrd(t,x) \tag{76}$$

Computational Design of Reaction-Diffusion Patterns Using DNA-Based Chemical Reaction Networks

Neil Dalchau[1], Georg Seelig[2], and Andrew Phillips[1]

[1] Microsoft Research, Cambridge, CB1 2FB, UK
{ndalchau,aphillip}@microsoft.com
[2] University of Washington, WA, USA
gseelig@uw.edu

Abstract. DNA self-assembly is a powerful technology for controlling matter at the nanometre to micron scale, with potential applications in high-precision organisation and positioning of molecular components. However, the ability to program DNA-only self-organisation beyond the microscopic scale is currently lacking. In this paper we propose a computational method for programming spatial organisation of DNA at the centimetre scale, by means of DNA strand displacement reaction diffusion systems. We use this method to analyse the spatiotemporal dynamics of an autocatalytic system, a predator-prey oscillator and a two-species consensus network. We find that both autocatalytic and oscillating systems can support travelling waves across centimetre distances, and that consensus in a spatial context results in the spontaneous formation of distinct spatial domains, in which one species is completely eliminated. Together, our results suggest that programmed spatial self-organisation of DNA, through a reaction diffusion mechanism, is achievable with current DNA strand displacement technology.

Keywords: DNA strand displacement, autocatalysis, consensus, approximate majority, reaction-diffusion, oscillators, travelling waves.

1 Introduction

Biological systems rely on a variety of mechanisms for the organisation of matter across spatial scales. At the subcellular scale, molecular self-assembly is an efficient way of creating shapes and structures, for example to assemble viral capsids from protein building blocks [1]. To propagate signals over distances beyond the cellular scale, biological systems often rely on diffusible signalling molecules that interact with other molecular components to form reaction diffusion patterns [2]. For instance, a reaction diffusion mechanism was recently proposed to explain digit formation in mouse embryos [3].

In recent years, DNA nanotechnology has made spectacular progress in the structural self-assembly of micron-sized objects with nanometre scale precision

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 84–99, 2014.

[4]. Even centimetre-length DNA crystals [5] and hydrogels [6] have been experimentally realised. However, these materials have relatively simple long-range order that is either periodic or random. To replicate the diversity and scale of biological organisms, novel approaches are needed that extend to the centimetre scale and beyond. A reaction diffusion mechanism based on DNA molecules diffusing through and reacting in a hydrogel or similar matrix could provide a promising solution.

There are several examples of chemical systems capable of pattern formation, most famously the Belousov-Zhabotinskii system [7]. One of the earliest examples of spatial organisation in a cell-free biochemical system is the travelling waves of *in vitro* evolving RNA observed by Bauer and co-workers [8]. More recently, Isalan *et al.* [9] engineered a cell-free transcription-translation system that mimicked the pattern forming program observed in *Drosophila*. Simpson *et al.* proposed a method for implementing amorphous computation with *in vitro* transcription networks [10]. Rondelez and co-workers elegantly demonstrated travelling wave patterns in a molecular predator prey model created using a combination of DNA molecules, nicking enzymes and polymerases [11]. However, all of these approaches rely on enzymes and are thus more sensitive to reaction conditions than an approach that uses DNA alone.

Ellington and co-workers took a first step towards demonstrating pattern formation with enzyme-free DNA strand displacement systems by engineering a DNA-based edge detection circuit [12, 13]. Using UV light, a target pattern was first projected onto a gel that contained multiple DNA reactants, some of which had photo-cleavable bases. This initial pre-patterning activated a reaction pathway that led to pattern refinement. Scalise and Schulman [14] developed a modelling framework based on reaction diffusion equations for generating arbitrary shapes and patterns, starting from a small set of localised DNA signal *sources*. In principle, transient wave patterns and even stable Turing patterns could emerge from homogeneous inital conditions with small random perturbations. If realised with DNA molecules, such emergent patterns could complement existing approaches for self-assembly or diffusible signalling based on pre-patterned information by providing a mechanism for initial symmetry breaking. To understand if such self-organised patterns are within reach of current DNA technology, we first investigate how information can propagate in simple DNA strand displacement systems with a single localised input source. Then, we investigate the behaviour of a multi-reaction network with homogeneous (but noisy) initial conditions.

Recent work demonstrated that it is possible, in principle, to build DNA components that can approximate the kinetics of any well-mixed chemical reaction network (CRN) [15–17]. CRNs were proposed as a prescriptive programming language for specifying a target behaviour, which is then realised with DNA components. Here, we propose to build on this work to go beyond well-mixed chemistry. In particular, we demonstrate that DNA molecules can be programmed to approximate the behaviour of chemical reaction diffusion systems that yield self-organising patterns with macroscopic dimensions.

Central to this work is the use of computer-aided design tools. The Visual DSD (vDSD) software uses a textual syntax to describe an initial set of DNA molecules, and automatically generates the corresponding strand displacement reactions [18, 19]. It has previously been used to aid the design of several DSD circuits [17, 20–22]. Here, we extend vDSD to enable simulations of reaction diffusion DSD systems in a range of spatially heterogeneous scenarios. We design systems that form spatial patterns and analyse their behaviour in the context of realistic kinetic parameters. We then propose a number of scenarios that could be tested experimentally. Specifically, we demonstrate the design of systems that generate periodic travelling waves and stationary patterns, and analyse the impact of leaks arising from dysfunctional DNA gates enabling us to identify key insights and constraints on our systems prior to their construction.

2 Methods

Two-Domain DNA Strand Displacement. We consider the implementation of high-level CRNs using DNA strand displacement, following the two-domain approach proposed in [16]. *Signal strands* are single-stranded DNA (ssDNA) molecules which combine a short *toehold* domain and a longer *recognition* domain. The recognition domain specifies the identity of a signal, while the toehold domain can be shared among multiple signals. Accordingly, we use the notation $\langle \texttt{t x} \rangle$ to represent a signal strand X. To implement the stoichiometry of an arbitrary chemical reaction in the two-domain scheme, a nicked double-stranded DNA (ndsDNA) *join* gate receives all reactant strands and produces a *translator* strand, which then triggers the release of product strands from a *fork* gate (see [16, 17] for a detailed description). We adopt a naming convention for join and fork gates in which subscripts identify the reactants and products respectively. For example, the reaction $B + X \rightarrow 2X$ is implemented by $\langle \texttt{t b} \rangle$ and $\langle \texttt{t x} \rangle$ strands binding Join_{BX} gates, with Fork_{2X} gates producing two $\langle \texttt{t x} \rangle$ strands.

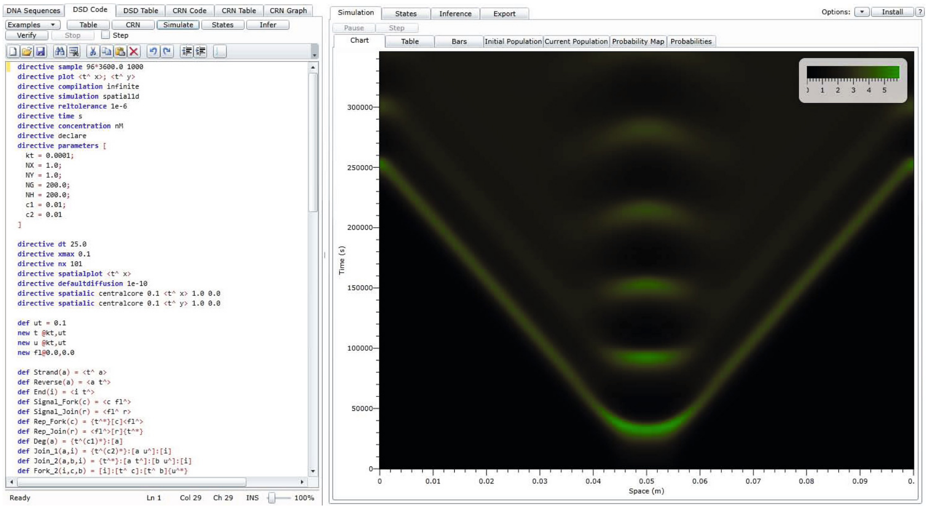
Error Modes. We consider the impact of imperfections in DNA strand/gate synthesis, as modelled in [17]. Specifically, we consider a *leak* parameter, which is the fraction of ndsDNA gates that spontaneously produce their output strand without requiring inputs.

Simulating Partial Differential Equations (PDEs) in Visual DSD. To analyse the spatiotemporal dynamics of DNA strand displacement circuits, and to provide the means for others to do this conveniently, we incorporated numerical solvers for partial differential equations into the Visual DSD (vDSD) software. Specifically, we provide numerical algorithms to solve problems of the form

$$\frac{\partial c}{\partial t} = f(c) + D\nabla^2 c \tag{1}$$

where c is the vector of concentrations and D is the diagonal matrix of diffusion rates, and ∇^2 represents the second spatial derivative. Solvers for both 1- and

2- dimensional domains have been implemented, using a Crank-Nicolson finite difference approach [23]. Extensions to the DSD programming language enable specifying PDE solver parameters, by way of *directives* (see Appendix A[1] for a complete list; Table S1). A screenshot of the new PDE-solving capabilities of vDSD is shown in Fig. 1.



```
directive sample 96*3600.0 1000
directive plot <t^ x>; <t^ y>
directive compilation infinite
directive simulation spatial1d
directive reltolerance 1e-6
directive time s
directive concentration nM
directive declare
directive parameters [
  kt = 0.0001;
  NX = 1.0;
  NY = 1.0;
  NG = 200.0;
  NH = 200.0;
  c1 = 0.01;
  c2 = 0.01
]

directive dt 25.0
directive xmax 0.1
directive nx 101
directive spatialplot <t^ x>
directive defaultdiffusion 1e-10
directive spatialic centralcore 0.1 <t^ x> 1.0 0.0
directive spatialic centralcore 0.1 <t^ y> 1.0 0.0

def ut = 0.1
new t @kt,ut
new u @kt,ut
new fl@0.0,0.0

def Strand(a) = <t^ a>
def Reverse(a) = <a t^>
def End(i) = <i t^>
def Signal_Fork(c) = <c fl^>
def Signal_Join(r) = <fl^ r>
def Rep_Fork(c) = {t^*}[c]<fl^>
def Rep_Join(r) = <fl^>[r]{t^*}
def Deg(a) = {t^(c1)*}:[a]
def Join_1(a,i) = {t^(c2)*}:[a u^]:[i]
def Join_2(a,b,i) = {t^*}:[a t^]:[b u^]:[i]
def Fork_2(i,c,b) = [i]:[t^ c]:[t^ b]{u^*}
```

Fig. 1. Screenshot of the Visual DSD software during a 1d spatial simulation

Using the *defaultdiffusion* directive, we applied a diffusion rate of 10^{-10} m^2 s^{-1} for all ssDNA strands and dsDNA gates, as approximated in [24]. Note that when simulating reaction-diffusion equations, modifications to a uniform D (i.e. $D \mapsto \alpha D'$) can be applied by rescaling the spatial domain as $x \mapsto \sqrt{\alpha}x'$. Therefore, all qualitative behaviours presented here are independent of the choice of diffusion rate.

Simulation of an Invasion Scenario. To determine how local signals propagate through a reactive medium, we consider *invasion scenarios*. These occur when the initial concentrations of specified molecular species are non-zero only in some sub-region $\mathcal{R}$ of the full domain. In a vector of concentrations $c = c(x,t)$, where x is the position and t is the time, the elements c_I (where I is a subset of the species labels) have initial conditions

$$c_I(x,0) = \begin{cases} c_I^0 & \text{if } x \in \mathcal{R} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In the case of DNA strand displacement, we supplied gates and auxiliary strands uniformly in space. In vDSD, the species in the subset I can be initialised

[1] Appendices in the online version, available from the author's website.

according to (2) with $\mathcal{R}$ as a central channel in 1d or a central square in 2d, by using the `centralcore` option of the `spatialic` directive (Table S1).

3 Results

To predict how DNA diffusion might enable pattern formation over macroscopic scales, we considered three circuits that have previously been well-studied at the CRN level: an autocatalytic circuit, a Lotka-Volterra predator-prey oscillator and a consensus network. We implemented DNA strand displacement versions of these circuits, and analysed their spatiotemporal dynamics with vDSD.

3.1 Wave Propagation in an Autocatalytic Circuit

One of the simplest known systems for which travelling wave phenomena have been observed is an autocatalytic reaction [25]

$$X + Y \xrightarrow{k} 2Y \tag{3}$$

The distance travelled in time t by the front of a travelling wave in an autocatalytic reaction-diffusion system is $d_W \approx 2\sqrt{fDt}$ [26], where f is the *intensity* of the interaction. In this case f becomes approximately equal to $k[X]_0$, where $[X]_0$ is the initial concentration of X (see Appendix B for a derivation). The diffusion coefficient of a single-stranded 20mer in water at 20°C was reported to be on the order of 10^{-10} m^2 s^{-1} [24]. Furthermore, the concentrations of DNA species in strand displacement systems are typically on the order of 10 nM – 1 μM. For a bimolecular rate constant $k = 5 \times 10^{-5}$ M^{-1} s^{-1} and an initial concentration of $[X]_0 = 5$ nM, the distance travelled by a wave front in 40 hours is approximately 4.55 cm, which agreed with simulations (Fig. 3a,e).

Several previous works have demonstrated the implementation of autocatalytic behaviours using DNA strand displacement [17, 27]. However, the spatiotemporal dynamics of this simple circuit has not yet been reproduced experimentally. While the wave-speed of the single reaction can be analytically characterised, the impact of diffusion and consumption of ndsDNA gates can only be predicted via numerical simulation. To address this, we compared simulations of the high-level autocatalytic reaction (3) in one spatial dimension with equivalent simulations of strand displacement implementations. Here we use a model of the autocatalytic $B + X \rightarrow 2X$ circuit that is identical to the model in [17] (Fig. 2).

We analysed spatiotemporal dynamics of the autocatalytic circuit from [17]. In an ideal parameterisation, in which all rates of toehold-mediated strand displacement are equal, we observed a very similar pattern to the CRN model (Fig. 3a,b,e). When using the experimentally calibrated rates from [17], we also found a similar pattern in the absence of leaks (Fig. 3c,e). However, approximately 1% leak from the Join$_{\mathrm{BX}}$ gate completely disrupted the travelling wave (Fig. 3d). The strong impact of leaks on spatiotemporal dynamics is analogous

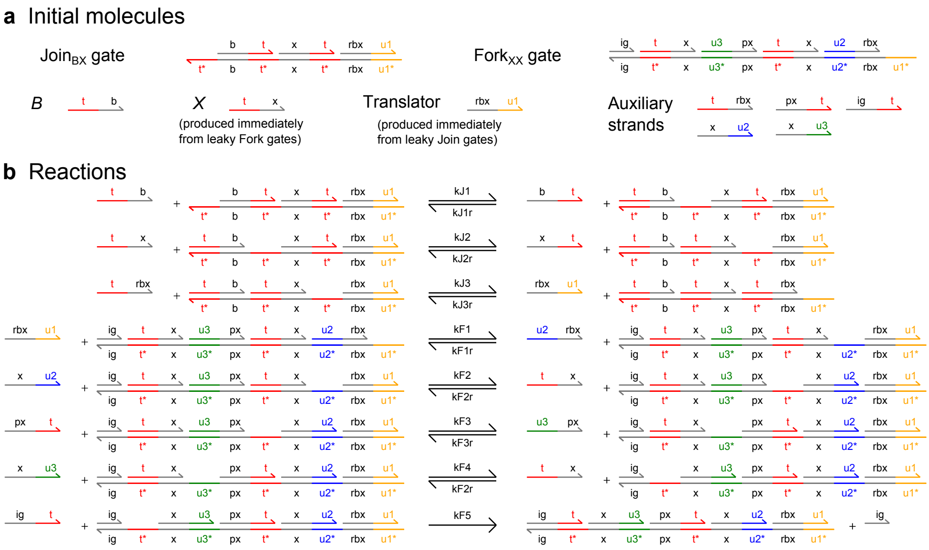


Fig. 2. DNA strand displacement implementation of an autocatalytic reaction. Two-domain signals and gates were used to implement the formal reaction $B + X \rightarrow 2X$, where B is the ⟨t b⟩ strand, and X is the ⟨t x⟩ strand. (a) The initial molecules are shown, including products of fast leak reactions, which are assumed to immediately produce gate outputs. (b) Complete reaction list. The reaction rates are quantified in [17], in which each rate was inferred from experimental data.

to its impact on purely temporal dynamics of autocatalytic circuits, observed in [17, 27], in which the presence of a small concentration of reactant kick-starts autocatalysis and eventually consumes all available gates. Here, as there are gates across the whole spatial domain, the presence of leaks means that diffusion from neighbouring locations is not required to kick-start autocatalysis, and so a travelling wave is not observed.

3.2 Periodic Travelling Waves in Lotka-Volterra Predator-Prey Oscillators

To analyse more complex spatiotemporal dynamics in an equivalent experimental setup, we considered designs for generating *periodic* travelling waves. Mathematical theory dictates that periodic travelling waves can be produced in spatially heterogeneous settings (i.e. with diffusion) when the corresponding spatially homogeneous dynamics show oscillations [2]. A range of CRN oscillators have been studied, providing an extensive set of examples to test. Among them, the Lotka-Volterra system is a canonical example of nonlinear dynamics, which has also been studied in the context of DNA strand displacement [15]. A Lotka-Volterra network can be described by

$$X \xrightarrow{k_1} 2X \qquad X + Y \xrightarrow{k_2} 2Y \qquad Y \xrightarrow{k_3} \emptyset \tag{4}$$

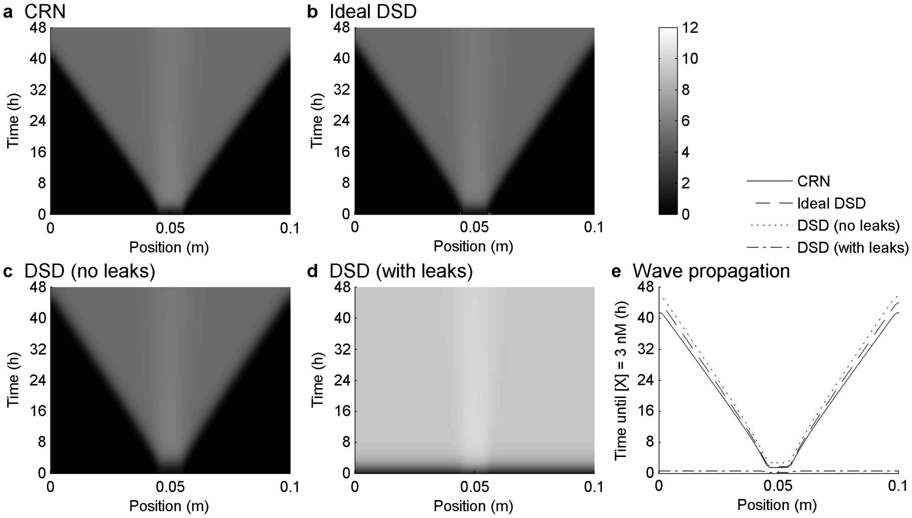


Fig. 3. Wave propagation in autocatalytic circuits. Simulations were carried out using vDSD for autocatalytic circuits. (**a**) CRN model with $k = 5 \times 10^{-5}$ nM^{-1} s^{-1}. (**b**) *Ideal* strand displacement implementation, in which all toehold-mediated strand displacement reactions occur at rate 10^{-4} nM^{-1} s^{-1}. (**c,d**) Strand displacement implementation of $B + X \rightarrow 2X$ in [17], where rates of toehold-mediated strand displacement were set to values inferred from experimental data. Leak parameters were either equal to zero (c), or at inferred quantities (d). In all cases, simulations were initialised with 1 nM of X in a central channel of width 0.01 m, and 5 nM of Y across the whole 0.1 m domain. In strand displacement models, 200 nM of gates and auxiliary strands were also supplied homogeneously. The concentration of X is indicated by the colour bar in the upper right, in nM units. (**e**) Wave propagation was characterised by determining the time at which the concentration of X reached 3 nM, half of the maximal level in the ideal system, for each point in space.

DSD Implementation of a Lotka-Volterra Predator-Prey System. To assess experimentally feasible designs for Lotka-Volterra oscillators, we translated the reactions (4) into a two-domain DNA strand displacement system (Fig. 4). In this design, the majority of toeholds are the same (domain t), though a second toehold sequence (domain u) specifically distinguishes translator strands from other single-stranded DNA strands in the system. A single toehold scheme is presented in Appendix C, where it is noted that auxiliary strand reuse (crosstalk) between gates complicates the selection of their initial concentrations.

Not all auxiliary strand crosstalk is removed in this design, as the $\langle \mathtt{x}\ \mathtt{t} \rangle$ strand required to release the second output on the Fork$_{2X}$ gate is also a *reverse* strand on Join$_{XY}$ gates. Thus, the effective rate constrant for $X + Y \rightarrow 2Y$ may increase as Fork$_{2X}$ gates are consumed. An optimisation of this design would therefore be to remove this crosstalk.

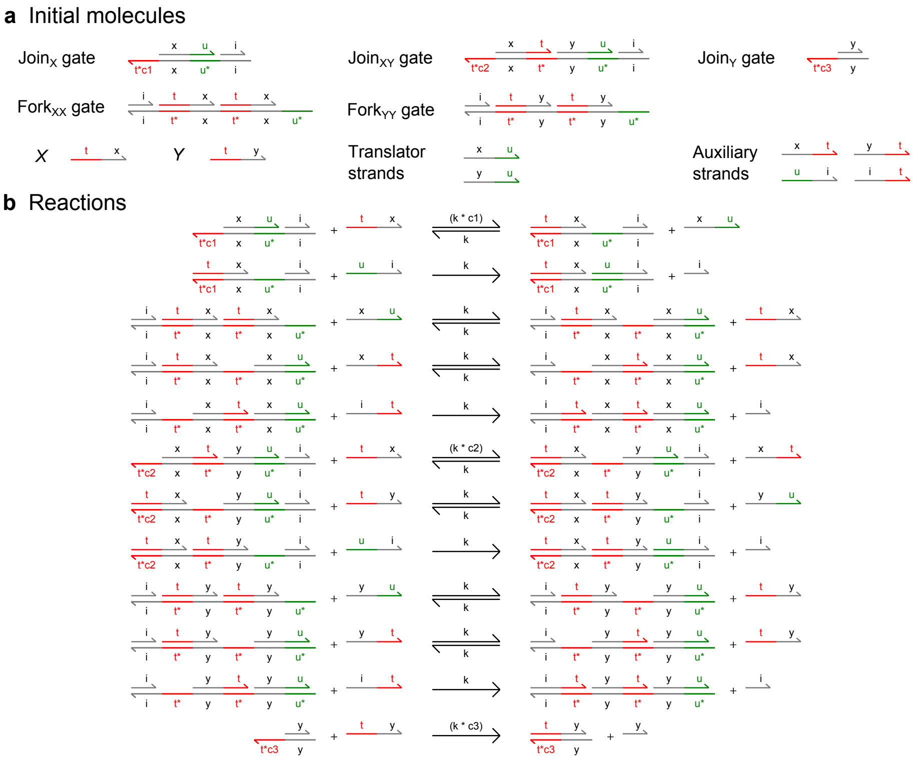


Fig. 4. Candidate DNA strand displacement implementation of a Lotka-Volterra oscillator. Two-domain signals and gates were used to implement the reaction system (4), where X is the $\langle \texttt{t x} \rangle$ strand and Y is the $\langle \texttt{t y} \rangle$ strand. (**a**) The initial molecules required. (**b**) Complete reaction list. The reaction rates are all set as `k` except for the binding of the first input strands to Join gates, which include a coefficient representing possible manipulations to the degree of complementarity. Specifically, $k_{\text{JoinX}} = \texttt{c1*k}$, $k_{\text{JoinXY}} = \texttt{c2*k}$, and $k_{\text{JoinY}} = \texttt{c3*k}$.

Generating Oscillatory Behaviours by Modulating External Toehold Binding Rates. In order to generate periodic travelling waves, we sought conditions under which DSD versions of the Lotka-Volterra reactions gave oscillatory behaviours. Despite the potential for modulating effective rate constants by varying concentrations of auxiliary strands, we found no regime in which more than a single oscillation could be observed (simulations not shown). Instead, we considered modifications to the binding sites of inputs on the join gates, in order to obtain direct control over the effective rate constant. By shortening the sequence of the exposed toehold and thus the degree of complementarity, strand displacement reactions can be slowed down by several orders of magnitude [28]. Manipulating the rate constants in this way is equivalent to the approach in [15], in which the first step of the implementation of the bimolecular reaction is 3 orders of magnitude slower than the unimolecular reactions.

To determine an appropriate parameter regime for oscillatory behaviours, we simulated combinations of $(k_{\mathrm{JoinX}}, k_{\mathrm{JoinXY}}, k_{\mathrm{JoinY}})$, where k_G is the binding of the input to gate $G \in \{\mathrm{JoinX}, \mathrm{JoinXY}, \mathrm{JoinY}\}$, and analysed the resulting traces for the number of turning points within 96 h. A fixed duration was important to rule out slow yet persistent oscillations, which would be challenging to observe experimentally. Also, as gates are consumed, both the amplitude and frequency of oscillations will diminish over time, meaning it was important to use a methodology suitable for damped oscillations. Reducing binding rates to Join_X and Join_Y gates led to a dramatic improvement in oscillatory behaviours (Fig. 5). A reduction of approximately 2 orders of magnitude was optimal (see leftmost panels of Fig. 5a,b), and produced up to 10 turning points in the 96 h window. Reducing k_{JoinXY} led to fewer turning points in both $\langle \mathtt{t\ x} \rangle$ and $\langle \mathtt{t\ y} \rangle$ traces (Fig. 5c,d). The strong dependency of oscillatory behaviours on specific values of the parameters emphasises the importance of accurate quantification of these rates.

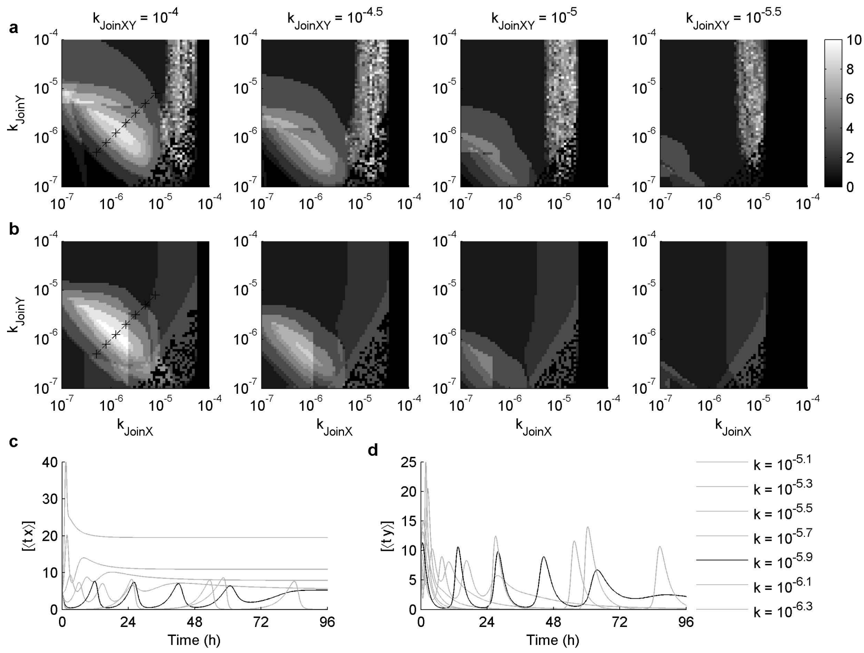


Fig. 5. Parameter analysis for the proposed DSD implementation of a Lotka-Volterra oscillator. Simulations were performed over different combinations of rates for inputs binding join gates. The simulations of (**a**) $\langle \mathtt{t\ x} \rangle$ and (**b**) $\langle \mathtt{t\ y} \rangle$ were analysed for the number of turning points in 96 h. (**c**,**d**) Example simulations corresponding to the black crosshairs in the leftmost panels of a and b. Here, $k_{\mathrm{JoinXY}} = 10^{-4}\ \mathrm{nM}^{-1}\mathrm{s}^{-1}$, the values for both k_{JoinX} and k_{JoinY} are as indicated in the legend, and all other rates of toehold-mediated strand displacement were assumed to be $10^{-4}\ \mathrm{nM}^{-1}\mathrm{s}^{-1}$. All gates and auxiliary strands were initialised at 200 nM, and $\langle \mathtt{t\ x} \rangle$ and $\langle \mathtt{t\ y} \rangle$ were initialised at 10 nM.

Spatiotemporal Dynamics for Strand Displacement Oscillators. Having established parameter regimes for DNA strand displacement implementations in which oscillatory behaviour persists over 96 hours, we sought to determine whether periodic travelling wave phenomena could be produced. In addition to the consumption of gates diminishing oscillatory behaviour, we would expect the diffusion of all DNA molecules to spatially homogenise behaviours that are more heterogeneous in the CRN version. We simulated invasions of ⟨t x⟩ and ⟨t y⟩ strands in media containing the gates and auxiliary strands. We selected a parameterisation in which non-spatial simulations predicted robust cycling with minimal amplitude decay (Fig. 6a). In 1d, we observed periodic travelling waves, though the waves both decayed in amplitude and decoupled from cycles in the centre (Fig. 6b). In 2d, we observed a similar pattern, with the emergence of several travelling waves with decaying amplitude (Fig. 6c).

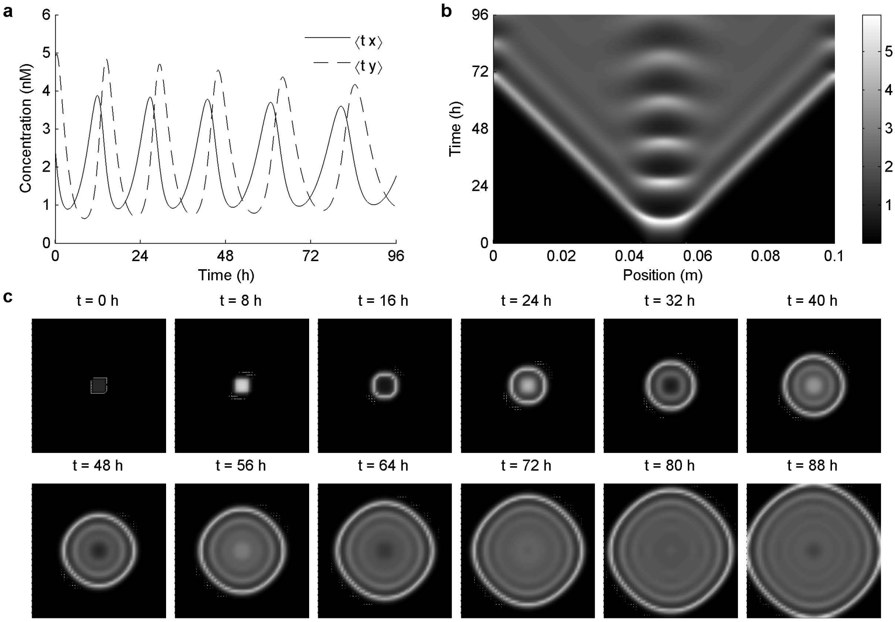


Fig. 6. Spatiotemporal dynamics of the proposed DSD implementation of a Lotka-Volterra oscillator. This DSD implementation was simulated in vDSD using parameter values identified in Fig. 5 as leading to oscillatory dynamics. Specifically, $k_{\mathrm{JoinX}} = k_{\mathrm{JoinY}} = 2 \times 10^{-6}$ nM^{-1} s^{-1}, and all other toehold-mediated strand displacement rates at $k = 10^{-4}$ nM^{-1} s^{-1}. (**a**) Simulation of the spatially inhomogeneous problem. (**b**) Simulation of spatiotemporal dynamics in 1d. (**c**) Simulation of spatiotemporal dynamics in 2d. In b,c, the domain was 0.1 m wide, and the solver used 101 grid-points. A central core of relative width 0.1 was applied to ⟨t x⟩ and ⟨t y⟩, 1 nM internally and 0 nM externally. All gates and auxiliary strands were supplied uniformly at 200 nM. In b and c, the concentration of ⟨t x⟩ is indicated by the colour bar in the upper right, in nM units.

Similar analysis applied to the single toehold DSD implementation revealed greater amplitude loss in non-spatial simulations, and equivalently weaker travelling waves in both 1d and 2d simulations (Fig. S3).

3.3 Emergence of Stationary Patterns from a Consensus Network

In previous sections, we predicted that non-stationary spatial patterns could be generated from invasion scenarios. However, owing to technical challenges in producing such initialisations in an experiment, here we considered pattern formation that only relies on inherent random spatial heterogeneity. We considered circuits with bistable dynamics, which might give rise to spatial bistability, and thus stationary patterns. Consensus algorithms use bistability to enable distributed agents holding a mixture of beliefs to reach consensus. For a binary belief, X or Y say, it is possible for a population of N agents to reach consensus on the initial majority in $O(N \log N)$ steps [29]. There are several ways to describe the algorithm in terms of chemical reaction networks (CRNs). Here, we consider the three reaction scheme

$$X + Y \xrightarrow{k} 2B \qquad B + X \xrightarrow{k} 2X \qquad B + Y \xrightarrow{k} 2Y, \tag{5}$$

as previously implemented using two-domain strand displacement [17]. This scheme was also used in a recent analysis of consensus algorithms in a spatial context [30].

Simulation of Spatiotemporal Dynamics for the Consensus Network. To gain insight into how diffusion of DNA molecules might modulate consensus dynamics in a spatially heterogeneous experimental system, we simulated consensus networks in vDSD. As with the autocatalytic circuits in Section 3.1, we compared the spatiotemporal dynamics of the CRN level model with those from strand displacement-level models. To encode the more realistically achievable experimental setting of providing a target concentration uniformly in space, but subject to spatial variations, we used the *random* option of the spatial initial condition directive (see Table S1).

We found that consensus networks routinely produced spatial patterns in 1d within 2 days (Fig. 7). As predicted from theory [31, 32], the CRN model produced a variety of patterns that depended on the random initial configuration of the concentrations of X and Y molecules, and were demonstrably stable in time (Fig. 7a). Furthermore, these patterns were reasonably robust to deviations from a zero majority, as seen by varying $[X]$ both above and below the value of $[Y]$ (Fig. 8). We next simulated the DSD version of the consensus network, described in [17]. This network has a majority threshold that is not 1:1, owing to differences in the effective rate constants of the two autocatalytic reactions. Simulations of random perturbations to $[X] = 7.5$ nM and $[Y] = 5$ nM led to the emergence of patterns that persisted in excess of 10 days, similar to the CRN model, both in the absence (Fig. 7b) and presence (Fig. 7c) of fast leaks arising from dysfunctional dsDNA gates or ssDNA strands. The presence of leaks increased the maximum concentration of $[X]$ considerably as compared with the

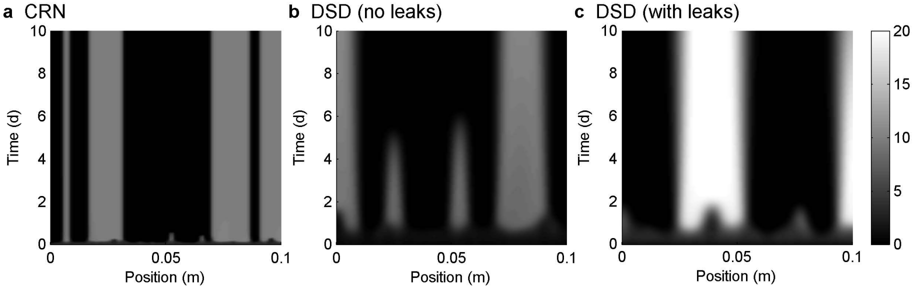


Fig. 7. Pattern formation in 1d for diffusive consensus networks. Simulations were carried out using vDSD for 1d domains. (**a**) The CRN version of the consensus network (5) was simulated with $[X]$ and $[Y]$ initialised with random perturbations to 5 nM of strength 0.2 (see Appendix A) at each grid-point. (**b**) DSD version of the consensus network assuming no leaks. (**c**) DSD version of the consensus network assuming leak parameters as quantified in [17]. The concentration of X is indicated by the colour bars on the right, in nM units.

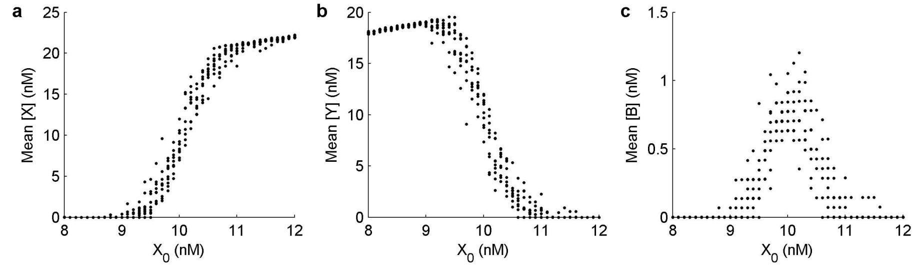


Fig. 8. Robustness of stationary patterns to initial conditions. The reaction-diffusion equations were solved for the CRN version on a 0.1 m 1d grid of 101 points, analogous to Fig. 7. Using the *random* option for the spatialic directive, initial conditions were random perturbations from $[Y] = 10$ nM and $[X]$ as indiciated on the horizontal axes. Simulations were conducted 10 times each with $[X]$ initialised over 0.1 nM increments between 8 nM and 12 nM. After 20 days of simulation, the mean average of concentrations for (**a**) X, (**b**) Y and (**c**) B was computed over the domain. When the mean concentration of X is 0, this indicates that no stationary pattern was observed, and similarly when the mean of $[X]$ is equal to $10 + X_0$. Mean concentrations in between these values indicate the presence of stationary patterns. This illustrates how robustly deviations from the unstable coexistence state can be stabilised by diffusion.

CRN or DSD system without leaks, though gave qualitatively similar patterns in the same spatial domain.

Spatial patterns were also observable in 2d domains (Fig. 9). Starting from a random initial configuration of the concentrations of X and Y molecules, patterns with high amplitude emerged within 1 day of simulated time in CRN (Fig. 9a), and DNA strand displacement (Figs. 9b,c) models. As for 1-dimensional simulations, leaks did not change the qualitative behaviour of the patterns,

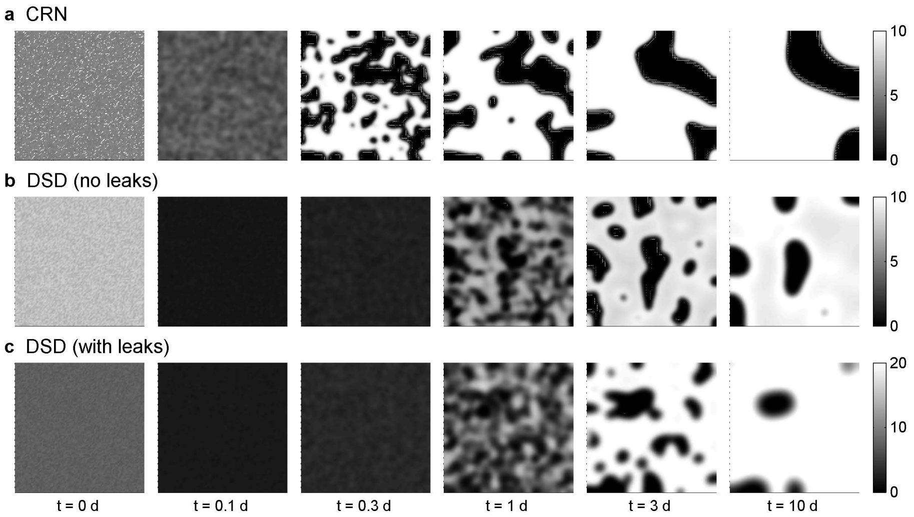


Fig. 9. Pattern formation in 2d for diffusive consensus networks. Simulations were carried out using vDSD for 2d domains. (**a**) The CRN version of the consensus network (5) was simulated with $[X]$ and $[Y]$ initialised with random perturbations to 5 nM at each grid-point. (**b**) DSD version assuming no leaks. (**c**) DSD version assuming leak parameters as described and quantified in [17]. The concentration of X is indicated by the colour bars on the right, in nM units.

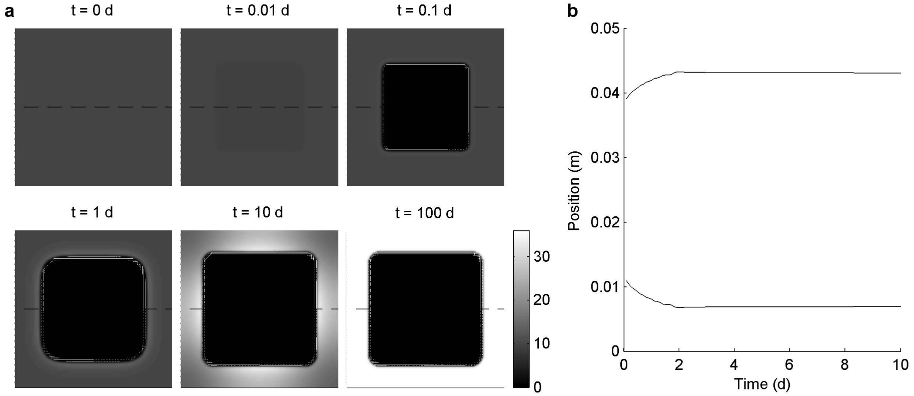


Fig. 10. Stationary pattern for the consensus network in a 2d domain. The CRN version of the consensus network (5) was simulated using vDSD, using the CRN tab, on a 0.05 m x 0.05 m grid with 101 points in each dimension. The initial conditions were set to $[Y]$ = 10 nM uniformly, and $[X]$ = 100 nM in a central square of size 0.025 m x 0.025 m. (**a**) The time-evolution of $[Y]$ at times specified above each panel. (**b**) The horizontal position at which $[Y]$ = 5 nM at each time-slice, at the middle of the vertical axes (indicated by dashed lines in a).

though increased the maximum concentration of X strands throughout. However, in contrast to 1d, we found that the patterns gradually disappeared over time (up to 10 days from the start of the simulated experiment). This might be due to an increased directional effect of diffusion in 2d, though could not be corrected for by reducing the rate of diffusion (simulations not shown). However, we found that stationary patterns could arise in 2d problems in invasion scenarios, in which invading species X were placed at the centre of a spatially homogeneous concentration of Y (Fig. 10). In simulations of the CRN model, a wave separating regions of high $[X]$ from high $[Y]$ transiently moved position, though became spatially stable by 2 days of simulated time (Fig. 10). These contrasting behaviours emphasise the importance of initial conditions on pattern formation in an experimental setting.

4 Discussion

In this article, we have shown that purely DNA-based circuits have great potential for producing spatial patterns. By using an experimentally demonstrable strategy for mapping CRNs to DNA strand displacement circuits, we provide realistic analysis of emergent spatiotemporal dynamics in the presence of isothermal diffusion. Specifically, we find that strand displacement realisations of autocatalytic circuits are particularly sensitive to leaks, which completely disrupt wave propagation. Additionally, diffusion degrades periodic travelling waves emanating from a strand displacement realisation of the Lotka-Volterra predator-prey network. However, we find that the consensus network in [17] produces stationary and travelling waves in different scenarios, both of which are robust to the presence of leaks. More generally, our simulations demonstrate that it is possible to exploit chemical diffusion to produce emergent pattern formation at the centimetre scale. The formation of patterns at this scale could open up new opportunities for programmed self-assembly.

The ability to design robust DNA-based circuits in the presence of unplanned interactions is a fundamental problem in molecular programming. While we expect DNA synthesis methods to improve over time, there will likely always be some imperfections. Therefore, the ability to both experimentally characterise and model leaks will continue to be important for fine-tuning DNA circuits. We found that leaks had varying impacts on emergent pattern formation. Autocatalytic circuits were particularly sensitive (Fig. 3a), while the consensus network was qualitatively robust (Figs. 7 & 9).

To experimentally observe the spatial patterns presented here, we note several challenges. To reproduce the scenarios in Figs. 3 and 6b,c, input DNA strands will need to be applied in a specific sub-region of the media. However, pipetting will likely result in a disturbance that will produce a travelling wave in the fluid, which might disrupt the diffusion of the DNA molecules within the solution. Therefore, pattern formation arising from more spatially uniform initial conditions is likely to be easier to achieve. In Figs. 7 and 9, initial conditions were set to be random perturbations from a target concentration, which models the inevitable spatial heterogeneity in a mixture.

Our approach to analysing DNA strand displacement circuits in the presence of diffusion has involved the extension of the Visual DSD software to describe and simulate reaction-diffusion equations. This has allowed us to conveniently analyse the dependence of spatiotemporal dynamics on different DNA gate designs, unplanned DNA interactions, fluctuations in initial conditions and kinetic rates. Future extensions could include the specification of irregular domains and a range of boundary conditions, which could be important for applications of DNA-based patterning or for describing experimental systems such as microchemostats, which involve measuring molecular interactions in chambers subjected to fluxes of reactants.

It is hoped that our methodology and corresponding software implementation will provide the means to design strand displacement circuits that both acknowledge and take advantage of the inherent spatial heterogeneity in physical systems.

Acknowledgements. The authors would like to thank David Soloveichik (UCSF), Rebecca Schulman (John Hopkins), Boyan Yordanov, Niall Murphy and Luca Cardelli (Microsoft Research) for advice on the manuscript. We also thank Filippo Polo and Colin Gravill for assistance with software development for Visual DSD. This work was supported through the NSF grant NSF CCF-1162141 to GS.

References

1. Hagan, M.F., Chandler, D.: Dynamic pathways for viral capsid assembly. Biophysical J. 91, 42–54 (2006)
2. Murray, J.D.: Mathematical biology. Springer (2003)
3. Sheth, R., Marcon, L., Bastida, M.F., Junco, M., Quintana, L., Dahn, R., Kmita, M., Sharpe, J., Ros, M.A.: Hox genes regulate digit patterning by controlling the wavelength of a turing-type mechanism. Science 338, 1476–1480 (2012)
4. Reif, J., Chandran, H., Gopalkrishnan, N., LaBean, T.: Self-assembled DNA nanostructures and DNA devices. In: Cabrini, S., Kawata, S. (eds.) Nanofabrication Handbook, pp. 299–328. CRC Press, Taylor and Francis Group, New York (2012)
5. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature 394, 539–544 (1998)
6. Um, S.H., Lee, J.B., Park, N., Kwon, S.Y., Umbach, C.C., Luo, D.: Enzyme-catalysed assembly of DNA hydrogel. Nat. Mater. 5, 797–801 (2006)
7. Zaikin, A., Zhabotinsky, A.: Concentration wave propagation in two-dimensional liquid-phase self-oscillating system. Nature 225, 535–537 (1970)
8. Bauer, G., McCaskill, J., Otten, H.: Traveling waves of in vitro evolving RNA. Proc. Natl. Acad. Sci. 86, 7937–7941 (1989)
9. Isalan, M., Lemerle, C., Serrano, L.: Engineering gene networks to emulate Drosophila embryonic pattern formation. PLoS Biology 3, e64 (2005)
10. Simpson, Z.B., Tsai, T.L., Nguyen, N., Chen, X., Ellington, A.D.: Modelling amorphous computations with transcription networks. J. R. Soc. Interface 6(suppl. 4), S523–S533 (2009)
11. Padirac, A., Fujii, T., Estvez-Torres, A., Rondelez, Y.: Spatial waves in synthetic biochemical networks. J. Am. Chem. Soc. 135, 14586–14592 (2013)

12. Chirieleison, S.M., Allen, P.B., Simpson, Z.B., Ellington, A.D., Chen, X.: Pattern transformation with dna circuits. Nat. Chem. 5, 1000–1005 (2013)
13. Allen, P.B., Chen, X., Simpson, Z.B., Ellington, A.D.: Modeling scalable pattern generation in dna reaction networks. Natural Computing, 1–13 (2012)
14. Scalise, D., Schulman, R.: Designing modular reaction-diffusion programs for complex pattern formation. Technology 2, 55–66 (2014)
15. Soloveichik, D., Seelig, G., Winfree, E.: Dna as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. 107(12), 5393–5398 (2010)
16. Cardelli, L.: Two-domain DNA strand displacement. Math. Struct. Comput. Sci. 23(02), 247–271 (2013)
17. Chen, Y.J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from DNA. Nat. Nanotechnol. 8, 755–762 (2013)
18. Phillips, A., Cardelli, L.: A programming language for composable DNA circuits. J. R. Soc. Interface 6 (suppl. 4), S419–S436 (2009)
19. Lakin, M.R., Youssef, S., Polo, F., Emmott, S., Phillips, A.: Visual DSD: a design and analysis tool for DNA strand displacement systems. Bioinformatics 27, 3211–3213 (2011)
20. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science 332, 1196–1201 (2011)
21. Qian, L., Winfree, E., Bruck, J.: Neural network computation with DNA strand displacement cascades. Nature 475, 368–372 (2011)
22. Amir, Y., Ben-Ishay, E., Levner, D., Ittah, S., Abu-Horowitz, A., Bachelet, I.: Universal computing by DNA origami robots in a living animal. Nat. Nanotechnol. (2014)
23. Smith, G.D.: Numerical solution of partial differential equations: finite difference methods. Oxford University Press (1985)
24. Stellwagen, E., Lu, Y., Stellwagen, N.C.: Unified description of electrophoresis and diffusion for DNA and other polyions. Biochemistry 42, 11745–11750 (2003)
25. Merkin, J., Needham, D.: Propagating reaction-diffusion waves in a simple isothermal quadratic autocatalytic chemical system. J. Eng. Math. 23, 343–356 (1989)
26. Fisher, R.A.: The wave of advance of advantageous genes. Ann. Eugen. 7, 355–369 (1937)
27. Zhang, D.Y., Turberfield, A.J., Yurke, B., Winfree, E.: Engineering entropy-driven reactions and networks catalyzed by DNA. Science 318, 1121–1125 (2007)
28. Yurke, B., Mills Jr., A.P.: Using DNA to power nanostructures. Genet. Program. Evol. M 4, 111–122 (2003)
29. Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. Distrib. Comput. 21, 87–102 (2008)
30. Lakin, M.R., Stefanovic, D.: Pattern formation by spatially organized approximate majority reactions. In: Ibarra, O.H., Kari, L., Kopecki, S. (eds.) UCNC 2014. LNCS, vol. 8553, pp. 254–266. Springer, Heidelberg (2014)
31. Gardner, R.A.: Existence and stability of travelling wave solutions of competition models: A degree theoretic approach. J. Differential Equations 44, 343–364 (1982)
32. Kan-on, Y.: Global bifurcation structure of positive stationary solutions for a classical Lotka-Volterra competition model with diffusion. Japan J. Indust. Appl. Math. 20(3), 285–310 (2003)

Output Stability and Semilinear Sets in Chemical Reaction Networks and Deciders

Robert Brijder

Hasselt University and Transnational University of Limburg, Belgium
robert.brijder@uhasselt.be

Abstract. We study the set of output stable configurations of chemical reaction deciders (CRDs). It turns out that CRDs with only bimolecular reactions (which are almost equivalent to population protocols) have a special structure that allows for an algorithm to efficiently calculate the (finite) set of minimal output stable configurations. As a consequence, a relatively large sequence of configurations may be efficiently checked for output stability.

We also provide a number of observations regarding the semilinearity result of Angluin et al. [Distrib. Comput., 2007] from the context of population protocols (which is a central result for output stable CRDs). In particular, we observe that the computation-friendly class of totally stable CRDs has equal expressive power as the larger class of output stable CRDs.

1 Introduction

In scenarios where the number of molecules in a chemical reaction network (CRN) is small, traditional continuous models for CRNs based on mass action kinetics are not suitable and one may need to consider discrete CRNs. In discrete CRNs, the number of molecules of each species is represented by a nonnegative integer and probabilities are assigned to each reaction. The computational power of discrete CRNs has been formally studied in [16] (see also [7]), where it is shown that Turing-universal computation is possible with arbitrary small (but nonzero) error probability. The implementability of arbitrary CRNs has been studied using strand displacement reactions as a primitive [17]. As observed in [16], discrete CRNs are similar to population protocols [1,4] and results carry over from one domain to the other. From now on we consider only discrete CRNs, and so we omit the adjective "discrete".

We continue in this paper the study of CRNs that has for each given input a deterministic output [5]. Thus, we are concerned here with error-free computation and so probabilities are irrelevant and only reachability is important. A given input is accepted by such a "deterministic" CRN, or more precisely *output stable chemical reaction decider* (CRD) [5], if at the end of the "useful" computation we obtain an accept configuration c, which is a configuration where at least one yes voter is present and none of the no voters (each species is marked by the CRD as either a yes or a no voter). Otherwise, the input is rejected and

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 100–113, 2014.

c is a reject configuration, which is a configuration where at least one no voter is present and none of the yes voters. The configuration c may still change, but it stays an accept configuration when c is an accept configuration (and similar for reject). In this case c is called *output stable*.

In Section 3, we provide a number of observations regarding the semilinearity result for population protocols of [1,2]. First we mention that this result has a small gap in its proof which is easily fixable, except for the corner case where the semilinear set contains the zero vector. Next, we define a stricter variant of the notion of output stable, called *totally stable*. In contrast to output stable CRDs, totally stable CRDs eventually (completely) halt for every input. For totally stable CRDs it is computationally easy to determine when the computation has ended. We mention that the semilinearity result of [1,2] works also for totally stable CRDs, and consequently the class of totally stable CRDs has equal expressive power as the larger class of output stable CRDs.

CRNs are similar to Petri nets [14] and vector addition systems (VASs) [11], see [16]. However, Petri nets and VASs operate as “generators” where the computation starts in the given fixed starting configuration (called the initial marking) and one is (generally) interested in the reachable configurations. In contrast, a CRD is a decider where one is (generally) interested in determining the set of inputs that is accepted by the CRD. Despite these differences, various results concerning Petri nets and VASs can be carried over to CRDs.

In Section 4, we take a closer look at the notion of output stable. First, using some well-known results for VASs, we show that determining whether or not a configuration is output stable for an output stable CRD is decidable. Next, we turn to bimolecular CRNs, i.e., CRNs where each reaction has two reactants and two products. It turns out that bimolecular CRDs provide a special structure on the set of output stable configurations. More precisely, it turns out that the set of minimal elements M of the upward closed set of output unstable configurations may be efficiently determined for bimolecular CRDs, cf. Theorem 6 — this is the main result of the paper. Given M, it is then computationally easy to determine if a given configuration c is output stable. Consequently, the algorithm to determine M provides for an efficient method to test a relatively large number of configurations for output stability (the preprocessing cost to generate M becomes smaller, relatively, when testing more configurations for output stability).

Recent work related to CRNs include the calculus of chemical systems [15], the study of timing issues in CRNs [9], and the study of rate-independent continuous CRNs [6].

2 Chemical Reaction Networks and Deciders and Population Protocols

2.1 Chemical Reaction Networks

The notation and terminology of this subsection and the next are similar as in [10].

Let $\mathbb{N} = \{0, 1, \ldots\}$. Let $\varLambda$ be a finite set. The set of vectors over $\mathbb{N}$ indexed by $\varLambda$ (i.e., the set of functions $\varphi : \varLambda \to \mathbb{N}$) is denoted by $\mathbb{N}^{\varLambda}$. For $x \in \mathbb{N}^{\varLambda}$, we define $\|x\| = \sum_{i \in \varLambda} x(i)$. We denote the restriction of x to $\varSigma \subseteq \varLambda$ by $x|_{\varSigma}$. For $x, y \in \mathbb{N}^{\varLambda}$ we write $x \leq y$ iff $x(i) \leq y(i)$ for all $i \in \varLambda$. For notational convenience we now also denote vectors in $\mathbb{N}^{\varLambda}$, which can be regarded as multisets, by their string representations. Thus we denote $c \in \mathbb{N}^{\varLambda}$ by the string $A_1^{c(A_1)} \cdots A_n^{c(A_n)}$ (or any permutation of these letters) where $\varLambda = \{A_1, \ldots, A_n\}$.

Let $\varLambda$ be a finite set. A *reaction* α over $\varLambda$ is a tuple (r, p) with $r, p \in \mathbb{N}^{\varLambda}$; r and p are called the *reactants* and *products* of α, respectively. We say that α is *mute* if $r = p$. We say that α is *bimolecular* if $\|r\| = \|p\| = 2$. A *chemical reaction network* (CRN, for short) is a tuple $\mathcal{R} = (\varLambda, R)$ with $\varLambda$ a finite set and R a finite set of reactions over $\varLambda$. The elements of $\varLambda$ are called the *species* of $\mathcal{R}$. The elements of $\mathbb{N}^{\varLambda}$ are called the *configurations* of $\mathcal{R}$. For a configuration c, $\|c\|$ is the number of *molecules* of c.

For a $c \in \mathbb{N}^{\varLambda}$ and a reaction α over $\varLambda$, we say that $\alpha = (r, p)$ is *applicable* to c if $r \leq c$. If α is applicable to c, then the *result* of applying α to c, denoted by $\alpha(c)$, is $c' = c - r + p$. Note that $\alpha(c) \in \mathbb{N}^{\varLambda}$. In this case, we also write $c \to_{\alpha} c'$. Moreover, we write $c \to_{\mathcal{R}} c'$ if $c \to_{\alpha} c'$ for some reaction α of $\mathcal{R}$. The transitive and reflexive closure of $\to_{\mathcal{R}}$ is denoted by $\to_{\mathcal{R}}^{*}$. We say that c' is *reachable* from c in $\mathcal{R}$ if $c \to_{\mathcal{R}}^{*} c'$. If $\mathcal{R}$ is clear from the context, then we simply write $\to$ and $\to^{*}$ for $\to_{\mathcal{R}}$ and $\to_{\mathcal{R}}^{*}$, respectively.

We remark that a CRN is similar to a Petri net N [14] without the initial marking M: the set $\varLambda$ corresponds to the set of places of N and the set of reactions R corresponds to the set of transitions of N. While in a Petri net distinct transitions in N may correspond to a single reaction in R (i.e., there may be "copies" of each transition), this is irrelevant for our purposes.

A CRN is also similar to a vector addition system (VAS) [11]. A *VAS* V is a tuple $(\varLambda, S)$ with $\varLambda$ a finite set and S a finite subset of $\mathbb{Z}^{\varLambda}$. Again, the elements of $\mathbb{N}^{\varLambda}$ are the *configurations* of V. One is interested in the relation $\to$ over $\mathbb{N}^{\varLambda}$, where $c \to c'$ iff $c' = c + x$ for some $x \in V$. Reachability problems concerning CRNs can be straightforwardly translated to VASs (or Petri nets) and vice versa, see [16, Appendix A.6].

2.2 Chemical Reaction Deciders

A *(leaderless) chemical reaction decider* (CRD, for short) is a tuple $\mathcal{D} = (\varLambda, R, \varSigma, \varUpsilon)$, where $(\varLambda, R)$ is a CRN, $\varSigma \subseteq \varLambda$, $\varUpsilon : \varLambda \to \{0, 1\}$. The elements of $\varSigma$, $\varUpsilon^{-1}(0)$, and $\varUpsilon^{-1}(1)$ are called the *input species*, *no voters*, and *yes voters* of $\mathcal{D}$, respectively. Notation and terminology concerning CRNs carry over to CRDs. For example, we may speak of a configuration of $\mathcal{D}$. An *initial configuration* of $\mathcal{D}$ is a nonzero configuration c of $\mathcal{D}$ where $c|_{\varLambda \setminus \varSigma} = 0$ (by abuse of notation we denote the zero vector over suitable alphabet by 0). A CRD is called *bimolecular* if all reactions of R are bimolecular.

We define the following function $\varPhi_{\mathcal{D}} : \mathbb{N}^{\varLambda} \to \{0, 1, \mathrm{und}\}$. For $x \in \mathbb{N}^{\varLambda}$, let $I_x = \{S \in \varLambda \mid x(S) > 0\}$. Then, for $i \in \{0, 1\}$, we have $\varPhi_{\mathcal{D}}(x) = i$ iff both $I_x \cap \varUpsilon^{-1}(i) \neq \varnothing$ and $I_x \cap \varUpsilon^{-1}(1 - i) = \varnothing$ (as usual, $\varUpsilon^{-1}$ denotes the preimage of Υ). If x is zero or $I_x \cap \Upsilon^{-1}(0) \neq \varnothing \neq I_x \cap \Upsilon^{-1}(1)$, then $\Phi_{\mathcal{D}}(x) = \text{und}$. Here, the value und is regarded as "undefined".

A configuration c is called *totally stable* (*t-stable* for short) in $\mathcal{D}$ if both $\Phi_{\mathcal{D}}(c) \in \{0,1\}$ and, for all c' with $c \rightarrow^* c'$, we have $c' = c$. Note that if c is t-stable in $\mathcal{D}$, then for all c' with $c \rightarrow c'$, we have $c' = c$. A configuration c is called *output stable* (*o-stable* for short) in $\mathcal{D}$ if both $\Phi_{\mathcal{D}}(c) \in \{0,1\}$ and, for all c' with $c \rightarrow^* c'$, $\Phi_{\mathcal{D}}(c') = \Phi_{\mathcal{D}}(c)$. Note that every t-stable configuration is o-stable. A configuration that is not o-stable (t-stable, resp.) and nonzero is called *o-unstable* (*t-unstable*, resp.).

We say that $\mathcal{D}$ *o-stably decides* (*t-stably decides*, resp.) the function $\varphi : \mathbb{N}^{\Sigma} \setminus \{0\} \rightarrow \{0,1\}$ if for each initial configuration c of $\mathcal{D}$ and each configuration c' with $c \rightarrow^* c'$, we have $c' \rightarrow^* c''$ where c'' is o-stable (t-stable, resp.) in $\mathcal{D}$ and $\varphi(c|_{\Sigma}) = \Phi_{\mathcal{D}}(c'')$. In this case, we also say that $\mathcal{D}$ *o-stably decides* (*t-stably decides*, resp.) the set $\varphi^{-1}(1)$ and that $\mathcal{D}$ is *o-stable* (*t-stable*, resp.). Note that $\varphi^{-1}(1)$ along with the set Σ, uniquely determine φ. In [1] (and [10]), only o-stable CRDs are considered, and as a result the prefix *output* is omitted there.

Remark 1. We adopt here the definition of o-stably decides from [2, Section 2]. In the original definition of o-stably decides from [1], an initial configuration may be the zero vector and the domain of φ contains the zero vector. Since the zero vector corresponds to an input without any molecules and the number of molecules in a bimolecular CRD stays fixed, no molecule can be introduced and, in particular, none of the yes or no voters can be introduced. As a result, there exist no o-stable bimolecular CRDs when (strictly) using the definition of [1]. Finally, we remark that there are (leaderless) CRDs that are o-stable CRDs using the definition of [1], since we may then have reactions (r,p) with r the zero vector. However, it is easy to verify that these CRDs can only decide $\mathbb{N}^{\Sigma}$ or the empty set, and thus this notion is also not interesting for the (larger) class of CRDs.

2.3 Population Protocols

The notion of population protocol [1,4] is almost equivalent to the notion of bimolecular CRD. The only difference is that, in a population protocol, the set of reactions R is replaced by a *transition function* $\delta : \Lambda^2 \rightarrow \Lambda^2$. In this setting, $\delta(A,B) = (C,D)$ corresponds to the reaction (r,p) with $r = AB$ and $p = CD$ (recall that we may denote vectors by strings). Note that the tuples (A,B) and (C,D) are ordered. Note also that, for given $A,B \in \Lambda$, there are at most two non-mute reactions with A and B as reactants (since we have a transition for (A,B) and for (B,A)), while for bimolecular CRDs there can be arbitrary many such reactions.

Reactions, molecules, and species are called *transitions*, *agents*, and *states*, respectively, in the context of population protocols.

An important property of bimolecular CRDs is that the number of molecules stays fixed, i.e., if $c \rightarrow^* c'$, then $\|c\| = \|c'\|$.

Remark 2. In [1], $\delta(A, B) = (C, D)$ is interpreted as follows: a molecule of type A is transformed into a molecule of type C and simultaneously a molecule of type B is transformed into a molecule of type D. As a consequence, applying the "reaction" $\delta(A, B) = (B, A)$ would result in a different configuration. However, in [2] this interpretation is abandoned and $\delta(A, B) = (B, A)$ is considered a mute reaction. We adopt the convention of [2].

3 Semilinearity

In this section we state a number of modest, but useful, observations we made when studying the proof of the semilinearity result of [1].

Let Λ be a finite set. A set $S \subseteq \mathbb{N}^\Lambda$ is called *linear* (over Λ) if there are $v_0, \ldots, v_n \in \mathbb{N}^\Lambda$ such that $S = \{v_0 + \sum_{i=1}^{n} k_i v_i \mid k_i \in \mathbb{N}, i \in \{1, \ldots, n\}\}$. A set $S \subseteq \mathbb{N}^\Lambda$ is called *semilinear* (over Λ) if S is the union of a finite number of linear sets over Λ.

It is stated in [1] that every semilinear set S is o-stably decidable by a population protocol (i.e., a bimolecular CRD). While this result is often cited in the literature, it is straightforward to verify that the result fails if S contains the zero vector. Indeed, by definition semilinear sets may contain the zero vector, while the domain of φ in the above definition of stably decides is restricted to nonzero vectors (recall from Remark 1 that we have to use the definition of [2] instead of [1]). This small counterexample led us to revisit the proof of [1]. It turns out that Lemma 5 of [1] implicitly assumes that there are at least 2 agents (i.e., molecules), which translate into an initial configuration of size at least 2. Fortunately, this proof can be straightforwardly modified to allow for initial configurations of size 1, by letting, in [1, Lemma 5], I map σ_i to $(1, b, a_i)$ with $b = 1$ iff $a_i < c$ for case 1, and with $b = 1$ iff $a_i = c \mod m$ for case 2 (instead of to $(1, 0, a_i)$). In [2] (see also [3]), it is shown that if $S \subseteq \mathbb{N}^\Lambda$ is o-stably decidable by a population protocol, then S is semilinear. Thus we have the following (attributed, of course, to [1,2]).

Theorem 1 ([1,2]). *For every $S \subseteq \mathbb{N}^\Sigma$, S is o-stably decidable by a population protocol (i.e., a bimolecular CRD) iff S is both semilinear and does not contain the zero vector.*

As recalled in [5], the result from [2] that the sets o-stably decidable by population protocols are semilinear holds not only for population protocols, but for any reflexive and transitive relation $\rightarrow^*$ that respects addition (i.e., for $c, c', x \in \mathbb{N}^\Sigma$, $c \rightarrow^* c'$ implies $c + x \rightarrow^* c' + x$). Hence, Theorem 1 holds also for the (broader) family of all CRDs.

Another observation one can make when studying [1] is that the proof concerning o-stable CRDs holds unchanged for the smaller class of t-stable CRDs. By expressive power of a family $\mathcal{F}$ of CRDs we mean the family of sets decidable by $\mathcal{F}$. As the result follows from the proof of [1], we attribute it to [1].

Theorem 2 ([1]). *The family of t-stable bimolecular CRDs have equal expressive power as the family of o-stable CRDs. Equivalently, the sets that are t-stably* *decidable by bimolecular CRDs are precisely the semilinear sets without the zero vector.*

Proof. First recall, by the comment below Theorem 1, that the expressive powers of the families of o-stable CRDs and o-stable bimolecular CRDs are equal. Now, the family of t-stable bimolecular CRDs is a subset of the family of o-stable bimolecular CRDs. Thus it suffices to show that the if-direction of Theorem 1 holds for t-stable bimolecular CRDs.

The essential part of the if-direction of the proof of Theorem 1 above is Lemma 3 and Lemma 5 from [1]. In the proof of Lemma 5 in [1] a population protocol P is described that eventually reaches a configuration c which is called “stable” in [1], and which, in fact, is easily seen to be t-stable (by checking the three conditions of “stable” in [1]). The proof of Lemma 3 in [1] trivially holds for t-stable bimolecular CRDs. □

Since the bimolecular CRDs form a subset of the CRDs, Theorem 2 holds also when omitting the word “bimolecular”.

The family of t-stable CRDs form an interesting subclass of CRDs. Indeed, it is easy to verify, during a run of a t-stable CRD, whether or not a configuration is t-stable: one simply needs to verify whether or not there is an applicable (non-mute) reaction. In other words, it is easily verified whether or not the computation has ended. In the larger class of o-stable CRDs, it is not clear whether or not it is computationally easy to verify if a given configuration is o-stable or not. We revisit this latter problem in Section 4.

The concept of *CRDs with leaders* was introduced in [5] (it is simply called a CRD in [5]). The difference with (leaderless) CRDs is that for CRDs with leaders an additional vector $\sigma \in \mathbb{N}^{\Lambda \setminus \Sigma}$ is given and that the initial configurations c have the condition that $c|_{\Lambda \setminus \Sigma}$ is equal to σ (instead of equal to 0). Moreover, in the definition of o/t-stably decides the domain of the function φ is $\mathbb{N}^{\Sigma}$ instead of $\mathbb{N}^{\Sigma} \setminus \{0\}$. Using Theorem 1, we now straightforwardly observe that CRDs with leaders decide all semilinear sets.

Theorem 3 ([5]). *For every $S \subseteq \mathbb{N}^{\Sigma}$, S is o-stably decidable by a CRD with leaders iff S is semilinear.*

Proof. Again, by [2], every set o-stably decidable by a CRD with leaders is semilinear.

Conversely, let $S \subseteq \mathbb{N}^{\Sigma}$ be semilinear. Consider $\Sigma' = \{t\} \cup \Sigma$, where t is an element outside Σ. Let $S' = \{x \in \mathbb{N}^{\Sigma'} \mid x(t) = 1, x|_{\Sigma} \in S\}$. It is easy to verify that S' is semilinear. Indeed, let $v_0, \ldots, v_n$ be the vectors (cf. the definition of linear set) for one of the linear sets that together make up S. Then by adding an entry for t with value 1 for v_0 and value 0 for the other vectors, we see that the obtained vectors define a corresponding linear set for S'. Consequently, S' is semilinear. Note that S' does not contain the zero vector. By Theorem 1, there is a CRD $\mathcal{D} = (\Lambda, R, \Sigma', \Upsilon)$ that o-stably decides S'. Consider now the CRD $\mathcal{D}' = (\Lambda, R, \Sigma, \Upsilon, \sigma)$ with leaders where $\sigma \in \mathbb{N}^{\Lambda \setminus \Sigma}$ is such that $\sigma(t) = 1$ and $\sigma(i) = 0$ if $i \in \Lambda \setminus \Sigma'$. Consequently, the difference between $\mathcal{D}$ and $\mathcal{D}'$ is that index t is not part of the input species. Hence, $\mathcal{D}'$ o-stably decides S. □

Of course, (the proof of) Theorem 3 also holds by replacing o-stable by t-stable and/or replacing CRDs by bimolecular CRDs.

4 Determining the Output Stable Configurations

In this section we consider the problem of determining whether or not the "useful" computation of an o-stable CRD has ended. More precisely, we consider the problem of determining whether or not a given configuration of a o-stable CRD is output stable. Recall from the previous section that it *is* straightforward to determine whether or not a given configuration c is t-stable: one simply needs to check whether or not a non-mute reaction is applicable to c (and check that $\Phi_{\mathcal{D}}(c) \in \{0,1\}$).

Similar as done in [16, Theorem 4.2], we formulate now [11, Corollary 4.1] (defined in the context of VASs) in terms of CRNs.

Proposition 1 ([11]). *For given CRN $\mathcal{R}$ and configurations x, y of $\mathcal{R}$, it is decidable whether or not $x \to^* y'$ for some configuration $y' \geq y$.*

A much more involved result is known as the decidability of the reachability problem for vector addition systems, shown in [13] (see [12] for a simplified proof).

Proposition 2 ([13]). *For given CRN $\mathcal{R}$ and configurations x, y of $\mathcal{R}$, it is decidable whether or not $x \to^* y$.*

The precise complexity of the reachability problem of Proposition 2 is famously unknown (see, e.g., [12]).

By Propositions 1 and 2 we straightforwardly obtain the following result.

Theorem 4. *For a given o-stable CRD $\mathcal{D}$ and configuration c of $\mathcal{D}$, it is decidable whether or not c is o-stable in $\mathcal{D}$.*

Proof. Testing whether or not $\Phi_{\mathcal{D}}(c) \in \{0,1\}$ is clearly decidable. Let $\Phi_{\mathcal{D}}(c) = j$. Let, for $X \in \Lambda$, y_X be the configuration with $\|y_X\| = 1$ and $y_X(X) = 1$. By Proposition 1 it is decidable, for each $X \in \Upsilon^{-1}(1-j)$, whether or not there exists a c' such that $c \to^* c'$ and $c' \geq y_X$, i.e., $c'(X) > 0$. Hence if c contains only yes voters, then we can decide if there is a reachable configuration with no voters (and analogously if c contains only no voters). The only case left to decide is whether or not $c \to^* 0$ (again, 0 denotes the zero vector over Λ). By Proposition 2 it is decidable if the zero vector is reachable. Consequently, it is decidable if c is o-stable in $\mathcal{D}$. □

We now investigate more deeply some complexity issues involved to decide whether or not a configuration is o-stable. In fact, it turns out that bimolecular CRDs provide a convenient added "structure" for this problem.

Let $\mathcal{D}$ be an o-stable CRD. We now consider the set $U_{\mathcal{D}}$ of all output unstable configurations of $\mathcal{D}$. If $\mathcal{D}$ is clear from the context, then we simply write U for $U_{\mathcal{D}}$. We now recall a useful result from [2, Lemma 10]. For convenience, we also recall its short proof.

Proposition 3 ([2]). *Let $\mathcal{D}$ be an o-stable CRD. Then U is closed upward under $\leq$. In other words, for all $c, c' \in \mathbb{N}^{\Lambda}$ with $c \leq c'$, if $c \in U$, then $c' \in U$.*

Proof. Let $c \in U$ and $c \leq c'$. If $\Phi_{\mathcal{D}}(c) = \text{und}$, then c contains both yes and no voters (since $c \in U$, c is nonzero). Thus c' also contains both yes and no voters and we have $c' \in U$. Assume that $\Phi_{\mathcal{D}}(c) \in \{0,1\}$. If $\Phi_{\mathcal{D}}(c') = \text{und}$, then there is nothing to prove. Thus assume that $\Phi_{\mathcal{D}}(c) = \Phi_{\mathcal{D}}(c')$. Since $c \in U$, there is a c'' with $c \rightarrow^* c''$ with $\Phi_{\mathcal{D}}(c'') \neq \Phi_{\mathcal{D}}(c)$. Let $x := c' - c \in \mathbb{N}^{\Lambda}$. Then $c' = c + x \rightarrow^* c'' + x$ with $\Phi_{\mathcal{D}}(c'' + x) \neq \Phi_{\mathcal{D}}(c) = \Phi_{\mathcal{D}}(c')$ and $c' \in U$. □

Remark 3. In some papers, such as [5], not all species in CRDs need to be voters. In other words, in the definition of CRD we have $\Upsilon : E \rightarrow \{0,1\}$ for some $E \subseteq \Lambda$ (instead of $E = \Lambda$). We remark that Proposition 3 fails in this more general setting. Indeed, if nonzero c contains no voters, then $c \in U$, but by extending c with, say, a yes voter may result in an output stable configuration.

By Proposition 3, the set U is characterized by the set $\min(U)$ of minimal elements of U under $\leq$. By Dickson's lemma, recalled below, $\min(U)$ is a finite set.

Proposition 4 (Dickson's lemma [8]). *Let Λ be a finite set. Then for every $S \subseteq \mathbb{N}^{\Lambda}$, $\min(S)$ is finite.*

Given an o-stable CRD $\mathcal{D}$ *and* the set $\min(U)$, it is straightforward to verify if a given configuration c is o-stable in $\mathcal{D}$. Indeed, c is o-stable in $\mathcal{D}$ iff $u \not\leq c$ for all $u \in \min(U)$. Thus, to check whether or not c is o-stable in $\mathcal{D}$ takes $|\min(U)| \cdot |\Lambda|$ comparisons of molecule counts, which corresponds to a complexity of $O(|\min(U)| \cdot |\Lambda| \cdot \log(z))$-time, where z is the largest entry among the configurations in U (assuming the entries of a vector are encoded, say, in binary). Note that this complexity bound depends only on $\mathcal{D}$, i.e., it is independent of c.

We now show that $\min(U)$ can be efficiently determined when $\mathcal{D}$ is bimolecular. This is particularly useful when one wants to test for o-stability for some large (finite) set of configurations (instead of just a single configuration).

Let, for $k \geq 0$, $\mathcal{C}_{\leq k}$ ($\mathcal{C}_{=k}$, resp.) be the set of configurations $c \in \mathbb{N}^{\Lambda}$ with $\|c\| \leq k$ ($\|c\| = k$, resp.).

We remark that the naive approach to determine whether or not a particular configuration c is o-stable in a o-stable bimolecular CRD $\mathcal{D}$, would compute the set R_c of all configurations reachable from c and then verify that $\Phi_{\mathcal{D}}(c') = \Phi_{\mathcal{D}}(c)$ for all $c' \in R_c$. Note that $R_c \subseteq \mathcal{C}_{=k}$ with $k = \|c\|$ since $\mathcal{D}$ is bimolecular. Thus, in the worst case, one needs to compute in the order of $|\mathcal{C}_{=k}|$ configurations. The value of $|\mathcal{C}_{=k}|$ is equal to the number of multisets of cardinality k over Λ. This number (called figurate number, simplex number, or multiset coefficient), sometimes denoted by $\left(\!\binom{|\Lambda|}{k}\!\right)$, is equal to the binomial coefficient $\binom{|\Lambda|+k-1}{k}$, see, e.g., [18, Section 1.2].

First we prove a technical lemma. For $c, c' \in \min(U)$, denote $c \hookrightarrow_{\alpha} c'$ if $c \rightarrow_{\alpha} c' + b$ where $\alpha = (r, p)$ is a reaction and b is some configuration with $b \leq p$

and $b \neq p$. We write $c \hookrightarrow c'$ if $c \hookrightarrow_\alpha c'$ for some reaction α. It is important to realize that $\hookrightarrow$ is a relation on $\min(U)$. Again, the transitive closure of $\hookrightarrow$ is denoted by $\hookrightarrow^*$. For the next result, recall again that we may denote vectors by strings.

Lemma 1. *Let $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon)$ be an o-stable CRD. Let $M_1 = \{c \in \min(U) \mid \Phi(c) = \mathrm{und}\}$, $M_2 = \{c \in \min(U) \mid \Phi(c) \in \{0,1\}, c \rightarrow c'$ for some c' with $\Phi(c') \neq \Phi(c)\}$, and $T = \{r \mid (r,p) \in R, \Upsilon(A) \neq \Upsilon(B)$ for some $A, B \in \Lambda$ with $r(A) \neq 0 \neq p(B)\}$. We have the following.*

1. *If $\|r\| \neq 1$ for all $(r,p) \in R$, then $M_1 = \{AB \mid A, B \in \Lambda, \Upsilon(A) \neq \Upsilon(B)\}$.*
2. *If $\|p\| \neq 0$ for all $(r,p) \in R$, then $M_2 \subseteq T$.*
3. *If $\|r\| = 2$ for all $(r,p) \in R$, then $T \subseteq M_1 \cup M_2$.*
4. *If $c \rightarrow_\alpha c'$ for some $\alpha \in R$, $c \in \min(U)$, and $c' \in U$, then there is a $c'' \in \min(U)$ with $c \hookrightarrow_\alpha c''$.*
5. *If $\|r\| \geq \|p\|$ for all $(r,p) \in R$, then, for all $c \in \min(U)$, $c \hookrightarrow^* c'$ for some $c' \in M_1 \cup M_2$.*
6. *If $\mathcal{D}$ is bimolecular and $c \hookrightarrow c'$, then $\|c'\| = \|c\|$ or $\|c'\| = \|c\| - 1$.*

Proof. Since $\|r\| \neq 1$ for all reactions (r,p) of $\mathcal{D}$, the configurations of size 1 are o-stable, and so the elements of $\min(U)$ are of size at least 2. The nonzero configurations where $\Phi(c) = \mathrm{und}$ are those where there are $A, B \in \Lambda$ such that both $c(A) > 0$ and $c(B) > 0$, and $\Upsilon(A) \neq \Upsilon(B)$. The minimal such configurations are such that $c(A) = c(B) = 1$ and $c(X) = 0$ for all other species X, and since these configurations are of size 2, we obtain the first statement.

We now turn to the second statement. Let $c \in M_2$. Thus $c \in \min(U)$ with $\Phi(c) \in \{0,1\}$ and $c \rightarrow c'$ for some c' with $\Phi(c') \neq \Phi(c)$. Without loss of generality, assume that $\Phi(c) = 0$, i.e., c contains only no voters. Let $\alpha = (r,p)$ be the reaction of $\mathcal{D}$ such that $c \rightarrow_\alpha c'$. Since $\Phi(c') \neq \Phi(c)$, a yes voter has been introduced by α (c' cannot be the zero vector as $\|p\| \neq 0$). As $c \in \min(U)$, we have $c = r$. Also, we have $\Upsilon(A) = 0 \neq 1 = \Upsilon(B)$ for some $A, B \in \Lambda$ with $r(A) \neq 0 \neq p(B)$.

We turn to the third statement. Let $\alpha = (r,p)$ be a reaction of $\mathcal{D}$ such that $\Upsilon(A) \neq \Upsilon(B)$ for some $A, B \in \Lambda$ with $r(A) \neq 0 \neq p(B)$. Then $r \in U$ and since $\|r\| = 2$ we have $r \in \min(U)$ (note that the elements of $\min(U)$ are of size at least 2). Assume $r \notin M_1$, i.e., $\Phi(r) \in \{0,1\}$. Then $r \rightarrow_\alpha p$ with $\Phi(p) \neq \Phi(r)$ since $\Upsilon(A) \neq \Upsilon(B)$ for some $A, B \in \Lambda$ with $r(A) \neq 0 \neq p(B)$. Consequently, $r \in M_2$.

We now turn to the fourth statement. Let $\alpha = (r,p)$. Since $c \in \min(U)$, we have that $c - r = c' - p \notin U$. Since $c' \in U$, we have $c'' = c' - b \in \min(U)$ for some configuration $b \leq p$ and $b \neq p$. Therefore, $c \hookrightarrow_\alpha c''$.

We now turn to the fifth statement. If $c \in M_1$, then we are done. For all $c \in \min(U) \setminus M_1$, $c \rightarrow^* x \rightarrow y$ for some configurations x and y with $\Phi(x) \neq \Phi(y)$. For all such c, we assign the value (k,l) where $k = \|c\|$ and l is minimal such that $c \rightarrow^l x \rightarrow y$ for some configurations x and y with $\Phi(x) \neq \Phi(y)$ (by $\rightarrow^l$ we mean the l-th power of the relation $\rightarrow$). We show the result by induction on (k,l). If $l = 0$, then $c \in M_2$ and we are done. Assume $l > 0$. Then, by the fourth statement, $c \hookrightarrow c''$ and $c = c'' + b + r - p$. As $\|r\| \geq \|p\|$, we have $\|c''\| \leq \|c\|$. If $\|c''\| < \|c\|$, then, by the induction hypothesis, $c'' \hookrightarrow^* c'$ with $c' \in M_1 \cup M_2$ and so $c \hookrightarrow^* c'$. If $\|c''\| = \|c\|$, then $c'' \rightarrow^{l-1} x \rightarrow y$. This also leads, by the induction hypothesis, to $c'' \hookrightarrow^* c'$ with $c' \in M_1 \cup M_2$ and so $c \hookrightarrow^* c'$.

For the sixth statement, note that if $\mathcal{D}$ is bimolecular, then by the definition of the relation $\hookrightarrow$, we have $\|c'\| + \|b\| = \|c\|$ and $\|b\| < \|p\| = 2$. □

Lemma 1 above is key for Theorem 5 below. The strategy in the proof of Theorem 5 is to discover all elements of $\min(U)$ ordered by size: first all elements of $\min(U)$ of size k are computed, before any of the elements of $\min(U)$ of size $k+1$ are computed. This ensures that the generated candidates c can be tested for minimality in U, i.e., it can be tested whether or not $c \in \min(U)$. Otherwise, the number of generated candidates could potentially grow unbounded.

Theorem 5. *Let $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon)$ be an o-stable bimolecular CRD. Given $\mathcal{D}$, Algorithm 1 computes* $\min(U)$*.*

Proof. First, we initialize $M := M_1 \cup M_2 = M_1 \cup T$ with M_1, M_2, and T from Lemma 1, see Lines 3-4. The second (and final) phase is to iteratively augment M with the elements from $\min(U) \setminus (M_1 \cup M_2)$ as prescribed by Statements 5 and 6 of Lemma 1.

We show by induction that at Line 15, we have $M_{\text{it}} = \min(U) \cap \mathcal{C}_{=k}$ and $M = \min(U) \cap \mathcal{C}_{\leq k}$.

We first consider the basis case $k = 2$. Note that, by Lemma 1, $\min(U) \cap \mathcal{C}_{=2} = \min(U) \cap \mathcal{C}_{\leq 2}$ is obtained from $M_1 \cup M_2$ by adding all c' such that $c' \rightarrow^* c$ and $c \in M_1 \cup M_2$. Note that each such c' is minimal in U as $\|c'\| = 2$. This is accomplished in Lines 6-14.

We now consider the induction step. Let $k \geq 2$. Consider the set $X = \{c' \mid c' \rightarrow_\alpha c + B$, for some $\alpha \in R, c \in \min(U) \cap \mathcal{C}_{=k}, B \in \Lambda, c'' \not\leq c'$ for all $c'' \in \min(U) \cap \mathcal{C}_{\leq k}\}$, where we identify here $B \in \Lambda$ by the configuration b with $\|b\| = 1$ and $b(B) = 1$. Note that $X \subseteq U$. Since for all $c' \in X$, $\|c'\| = k+1$ and $c' \in U$, we have that $c'' \not\leq c'$ for all $c'' \in \min(U) \cap \mathcal{C}_{\leq k}$ iff $c'' \not\leq c'$ for all $c'' \in \min(U)$. Hence $X \subseteq \min(U) \cap \mathcal{C}_{=k+1}$. The set X is computed in Lines 16-21. Now, by Statements 5 and 6 of Lemma 1, $\min(U) \cap \mathcal{C}_{=k+1}$ is obtained from X by adding the configurations c' such that $c' \rightarrow^* c$ with $c \in X$ and $c'' \not\leq c'$ for all $c'' \in \min(U)$. Again, since $\|c'\| = k+1$ and $c' \in U$, we have that $c'' \not\leq c'$ for all $c'' \in \min(U)$ iff $c'' \not\leq c'$ for all $c'' \in \min(U) \cap \mathcal{C}_{\leq k}$. These additional configurations c' are (again) computed in Lines 6-14.

The algorithm halts as by Dickson's Lemma (Proposition 4), $\min(U)$ is finite. □

We now consider the time complexity of Algorithm 1.

Theorem 6. *Algorithm 1 computes* $\min(U)$ *in* $O(n \log^{|\Lambda| - \frac{1}{2}}(n) \cdot |R| \cdot |\Lambda|^2 \cdot \log(z))$ *time, where* $n = |\min(U)|$ *and z is the largest entry among the configurations in* $\min(U)$*.*

Algorithm 1. Generate the set M of minimal output unstable configurations of an o-stable bimolecular CRD $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon)$

```
 1: procedure GenMinUnstable($\mathcal{D}$)
 2:     $M \leftarrow \varnothing$
 3:     $M_{\text{it}} \leftarrow \{AB \mid A, B \in \Lambda, \Upsilon(A) \neq \Upsilon(B)\}$
 4:     $M_{\text{it}} \leftarrow M_{\text{it}} \cup \{r \mid (r,p) \in R, \Upsilon(A) \neq \Upsilon(B)$ for some $A, B \in \Lambda$ with $r(A) \neq 0 \neq p(B)\}$
 5:     while $M_{\text{it}} \neq \varnothing$ do
 6:         $M_{\text{new}} \leftarrow M_{\text{it}}$
 7:         while $M_{\text{new}} \neq \varnothing$ do
 8:             $M_{\text{old}}, M_{\text{new}} \leftarrow M_{\text{new}}, \varnothing$
 9:             for all $c \in M_{\text{old}}$, $\alpha \in R$ do
10:                 if $\exists\ c'$ with $c' \rightarrow_\alpha c$ and $c'' \not\leq c'$ for all $c'' \in M$ then
11:                     $M_{\text{new}}, M_{\text{it}}, M \leftarrow M_{\text{new}} \cup \{c'\}, M_{\text{it}} \cup \{c'\}, M \cup \{c'\}$
12:                 end if
13:             end for
14:         end while
15:         $\triangleright$ At this point $M = \min(U) \cap \mathcal{C}_{\leq k}$ and $M_{\text{it}} = \min(U) \cap \mathcal{C}_{=k}$ for some $k \geq 2$.
16:         $M_{\text{itold}}, M_{\text{it}} \leftarrow M_{\text{it}}, \varnothing$
17:         for all $c \in M_{\text{itold}}$, $\alpha \in R$, $B \in \Lambda$ do
18:             if $\exists\ c'$ with $c' \rightarrow_\alpha c + B$ and $c'' \not\leq c'$ for all $c'' \in M$ then
19:                 $M_{\text{it}}, M \leftarrow M_{\text{it}} \cup \{c'\}, M \cup \{c'\}$
20:             end if
21:         end for
22:     end while
23:     return $M$
24: end procedure
```

Proof. There are two inner loops. The first inner loop (at Lines 9-13) checks for every $c \in \min(U)$ and $\alpha \in R$, whether or not a $c' \rightarrow_\alpha c$ exists, and if such a c' exists, whether or not $c'' \not\leq c'$ for all $c'' \in \min(U) \cap \mathcal{C}_{\leq \|c'\|-1}$. The second inner loop (at Lines 17-21) checks for every $c \in \min(U)$, $\alpha \in R$, and $B \in \Lambda$, whether or not a $c' \rightarrow_\alpha c + B$ exists, and if such a c' exists, whether or not $c'' \not\leq c'$ for all $c'' \in \min(U) \cap \mathcal{C}_{\leq \|c'\|-1}$. Consequently, the second inner loop is dominant and has at most $n \cdot |R| \cdot |\Lambda|$ iterations. We store the vectors of M in the k-fold tree $T_b(k)$ described in [19]. The value k is the dimension of the vectors of M, and thus $k = |\Lambda|$. To determine if a vector v is such that $w \not\leq v$ for all vectors w in $T_b(k)$ takes $O(\log^{k-\frac{1}{2}}(N))$ vector comparisons, where $N = |M|$ is the number of elements in $T_b(k)$. Thus, we have $O(n \log^{|\Lambda|-\frac{1}{2}}(n) \cdot |R| \cdot |\Lambda|)$ vector comparisons. Inserting a vector in $T_b(k)$ takes $O(\log^{k-\frac{1}{2}}(N))$ vector comparisons and so this step does not dominate. Comparison of two vectors takes $O(|\Lambda| \cdot \log(z))$ time, assuming the entries of a vector are binary encoded. Consequently, we obtain the stated complexity. □

We remark that there is no obvious way to extend Algorithm 1 for arbitrary o-stable CRDs. Indeed, Lemma 1 depends on $\mathcal{D}$ being bimolecular. Moreover, it is not clear how to generate the elements of $\min(U)$ in order of their size (as used in the proof of Theorem 5) since minimal configurations may generate larger minimal configurations. In fact, it is not even clear if it is decidable, given an arbitrary o-stable CRD $\mathcal{D}$ and a finite set M of configurations, whether or not $M = \min(U)$.

In view of Theorem 6, it would be interesting to obtain an upper bound on $|\min(U)|$. In fact, it is perhaps reasonable to view $|\min(U)|$ as a measure for the "complexity" of the underlying o-stable CRD $\mathcal{D}$. The set $\min(U)$ is an antichain, as any two elements of $\min(U)$ are incomparable (i.e., if $x, y \in \min(U)$ are distinct, then $x \not\leq y$ and $y \not\leq x$). In general, antichains can be arbitrary large for fixed Λ: for example, for every $k \in \mathbb{N}$, $\mathcal{C}_{=k}$ is an antichain with $|\mathcal{C}_{=k}| = \left(\!\binom{|\Lambda|}{k}\!\right) > k$ if $|\Lambda| \geq 2$. Note however that, by Lemma 1, if $x \in \min(U)$ with $\|x\| = k$, then for every $l \in \{2, \ldots, k-1\}$ there is a $y \in \min(U)$ with $\|y\| = l$. Thus, in particular, $\min(U)$ (for some o-stable bimolecular CRD $\mathcal{D}$) cannot be equal to $\mathcal{C}_{=k}$ for any $k \geq 3$. We expect, but it would be interesting to confirm, that the existence of these "small" configurations in $\min(U)$ significantly restricts the cardinality of the antichain $\min(U)$.

5 Discussion

Using the semilinearity proof of [1], we found that the class of t-stable CRDs have equal expressive power as the larger class of o-stable CRDs. Also, we shown a subtle difference in expressive power between CRDs and CRDs with leaders. Then, we considered the problem of determining whether or not a given configuration c is output stable. In particular, we have shown that the set $\min(U)$ of minimal output unstable configurations may be efficiently computed provided that we restrict to the class of o-stable bimolecular CRDs. Given $\min(U)$ it is straightforward to verify whether or not a given configuration c is output stable.

Various questions regarding the computational complexity of CRDs are open. For example, is it decidable whether or not a given CRD is o-stable, or whether or not it is t-stable? Also, likely some "bridges" between the domains of CRDs (functioning as acceptors/deciders) and Petri nets (functioning as generators) remain to be discovered. For example, the semilinear sets are precisely the sets of reachable markings of weakly persistent Petri nets [20]. This suggests a possible link between the notions of weak persistence (from the domain of Petri nets) and stable deciders (from the domain of CRDs).

Acknowledgements. We thank Jan Van den Bussche for interesting discussions on CRNs and for useful comments on an earlier version of this paper.

We also thank the anonymous reviewers for useful comments on the paper. R.B. is a postdoctoral fellow of the Research Foundation – Flanders (FWO).

References

1. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing 18(4), 235–253 (2006)
2. Angluin, D., Aspnes, J., Eisenstat, D.: Stably computable predicates are semilinear. In: Ruppert, E., Malkhi, D. (eds.) Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC 2006), pp. 292–299. ACM (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 EATCS 93, 98–117 (2007)
5. Chen, H.-L., Doty, D., Soloveichik, D.: Deterministic function computation with chemical reaction networks. In: Stefanovic, D., Turberfield, A. (eds.) DNA 18. LNCS, vol. 7433, pp. 25–42. Springer, Heidelberg (2012)
6. Chen, H.-L., Doty, D., Soloveichik, D.: Rate-independent computation in continuous chemical reaction networks. In: Naor, M. (ed.) Innovations in Theoretical Computer Science (ITCS 2014), pp. 313–326. ACM (2014)
7. Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses. Natural Computing Series, pp. 543–584. Springer, Heidelberg (2009)
8. Dickson, L.E.: Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. American Journal of Mathematics 35, 413–422 (1913)
9. Doty, D.: Timing in chemical reaction networks. In: Chekuri, C. (ed.) Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014), pp. 772–784. SIAM (2014)
10. Doty, D., Hajiaghayi, M.: Leaderless deterministic chemical reaction networks. In: Soloveichik, D., Yurke, B. (eds.) DNA 2013. LNCS, vol. 8141, pp. 46–60. Springer, Heidelberg (2013)
11. Karp, R.M., Miller, R.E.: Parallel program schemata. Journal of Computer and System Sciences 3(2), 147–195 (1969)
12. Leroux, J.: Vector addition systems reachability problem (a simpler solution). In: Voronkov, A. (ed.) Proceedings of the Alan Turing Centenary Conference (Turing-100). EPiC Series, vol. 10, pp. 214–228 (2012)
13. Mayr, E.W.: An algorithm for the general Petri net reachability problem. In: Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC 1981), pp. 238–246. ACM, New York (1981)
14. Peterson, J.L.: Petri nets. ACM Computing Surveys 9(3), 223–252 (1977)
15. Plotkin, G.D.: A calculus of chemical systems. In: Tannen, V., Wong, L., Libkin, L., Fan, W., Tan, W.-C., Fourman, M. (eds.) Buneman Festschrift 2013. LNCS, vol. 8000, pp. 445–465. Springer, Heidelberg (2013)
16. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing 7(4), 615–633 (2008)

17. 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)
18. Stanley, R.P.: Enumerative Combinatorics, 2nd edn. Cambridge Studies in Advanced Mathematics, vol. 1. Cambridge University Press (2011)
19. Willard, D.E.: New data structures for orthogonal range queries. SIAM Journal on Computing 14(1), 232–253 (1985)
20. Yamasaki, H.: On weak persistency of Petri nets. Information Processing Letters 13(3), 94–97 (1981)

Parallel and Scalable Computation and Spatial Dynamics with DNA-Based Chemical Reaction Networks on a Surface

Lulu Qian[1,2] and Erik Winfree[1,2,3]

[1] Bioengineering,
[2] Computer Science,
[3] Computation and Neural Systems
California Institute of Technology, Pasadena, CA 91125, USA
{luluqian,winfree}@caltech.edu

Abstract. We propose a theoretical framework that uses a novel DNA strand displacement mechanism to implement abstract chemical reaction networks (CRNs) on the surface of a DNA nanostructure, and show that surface CRNs can perform efficient algorithmic computation and create complex spatial dynamics. We argue that programming molecular behaviors with surface CRNs is systematic, parallel and scalable.

1 Introduction

Despite the increasing complexity of synthetic DNA circuits and machinery [48], every step that is made towards building more sophisticated and powerful molecular systems has also revealed new challenges in the scalability and programmability of such systems. For example, in DNA strand displacement circuits, a larger circuit results in a larger number of distinct DNA molecules, and the spurious interactions among these DNA molecules can temporarily disable a fraction of the circuit components and thus slow down the computation and increase the error rate. A number of implementations have been proposed to explore the possibility of using spatial organization that allows circuit components to interact without requiring diffusion, and thus increase the speed of computation and limit spurious interactions to immediate neighbors [5,28]. Interestingly, localized circuits are related to molecular robotics [10]. Spatial organization could also enable running multiple instances of circuits in parallel, each on a different surface but all in the same test tube.

Another challenge is to implement fully general and efficient (space-limited) algorithmic computation that is experimentally feasible and scalable. By this we mean, informally, systems capable of storing and retrieving data from memory, performing logical operations, and executing iterative loops that repeat a processing step. For the purpose of this paper, we are not concerned with whether literally unbounded memory is available, so both a Turing machine and a laptop computer (i.e. a sequential digital logic circuit) would qualify. Efficient (space-limited) computations are possible on both, and a small program can perform a

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 114–131, 2014.

large computation. In contrast, feedforward digital logic circuits can only perform limited computation: they have no memory and during execution from input to output, each logic gate updates exactly once. Even if provided new inputs, a feedforward circuit can only make decisions based on the current input signals, while a Turing machine or sequential circuit can store information in memory and access it at a later point to make more sophisticated decisions based on both current and historical input signals.

In this work, we first explain an abstract model of chemical reaction networks (CRNs) on a surface, in the context of how bimolecular reactions alone can efficiently simulate (space-bounded) Turing machines. Then we introduce the implementation of formal unimolecular and bimolecular reactions on surface, based on a novel DNA mechanism (the three-way initiated four-way strand displacement reaction) that can recognize a DNA signal, pull it off from the surface, and simultaneously load a different signal onto the same location. Following the implementation, we give an example of propagating waves created from a simple set of surface CRNs. Finally, to demonstrate the power and elegance of surface CRNs, we develop systematic approaches for building continually active logic circuits and cellular automata. These approaches are highly efficient and scalable in two ways: First, they compute and generate complex spatial dynamics in parallel. Second, they use a constant number of distinct molecules for vastly different sizes of molecular programs.

2 Abstract Chemical Reaction Networks on a Surface

In the abstract model of surface CRNs, formal chemical species (e.g A, B, C, etc.) are located on a finite two-dimensional grid, and each site has a finite number of neighboring sites. Chemical species at any site can be recognized and converted into an arbitrary different species multiple times, either at a single site through a formal unimolecular reaction (e.g. $A \to B$), or cooperatively with a neighboring site through a formal bimolecular reaction (e.g. $A + B \to C + D$, if A and B are neighbors, then A gets converted to C while simultaneously B gets converted to D). Bimolecular reactions can be applied in any orientation on the surface, but always the first reactant is replaced by the first product, and the second by the second. Thus, $A + B \to C + D$ is effectively the same as $B + A \to D + C$, but is distinct from $A{+}B \to D{+}C$. Importantly, molecules do not move from site to site unless there is an explicit reaction, so by default there is no diffusion. (Diffusion could be "simulated" by including extra reactions such as $A + B \to B + A$ that allow species A and B to randomly walk through each other.) Unlike well-mixed CRNs, both unimolecular and bimolecular reaction rate constants have the same units, per second, because molecules have explicit locations and thus it is not necessary to approximate diffusion and collision probabilities.

Previous works have shown that synthetic DNA molecules can be used to implement arbitrary CRNs in a well-mixed solution [36,7]. As an extension and complement to well-mixed CRNs, we will show that DNA strands can be tethered on the surface of a DNA nanostructure such as DNA origami [32] to implement surface CRNs. Fig. 1a shows the abstract diagram of a small portion of a DNA

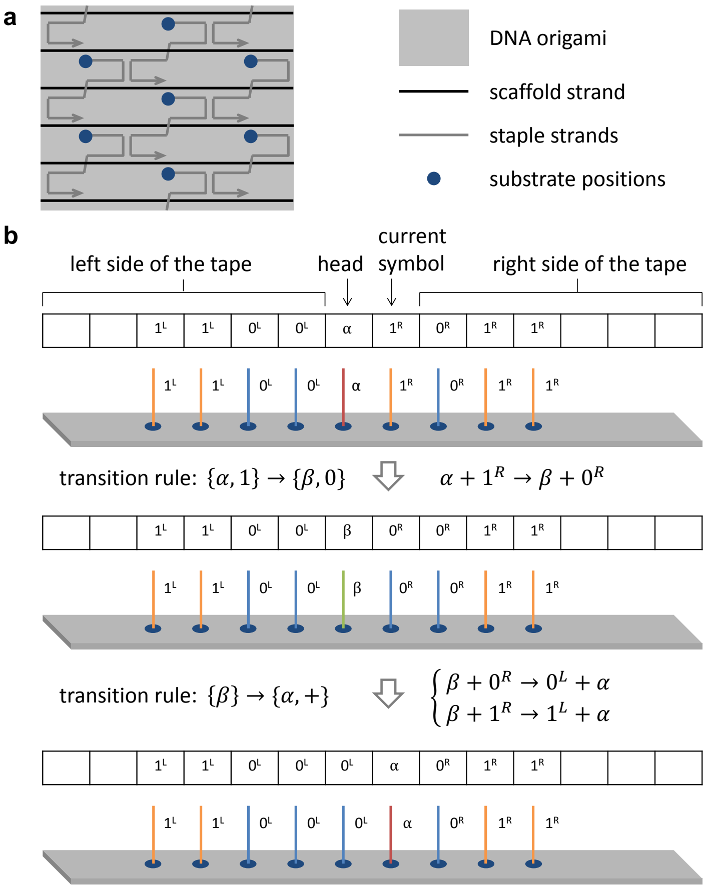


Fig. 1. Abstract chemical reaction networks (CRNs) on a surface. **(a)** Abstract diagram of a small portion of a DNA origami. All substrate locations are about 6 nm apart from each other. **(b)** An example of efficient molecular Turing machines implemented with abstract surface CRNs.

origami. Each staple strand can be extended from the 5' or 3' end to provide substrate positions for tethering DNA molecules that represent distinct chemical species. Because all staple strands have distinct sequences, all substrate locations on a origami surface are uniquely addressable. This allows us to locate specific DNA-based chemical species at their initial sites on a two-dimensional hexagonal grid of about 200 sites with about 6 nm distance between any neighboring sites, using a single DNA origami. With DNA origami arrays [26], much larger two-dimensional grids can be created.

Turing machines are one of the simplest models for universal computation, and building molecular Turing machines has a been a challenge for over 30 years. Charles Bennett was the first to come up with an implementation that uses hypothetical enzymes [2]. Later, concrete molecular implementations were proposed. The DNA tile self-assembly model was developed and proved to be Turing universal, but it has the distinct disadvantage of storing the entire history of its computation within an array of DNA tiles [33]. Both non-autonomous [34,1,35] and autonomous [44] DNA-based Turing machines were designed; however they required enzymes such as ligase and restriction enzymes for their operation, and their complexity discouraged experimental implementation. Well-mixed CRNs were also shown to be probabilistically Turing universal, but the computation requires molecular counts and therefore volumes that grow exponentially with the amount of memory used [37]. Recently, we developed a stack machine model with DNA polymers [30] that can simulate Turing machines efficiently, but there must be exactly one copy of the polymer DNA molecule for each stack, which introduces significant experimental challenges.

As an example to illustrate the power and generality of surface CRNs, Fig. 1b shows the implementation for a hypothetical molecular Turing machine with a finite tape. Unlike the stack machine implementation with polymer CRNs, our new Turing machine implementation with surface CRNs allows multiple independent Turing machines to operate in parallel within the same test tube, and there is no slow-down as the reaction volume gets larger. However, whereas the stack machine construction has an explicit mechanism for growing an unbounded amount of memory, the Turing machine construction here is limited to the size of the origami surface; in principle, this limit is obviated by unbounded self-assembly of origami [26], ideally configured so that self-assembly is inhibited until triggered [11] by the Turing machine needing more memory.

To review, a Turing machine has a head that moves along a tape and reads or writes symbols on the tape. The function of a Turing machine is decided by a set of transition rules. Each rule updates the state of the head and the symbol near the head based on the current state and symbol information, and moves the head to the left or right on the tape. Here we use an equivalent variant of the standard Turing machine where state change and movement steps are separate [3]. A tethered DNA strand on a DNA origami surface represents a state (such as α or β) if the position corresponds to the head, and represents a symbol (such as 0 or 1) if the position is on the tape. All symbols on the left side of the tape are 0^L or 1^L, and those on the right side are 0^R or 1^R. Each transition rule can be encoded in one or more formal bimolecular reactions on a surface. For example, transition rule $\{\alpha, 1\} \to \{\beta, 0\}$, which reads current state α and current symbol 1 and updates the state to β and the symbol to 0, can be encoded in surface reaction $\alpha + 1^R \to \beta + 0^R$. Transition rule $\{\beta\} \to \{\alpha, +\}$, which reads current state β, updates the state to α, and moves the head to one cell on the right, can be encoded as two surface reactions $\beta + 0^R \to 0^L + \alpha$ and $\beta + 1^R \to 1^L + \alpha$.

3 Implementation of Surface CRNs

It is a significant challenge to implement surface CRNs. Existing DNA implementations of well-mixed CRNs [36,7] use the mechanism of three-way strand displacement [45]. Such implementations can not be used for recognizing and updating a DNA signal on a surface for two reasons: First, in the process of

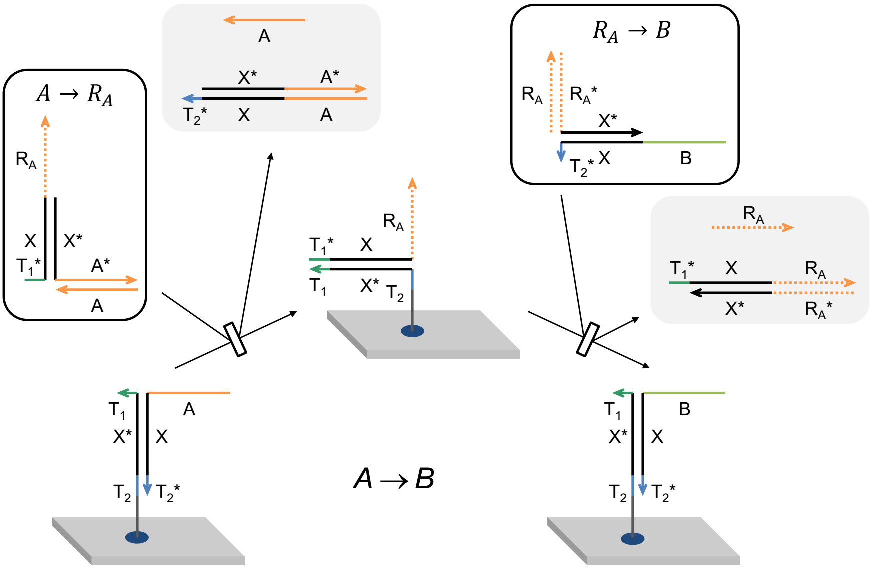


Fig. 2. DNA implementation of formal unimolecular reaction $A \rightarrow B$ on a surface. Fuel molecules are highlighted with outlines and waste molecules are shaded with light grey background; they are free-floating molecules in solutions that are maintained at a constant concentration.

three-way strand displacement, a single-stranded signal that is being recognized will become bound to a complementary strand that is eventually part of a waste molecule. Thus the waste molecule would be stuck on the surface attached to the current signal strand. Second, upon completion of three-way strand displacement, a new signal strand would be released into the solution, instead of staying on the surface as an updated signal.

In contrast, the mechanism of four-way branch migration [40,27] allows the recognition of a double-stranded signal with two adjacent single-stranded toeholds. Upon completion of branch migration, all strands will have swapped their base-paring partners. One strand in the original signal will now become part of a new signal as a result of the formation of a new pair of adjacent single-stranded toeholds, which makes it possible to recognize and update a signal on a surface. Unlike the high specificity of signals that can be encoded in branch migration domains with three-way strand displacement [47], four-way branch migration relies on distinct toehold sequences to represent different signals, and thus it is limited to implementations that require a small number of signal species.

Here, we show that with the help of associative/combinatorial toehold [6,15], three-way strand displacement and four-way branch migration can be integrated into one single step, simultaneously achieving the high specificity of signals and the localized updating of signals through swapping base-paring partners. We call this new mechanism three-way initiated four-way strand displacement, and we use it to implement surface CRNs.

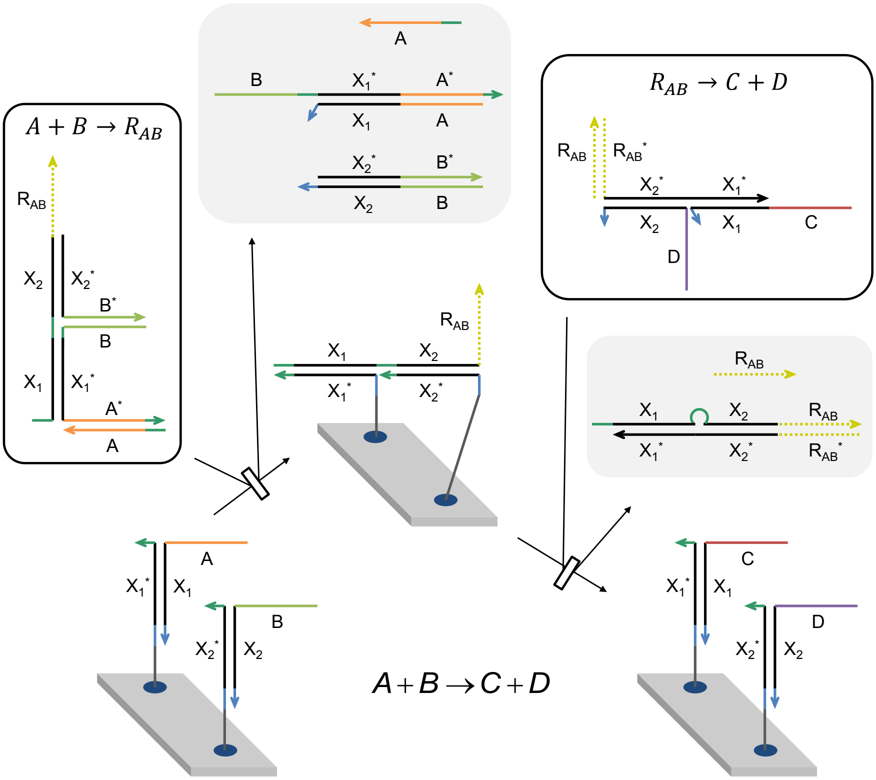


Fig. 3. DNA implementation of formal bimolecular reaction $A + B \to C + D$ on a surface. Fuel molecules are highlighted with outlines and waste molecules are shaded with light grey background, they are free-floating molecules in solution that are maintained at a constant concentration.

As shown in Fig. 2 and appendix Fig. A1, a DNA signal encoding chemical species A on a surface consists of a short single-stranded toehold domain (e.g. T_1, perhaps of 6 nucleotides) and a long single-stranded recognition domain (e.g. A, perhaps of 15 nucleotides) held together by a double-stranded branch migration domain (e.g. X and X^*) followed by a double-stranded toehold domain (e.g. T_2 and T_2^*). Initially, a fuel molecule $A \to R_A$ is free floating in solution. It first binds to signal A through the single-stranded domain T_1^*, three-way branch migration occurs within the double-stranded A and A^* domain, and a single-stranded waste molecule is produced. Simultaneous with the three-way branch migration, four-way branch migration occurs within two double-stranded X and X^* domains. When the double-stranded molecule on the right side is only attached by a short toehold T_2, it will spontaneously fall off the surface and become a waste molecule. At this point, signal A on a surface has been replaced by signal R_A, but with a different orientation. We then use a second fuel molecule $R_A \to B$ to replace R_A with B on the surface and to restore the original orientation of A. Note that the single strand A in fuel molecule $A \to R_A$ and single strand R_A

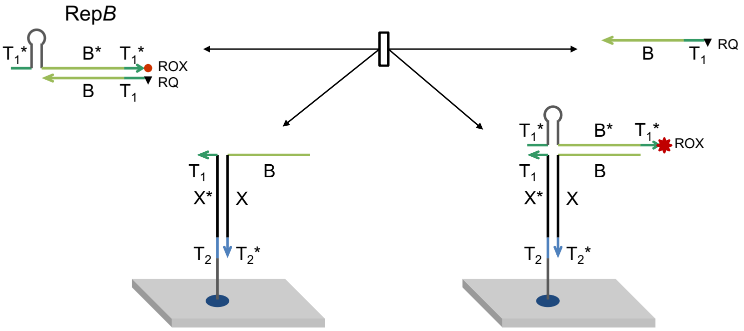


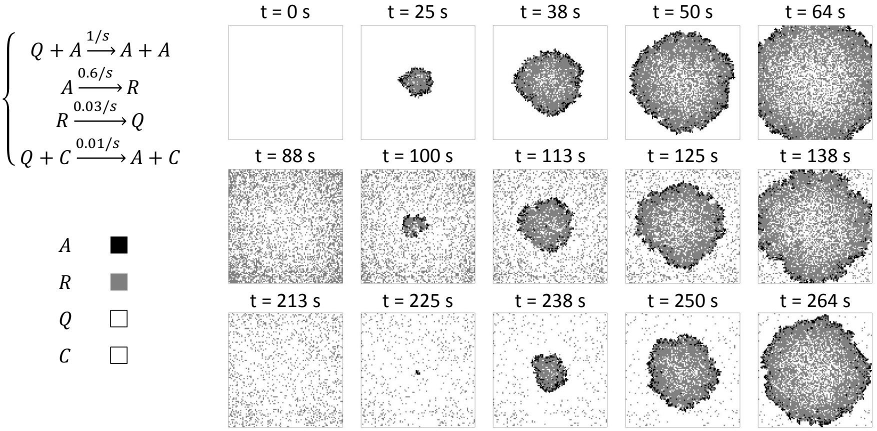
Fig. 4. DNA implementation of a reversible reporting reaction on a surface.

in fuel molecule $R_A \to B$ not only increase the specificity of signal recognition through branch migration, but also protect the fuel molecules from binding to each other in a larger network (e.g. when $A \to B$ and $X \to A$ co-exist).

Doubling the complexity of each fuel molecule, we can now implement the formal bimolecular reaction $A + B \to C + D$ on a surface. As shown in Fig. 3 and appendix Fig. A2, signals A and B are located at neighboring sites on the surface. Fuel molecule $A + B \to R_{AB}$ first undergoes three-way initiated four-way branch migration with signal A on the surface; at the end of this process two short toeholds spontaneously disassociate and neighboring signal B can bind to the intermediate product and undergo a second three-way initiated four-way branch migration reaction to replace both signals A and B with a joint signal R_{AB} on the surface. A second fuel molecule $R_{AB} \to C+D$ then recognizes R_{AB}, and signal D will be placed at the original site of B followed by C being placed at the original site of A.

The keen observer will note that this mechanism requires neighboring sites to make use of distinct branch migration domains X_1 and X_2, rather than universally X. This constraint can be accommodated by using a checkerboard arrangement of X_1 and X_2 sites on the origami and by multiplying the number of fuel species – for example, each unimolecular reaction will need both fuels using X_1 and fuels using X_2 (unless, for some reason, we wish to restrict the unimolecular reaction to just one color of the checkerboard). As it is straightforward to handle, we will henceforth ignore this minor complication.

To experimentally read DNA signals on a surface, we propose a reporting mechanism that reversibly produces a fluorescent signal. As shown in Fig. 4, the free-floating reporter molecule is labeled with a fluorophore and a quencher. A reversible three-way strand displacement reaction separates the fluorophore from the quencher and results in increased fluorescence that can be measured in bulk by a spectrofluorometer. This mechanism, reversibly binding and activating the fluorophore, is compatible with the DNA-PAINT method [22] for super-resolution microscopy, suggesting that dynamic spatial patterns could be observed on a single origami or origami array.



$$\begin{cases} Q + A \xrightarrow{1/s} A + A \\ A \xrightarrow{0.6/s} R \\ R \xrightarrow{0.03/s} Q \\ Q + C \xrightarrow{0.01/s} A + C \end{cases}$$

Fig. 5. A nanoscaled pacemaker that triggers a propagating wave. Simulated on a 100 by 100 grid, black pixels indicate signal A, grey pixels indicate signal R, and white pixels indicate signal Q. The signal in the center of the grid is always C.

4 Dynamic Spatial Patterns

With just unimolecular and bimolecular surface CRNs, dynamic spatial patterns can be created on a two-dimensional DNA origami surface. For example, with the set of four reactions shown in Fig. 5, a propagating wave pulse can be repeatedly generated. Initially, the site in the center of the grid has signal C ("the pacemaker") and all other sites have signal Q. The only reaction that can take place under this initial condition is $Q + C \to A + C$, allowing the signal in the center to update one of its neighbors from Q to A. Subsequently, a fast reaction $Q + A \to A + A$ will occur, and each site with signal A will update its neighbors from Q to A, creating the front of a wave. A slower reaction $A \to R$ will then convert signals A to R, thereby identifying older parts of the wave and helping establish directionality of the wave propagation. A even slower reaction $R \to Q$ will restore signals from R to Q after the wavefront has passed. Finally, the slowest reaction $Q + C \to A + C$ enables a new wave to emerge from the center after the previous wave has faded.

In this example, if all possible reactions execute synchronously (independent of rate constants), the propagating wave will expand deterministically — it is the 3-state Greenberg-Hastings model of excitable media [17]. But discrete CRNs are intrinsically asynchronous, all signals will be updated stochastically, and the edge of the wave will be less well defined, as shown in the simulation in Fig. 5. To obtain reliable wave propagation, rate parameters must be tuned roughly as described above. In the DNA implementation, desired rates can be achieved by varying the lengths of toeholds on fuel molecules encoding the unimolecular and bimolecular reactions (assuming the three-way initiated four-way strand displacement follow roughly the same range of kinetics as three-way strand displacement [46]). As an alternative to tuning rates, it is possible to design (typically larger) surface CRNs that behave exactly as if they were updated synchronously;

we will discuss such an approach later in the paper, in the context of cellular automata.

Rather than eliminate it, the randomness of asynchronous reaction execution can be embraced, and in combination with explicit reactions that simulate two-dimensional diffusion (e.g. $X + e \to e + X$ for all diffusible species X and a special "empty space" signal e), we obtain the entire space of chemical reaction-diffusion systems in the stochastic limit, closely analogous to reactive lattice gas automata models [4]. For example, spiral wave dynamical patterns could be achieved using the six-reaction Oregonator model of the Belousov-Zhabotinsky excitable medium [21].

5 Continuously Active Logic Circuits

Unlike in reaction-diffusion systems, signals in surface CRNs by default will remain in their exact location until a reaction occurs. This feature enables precise geometric control at the single-molecule level, and can be exploited to carry out precise tasks such as digital circuit computation. As shown in Fig. 6b, to construct a two-input OR gate with surface CRNs, three neighboring sites are initially assigned with blank signals $B^{\cup x}$, $B^{\cup y}$ and $B^{\cup z}$, and each of them has another blank neighboring site B to serve as a wire that moves signals around and connects layers of logic gates together (Fig. 6a). Logic OR computation can be performed with six bimolecular reactions. The first four reactions recognize the two input signals 00, 01, 10, or 11 at the x and y sites, update the y site to the correct output signal $0^{\cup k}$ or $1^{\cup k}$, and reset the x site to be blank. The last two reactions move the output signal to the z site and reset the y site to be blank. Similarly, AND gate and NOT gate can be implemented with six and two reactions respectively (Fig. 6cd). Additional straightforward reactions are needed to load the signal from the input wires onto the gate, and to push the output onto its wire (Fig. 6a). These unidirectional reactions ratchet the signals forward, despite the random walks on the wires.

With two additional sets of reactions implementing signal fan-out and crossing wires (Fig. 6ef), arbitrary feedforward logic circuits can be systematically translated into surface CRNs. An example circuit that calculates the square roots of four-bit binary numbers is shown in Fig. 6g. To run the circuit, 0 or 1 signals are initiated at $x_1, \ldots, x_4$, while y_1 and y_2 are in state B. Signals asynchronously propagate through the circuit. Two-input gates must wait until both input signals arrive, before they can produce an output. Crossing wires are designed to ensure that deadlock is impossible. The correct circuit outputs are eventually produced at y_1 and y_2 regardless of the order in which reactions execute.

Unlike the previous well-mixed DNA logic circuits, which deplete some circuit components by the end of each computation and thus are not capable of responding to a new set of input signals [31], these logic circuits with surface CRNs are continuously active. With free-floating fuel molecules in large excess, the signal on each site can be updated multiple times, switching between "0" and "1" and back, as needed. With reversible reporters that read the output signals without consuming them, a changed set of input signals will trigger a cascade of reactions resulting in the update of output signals and associated fluorescence.

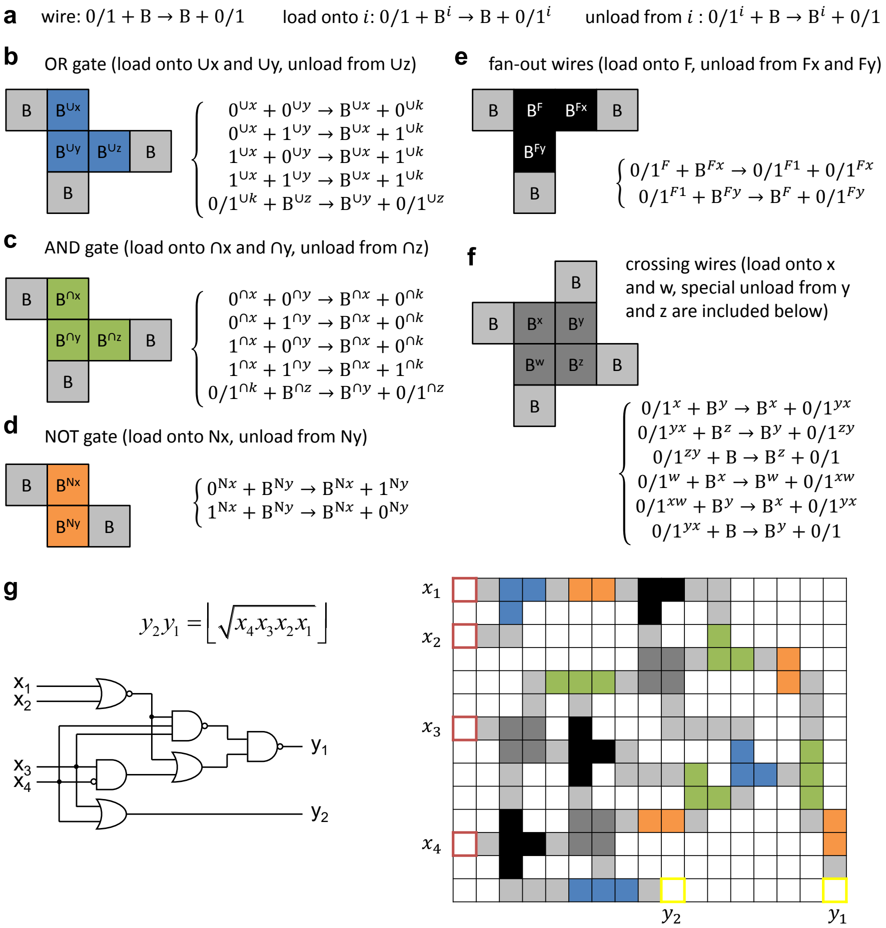

a
wire: $0/1 + B \rightarrow B + 0/1$
load onto i: $0/1 + B^i \rightarrow B + 0/1^i$
unload from i : $0/1^i + B \rightarrow B^i + 0/1$
b
OR gate (load onto ∪x and ∪y, unload from ∪z)
B
$B^{\cup x}$
$B^{\cup y}$
$B^{\cup z}$
$0^{\cup x} + 0^{\cup y} \rightarrow B^{\cup x} + 0^{\cup k}$
$0^{\cup x} + 1^{\cup y} \rightarrow B^{\cup x} + 1^{\cup k}$
$1^{\cup x} + 0^{\cup y} \rightarrow B^{\cup x} + 1^{\cup k}$
$1^{\cup x} + 1^{\cup y} \rightarrow B^{\cup x} + 1^{\cup k}$
$0/1^{\cup k} + B^{\cup z} \rightarrow B^{\cup y} + 0/1^{\cup z}$
c
AND gate (load onto ∩x and ∩y, unload from ∩z)
$B^{\cap x}$
$B^{\cap y}$
$B^{\cap z}$
$0^{\cap x} + 0^{\cap y} \rightarrow B^{\cap x} + 0^{\cap k}$
$0^{\cap x} + 1^{\cap y} \rightarrow B^{\cap x} + 0^{\cap k}$
$1^{\cap x} + 0^{\cap y} \rightarrow B^{\cap x} + 0^{\cap k}$
$1^{\cap x} + 1^{\cap y} \rightarrow B^{\cap x} + 1^{\cap k}$
$0/1^{\cap k} + B^{\cap z} \rightarrow B^{\cap y} + 0/1^{\cap z}$
d
NOT gate (load onto Nx, unload from Ny)
B^{Nx}
B^{Ny}
$0^{Nx} + B^{Ny} \rightarrow B^{Nx} + 1^{Ny}$
$1^{Nx} + B^{Ny} \rightarrow B^{Nx} + 0^{Ny}$
e
fan-out wires (load onto F, unload from Fx and Fy)
B^F
B^{Fx}
B^{Fy}
$0/1^F + B^{Fx} \rightarrow 0/1^{F1} + 0/1^{Fx}$
$0/1^{F1} + B^{Fy} \rightarrow B^F + 0/1^{Fy}$
f
crossing wires (load onto x and w, special unload from y and z are included below)
B^x
B^y
B^w
B^z
$0/1^x + B^y \rightarrow B^x + 0/1^{yx}$
$0/1^{yx} + B^z \rightarrow B^y + 0/1^{zy}$
$0/1^{zy} + B \rightarrow B^z + 0/1$
$0/1^w + B^x \rightarrow B^w + 0/1^{xw}$
$0/1^{xw} + B^y \rightarrow B^x + 0/1^{yx}$
$0/1^{yx} + B \rightarrow B^y + 0/1$
g
$y_2y_1 = \lfloor \sqrt{x_4x_3x_2x_1} \rfloor$
x_1
x_2
x_3
x_4
y_1
y_2

Fig. 6. Continuously active logic circuits. **(a)** wire, loading and unloading, **(b)** OR gate, **(c)** AND gate, **(d)** NOT gate, **(e)** fan-out wires and **(f)** crossing wires implemented with surface CRNs. "0/1" is shorthand for two rules of the same form, one with all instances "0" and the other with all instances "1". **(g)** A four-bit square root circuit implemented with surface CRNs. The three-input AND gate is implemented with 2 two-input AND gates. The fan-out of three is implemented similarly as (but distinctly from) the fan-out of two, with an extra state $F2$ of site B^F.

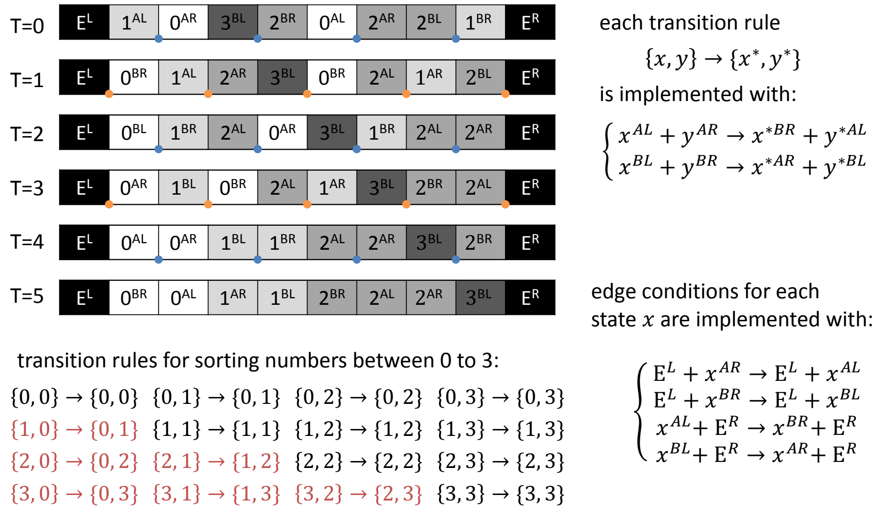
An additional benefit of the continuously active logic circuit architecture using surface CRNs is that iterative sequential circuits can be implemented using the same mechanisms. For example, if three NOT gates are wired together in a ring (a canonical oscillatory circuit), and a single 0 signal is placed on one of the wires, then the signal will travel around and around the ring, flipping from 0 to 1 and back as it goes. More usefully, to iterate a function $f(x_1, x_2, \ldots, x_n) = (x'_1, x'_2, \ldots, x'_n)$, the outputs of a circuit computing $f()$ merely need to be routed back to the input, controlled with a synchronizing signal that ensures that the new inputs are not activated until all of the previous outputs have been computed and collected. Thus, in principle, arbitrary finite state machines, and even standard central processing unit (CPU) designs, can be efficiently implemented using surface CRNs on a large enough origami array. Such designs are closely related to the implementation of delay-insensitive circuits in asynchronous cellular automata [25].

Logic circuits with surface CRNs should be more scalable than previous DNA-based logic circuits. Any feedforward or sequential logic circuit can be implemented with the same set of signal molecules on the surface and fuel molecules in solution, regardless of the circuit size. A different circuit will correspond to a different layout of signals on the surface, and a larger circuit simply requires a larger grid on DNA origami. For example, all three OR gates in the square root circuit (Fig. 6g) at different locations on the surface will interact with the same set of fuel molecules (Fig. 6b) to perform the desired computation, and all three OR gates can operate at the same time. Assuming the concentrations of all fuel molecules stay constant, which can be approximated by using a small amount of DNA origami (and hence signal molecules on its surface) and a large excess of fuel molecules in solution, the speed of each logic operation should stay the same in larger circuits. This is in contrast to well-mixed DNA circuits, where the maximum total DNA concentration limit requires that larger circuits operate at lower signal concentrations, resulting in a per-operation slowdown linear in the number of gates (c.f. SI section S15 of [31]). Finally, because each origami can contain a different circuit and/or be initialized with different input, billions of independent circuit computations can execute within a single test tube, in contrast to well-mixed bulk-phase reactions where only a single circuit is executed.

6 Cellular Automata

Compared to Turing machines where only the single site representing the head is updated at a time, cellular automata take full advantage of parallel computation and can efficiently generate complex patterns that evolve over time. A cellular automaton has a grid of cells, each with an initial state. Based on the current state of itself and its neighbors, each cell can be updated to a new state. A set of transition rules determine how the cells are updated, and these rules are applied to all cells in the grid simultaneously. Interesting biological processes such as the behavior of muscle cells, cardiac function, animal coat markings, and plant ecology have been simulated by cellular automata [13]. Even some of the simplest one-dimensional cellular automata with two states (0 and 1) are rich enough to support universal computation [8]. DNA tile self-assembly has been used to successfully implement cellular automata [33], but only by constructing a static pattern representing the cellular automata's space-time history. A previous proposal to implement one-dimensional cellular automata dynamics on one-dimensional structures [43] used a variety of enzymes and may not be experimentally feasible.

An example of cellular automata with surface CRNs is shown in Fig. 7. It is an implementation of a one-dimensional block cellular automaton that sorts



T=0	E^L	1^{AL}	0^{AR}	3^{BL}	2^{BR}	0^{AL}	2^{AR}	2^{BL}	1^{BR}	E^R
T=1	E^L	0^{BR}	1^{AL}	2^{AR}	3^{BL}	0^{BR}	2^{AL}	1^{AR}	2^{BL}	E^R
T=2	E^L	0^{BL}	1^{BR}	2^{AL}	0^{AR}	3^{BL}	1^{BR}	2^{AL}	2^{AR}	E^R
T=3	E^L	0^{AR}	1^{BL}	0^{BR}	2^{AL}	1^{AR}	3^{BL}	2^{BR}	2^{AL}	E^R
T=4	E^L	0^{AL}	0^{AR}	1^{BL}	1^{BR}	2^{AL}	2^{AR}	3^{BL}	2^{BR}	E^R
T=5	E^L	0^{BR}	0^{AL}	1^{AR}	1^{BL}	2^{BR}	2^{AL}	2^{AR}	3^{BL}	E^R

each transition rule

$$\{x,y\} \to \{x^*,y^*\}$$

is implemented with:

$$\begin{cases} x^{AL} + y^{AR} \to x^{*BR} + y^{*AL} \\ x^{BL} + y^{BR} \to x^{*AR} + y^{*BL} \end{cases}$$

edge conditions for each state x are implemented with:

$$\begin{cases} \mathrm{E}^L + x^{AR} \to \mathrm{E}^L + x^{AL} \\ \mathrm{E}^L + x^{BR} \to \mathrm{E}^L + x^{BL} \\ x^{AL} + \mathrm{E}^R \to x^{BR} + \mathrm{E}^R \\ x^{BL} + \mathrm{E}^R \to x^{AR} + \mathrm{E}^R \end{cases}$$

transition rules for sorting numbers between 0 to 3:

$\{0,0\} \to \{0,0\}$ $\{0,1\} \to \{0,1\}$ $\{0,2\} \to \{0,2\}$ $\{0,3\} \to \{0,3\}$
$\{1,0\} \to \{0,1\}$ $\{1,1\} \to \{1,1\}$ $\{1,2\} \to \{1,2\}$ $\{1,3\} \to \{1,3\}$
$\{2,0\} \to \{0,2\}$ $\{2,1\} \to \{1,2\}$ $\{2,2\} \to \{2,2\}$ $\{2,3\} \to \{2,3\}$
$\{3,0\} \to \{0,3\}$ $\{3,1\} \to \{1,3\}$ $\{3,2\} \to \{2,3\}$ $\{3,3\} \to \{3,3\}$

Fig. 7. A one-dimensional block cellular automaton that sorts numbers. The behavior with synchronous updates is shown, with blue and orange dots indicating pairing on alternate time steps. Superscripts indicate extra information that the asynchronous surface CRN must track to ensure correct behavior. Transition rules in red performs sorting and transition rules in black maintain the updates in alternating order.

single-digit numbers. A blocked (aka partitioning) cellular automata executes by dividing the array into pairs of neighboring sites, synchronously applying a look-up-table of rules for how to replace each pair by a new pair of values, shifts the pairing by one, and repeats. Unlike the cellular automata, our model of surface CRNs is intrinsically non-oriented and asynchronous. To allow the recognition of orientations and to synchronize the update of all sites, each signal molecule on a DNA origami surface (e.g. 1^{AL} and 0^{AR}) is designed to include both information about a number (e.g. 0, 1, 2, or 3) and information about block pairing (AL, AR, BL or BR, where AL indicates the left side of an "A" pair, AR indicates the right side of an "A" pair, and so on). Each transition rule $\{x, y\} \to \{x^\star, y^\star\}$ that reads the current states of two neighboring cells and updates them with new states can be implemented with two fuel molecules encoding $x^{AL} + y^{AR} \to x^{\star BR} + y^{\star AL}$ and $x^{BL} + y^{BR} \to x^{\star AR} + y^{\star BL}$. This ensures that the ways of pairing up signals alternate after each update, so all signals will be compared with neighbors on both sides; further, it ensures that a site won't be updated again until its neighbor has caught up. Transition rules such as $\{1, 0\} \to \{0, 1\}$, $\{2, 1\} \to \{1, 2\}$, and $\{3, 2\} \to \{2, 3\}$ ensure that smaller numbers will be sorted to the left and larger numbers will be sorted to the right. Left edge signal E^L, right edge signal E^R and corresponding fuel molecules are used to complete the function of sorting.

The approach illustrated here for simulating synchronous one-dimensional block cellular automata with asynchronous one-dimensional surface CRNs can be generalized to provide simulations of arbitrary synchronous two-dimensional block cellular automata [39] and even traditional cellular automata with von Neumann and Moore neighborhood update functions [23]. The construction uses the same basic principles but is more elaborate, and is not presented here.

7 Discussion

The key mechanism that we proposed for implementing surface CRNs is a novel strand displacement primitive that we call the three-way initiated four-way strand displacement reaction. This reaction is more complex than any other DNA strand displacement primitives that have been demonstrated so far, and it is important to understand the kinetics and robustness of this reaction. We argue that because the toehold binding step makes use of both a regular toehold plus a co-axial stacking bond, it is immediately followed by three-way branch migration (without the delay one might expect from a remote toehold [16]). Presumably right after the initiation of three-way branch migration, the four-way branch migration will simultaneously take place. Thus the reaction should be completed roughly as fast as a single-step three-way or four-way strand displacement reaction.

It is also important to evaluate the potential leak reactions in our surface CRNs implementation. Any leak reactions that could occur between signal molecules on a surface would have to involve two double-stranded domains without toeholds, which is unlikely. Any leak reactions that could occur between fuel molecules in solution will only produce waste molecules in solution and not affect the state of signal molecules on surfaces. The most likely leak would be between a fuel molecule and a mismatching surface-bound signal: although single short toeholds are not very effective for initiating 4-way branch migration [9], it is possible that at some rate 4-way branch migration could initiate even without the 3-way branch migration taking place. Overall, however, the potential leak reactions should be less significant than other types of strand displacement systems. Also note that, because the number of distinct fuel molecules is constant with any size of the logic circuits with surface CRNs, the number of leak reactions will not scale with the complexity of the circuits. For cellular automata, a small set of transition rules can be used to create very complex behaviors programmed by various initial conditions, and in these cases the leak reactions will not scale with the size of programs either.

The implementation of formal bimolecular reactions on a surface raises a couple of concerns, including the step that requires two toeholds to temporarily disassociate when other part of the molecules are still held together, and the step that requires initiation of four-way branch migration with a toehold on one side and a bulge on another. We believe it should be possible to simplify and optimize this implementation.

It is inevitable that experimental implementation of surface CRNs will have occasional errors, and therefore robustness and fault-tolerance will be important issues to address in future work. Successful approaches may depend upon the type of surface CRN being implemented. For example, for digital circuits on DNA origami, classical and modern techniques for faulty digital circuits [41,19] provide solutions if the faults are transient bit-flips or permanent device failure. But DNA surface CRNs may have other types of faults, such as arbitrary signal errors (not just 0-to-1 or 1-to-0) and even signal loss, origami-to-origami exchange, or stuck intermediate reaction steps. Asynchronous cellular automata models are closer models; although error correction is more difficult there, techniques have been developed both in one and two dimensions [14,42]. Finding practical and effective methods for molecular programming remains an important challenge.

Even if errors at the molecular level (leak reactions, stuck intermediates, defective molecules, etc.) are negligible, the complexity of understanding DNA strand displacement system implementations at the domain level already calls for simulation tools and logical verification techniques. An important step will be expanding the capability of software like Visual DSD [29] to handle DNA complexes with arbitrary secondary structure as well as localization on surfaces – this would also enable simulation of a large variety of molecular robotics implementations. Even when the full set of domain-level reactions have been worked out for DNA strand displacement systems, logical errors may be difficult to identify manually. Automated approaches for verifying the correctness of well-mixed DNA strand displacement systems are being developed [24] and these may provide a starting point for verifying surface-based systems.

If these considerable challenges can be overcome, the resulting control over spatially-organized molecular systems could provide important new capabilities. As a systematic implementation for an extremely general class of systems, our approach leverages the effort spent characterizing and debugging a small set of reaction mechanisms to construct a wide range of system behaviors, ranging from continuously active algorithmic computation with memory, to pattern formation and spatial dynamics. Applications could include synthetic molecular systems that regulate biochemical pathways in response to continuously changing environmental signals, therapeutic molecular devices [12] that efficiently produce treatment decisions not only based on biomarkers that are currently present, but also those that indicate historical conditions of the target cell, or molecular-scale devices and instruments that precisely regulate nanoscale components (such as plasmonic elements [38] or chemical moieties [18,20]) to achieve measurement or synthesis tasks.

Acknowledgments. We thank Ho-Lin Chen and David Yu Zhang for insightful discussions and suggestions. LQ and EW were supported by the Molecular Programming Project under NSF expedition in computing (0832824 and 1317694). LQ was supported by a Career Award at the Scientific Interface from the Burroughs Wellcome Fund (1010684), a Faculty Early Career Development Award from NSF (1351081), and a Okawa Foundation Research Grant (0656180).

References

1. Beaver, D.: A universal molecular computer. DNA Based Computers, DIMACS 27, 29–36 (1996)

2. Bennett, C.H.: The thermodynamics of computation – a review. International Journal of Theoretical Physics 21, 905–940 (1982)
3. Bennett, C.H.: Logical reversibility of computation. IBM Journal of Research and Development 17, 525–532 (1973)
4. Boon, J.P., Dab, D., Kapral, R., Lawniczak, A.: Lattice gas automata for reactive systems. Physics Reports 273, 55–147 (1996)
5. Chandran, H., Gopalkrishnan, N., Phillips, A., Reif, J.: Localized hybridization circuits. In: Cardelli, L., Shih, W. (eds.) DNA 17. LNCS, vol. 6937, pp. 64–83. Springer, Heidelberg (2011)
6. Chen, X.: Expanding the rule set of DNA circuitry with associative toehold activation. Journal of the American Chemical Society 134, 263–271 (2011)
7. Chen, Y.J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from DNA. Nature Nanotechnology 8, 755–762 (2013)
8. Cook, M.: Universality in elementary cellular automata. Complex Systems 15, 1–40 (2004)
9. Dabby, N.L.: Synthetic molecular machines for active self-assembly: prototype algorithms, designs, and experimental study. Ph.D. thesis, California Institute of Technology (2013)
10. Dannenberg, F., Kwiatkowska, M., Thachuk, C., Turberfield, A.J.: DNA walker circuits: Computational potential, design, and verification. In: Soloveichik, D., Yurke, B. (eds.) DNA 2013. LNCS, vol. 8141, pp. 31–45. Springer, Heidelberg (2013)
11. Dirks, R.M., Pierce, N.A.: Triggered amplification by hybridization chain reaction. Proceedings of the National Academy of Sciences 101, 15275–15278 (2004)
12. Douglas, S.M., Bachelet, I., Church, G.M.: A logic-gated nanorobot for targeted transport of molecular payloads. Science 335, 831–834 (2012)
13. Ermentrout, G.B., Edelstein-Keshet, L.: Cellular automata approaches to biological modeling. Journal of Theoretical Biology 160, 97–133 (1993)
14. Gács, P.: Reliable cellular automata with self-organization. Journal of Statistical Physics 103, 45–267 (2001)
15. Genot, A.J., Bath, J., Turberfield, A.J.: Combinatorial displacement of DNA strands: application to matrix multiplication and weighted sums. Angewandte Chemie International Edition 52, 1189–1192 (2013)
16. Genot, A.J., Zhang, D.Y., Bath, J., Turberfield, A.J.: Remote toehold: a mechanism for flexible control of DNA hybridization kinetics. Journal of the American Chemical Society 133, 2177–2182 (2011)
17. Greenberg, J.M., Hastings, S.: Spatial patterns for discrete models of diffusion in excitable media. SIAM Journal on Applied Mathematics 34(3), 515–523 (1978)
18. Gu, H., Chao, J., Xiao, S.J., Seeman, N.C.: A proximity-based programmable DNA nanoscale assembly line. Nature 465, 202–205 (2010)
19. Han, J., Jonker, P.: A defect-and fault-tolerant architecture for nanocomputers. Nanotechnology 14, 224 (2003)
20. He, Y., Liu, D.R.: Autonomous multistep organic synthesis in a single isothermal solution mediated by a DNA walker. Nature Nanotechnology 5, 778–782 (2010)
21. Jahnke, W., Winfree, A.T.: A survey of spiral-wave behaviors in the Oregonator model. International Journal of Bifurcation and Chaos 1, 445–466 (1991)
22. Jungmann, R., Steinhauer, C., Scheible, M., Kuzyk, A., Tinnefeld, P., Simmel, F.C.: Single-molecule kinetics and super-resolution microscopy by fluorescence imaging of transient binding on DNA origami. Nano Letters 10, 4756–4761 (2010)

23. Kari, J.: Theory of cellular automata: A survey. Theoretical Computer Science 334, 3–33 (2005)
24. Lakin, M.R., Phillips, A., Stefanovic, D.: Modular verification of DNA strand displacement networks via serializability analysis. In: Soloveichik, D., Yurke, B. (eds.) DNA 19. LNCS, vol. 8141, pp. 133–146. Springer, Heidelberg (2013)
25. Lee, J., Adachi, S., Peper, F., Mashiko, S.: Delay-insensitive computation in asynchronous cellular automata. Journal of Computer and System Sciences 70, 201–220 (2005)
26. Liu, W., Zhong, H., Wang, R., Seeman, N.C.: Crystalline two-dimensional DNA-origami arrays. Angewandte Chemie 123, 278–281 (2011)
27. Muscat, R.A., Bath, J., Turberfield, A.J.: A programmable molecular robot. Nano Letters 11, 982–987 (2011)
28. Muscat, R.A., Strauss, K., Ceze, L., Seelig, G.: DNA-based molecular architecture with spatially localized components. In: Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA), pp. 177–188 (2013)
29. Phillips, A., Cardelli, L.: A programming language for composable DNA circuits. Journal of The Royal Society Interface 6, S419–S436 (2009)
30. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
31. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science 332, 1196–1201 (2011)
32. Rothemund, P.W.K.: Folding DNA to create nanoscale shapes and patterns. Nature 440, 297–302 (2006)
33. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology 2, e424 (2004)
34. Rothemund, P.W.K.: A DNA and restriction enzyme implementation of Turing machines. DNA Based Computers, DIMACS 27, 75–119 (1996)
35. Smith, W.D.: DNA computers in vitro and in vivo. DNA Based Computers, DIMACS 27, 121–185 (1996)
36. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proceedings of the National Academy of Sciences 107, 5393–5398 (2010)
37. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing 7(4), 615–633 (2008)
38. Tan, S.J., Campolongo, M.J., Luo, D., Cheng, W.: Building plasmonic nanostructures with DNA. Nature Nanotechnology 6, 268–276 (2011)
39. Toffoli, T., Margolus, N.: Cellular automata machines: a new environment for modeling. MIT Press (1987)
40. Venkataraman, S., Dirks, R.M., Rothemund, P.W.K., Winfree, E., Pierce, N.A.: An autonomous polymerization motor powered by DNA hybridization. Nature Nanotechnology 2, 490–494 (2007)
41. Von Neumann, J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. Automata Studies 34, 43–98 (1956)
42. Wang, W.: An asynchronous two-dimensional self-correcting cellular automaton. In: Proceedings of 32nd Annual Symposium on Foundations of Computer Science, pp. 278–285. IEEE (1991)
43. Yin, P., Sahu, S., Turberfield, A.J., Reif, J.H.: Design of autonomous DNA cellular automata. In: Carbone, A., Pierce, N.A. (eds.) DNA 11. LNCS, vol. 3892, pp. 399–416. Springer, Heidelberg (2006)

44. Yin, P., Turberfield, A.J., Sahu, S., Reif, J.H.: Design of an autonomous DNA nanomechanical device capable of universal computation and universal translational motion. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 10. LNCS, vol. 3384, pp. 426–444. Springer, Heidelberg (2005)
45. Yurke, B., Turberfield, A.J., Mills, A.P., Simmel, F.C., Neumann, J.L.: A DNA-fuelled molecular machine made of DNA. Nature 406, 605–608 (2000)
46. Zhang, D.Y., Winfree, E.: Control of DNA strand displacement kinetics using toehold exchange. Journal of the American Chemical Society 131, 17303–17314 (2009)
47. Zhang, D.Y., Chen, S.X., Yin, P.: Optimizing the specificity of nucleic acid hybridization. Nature Chemistry 4, 208–214 (2012)
48. Zhang, D.Y., Seelig, G.: Dynamic DNA nanotechnology using strand-displacement reactions. Nature Chemistry 3, 103–113 (2011)

Appendix

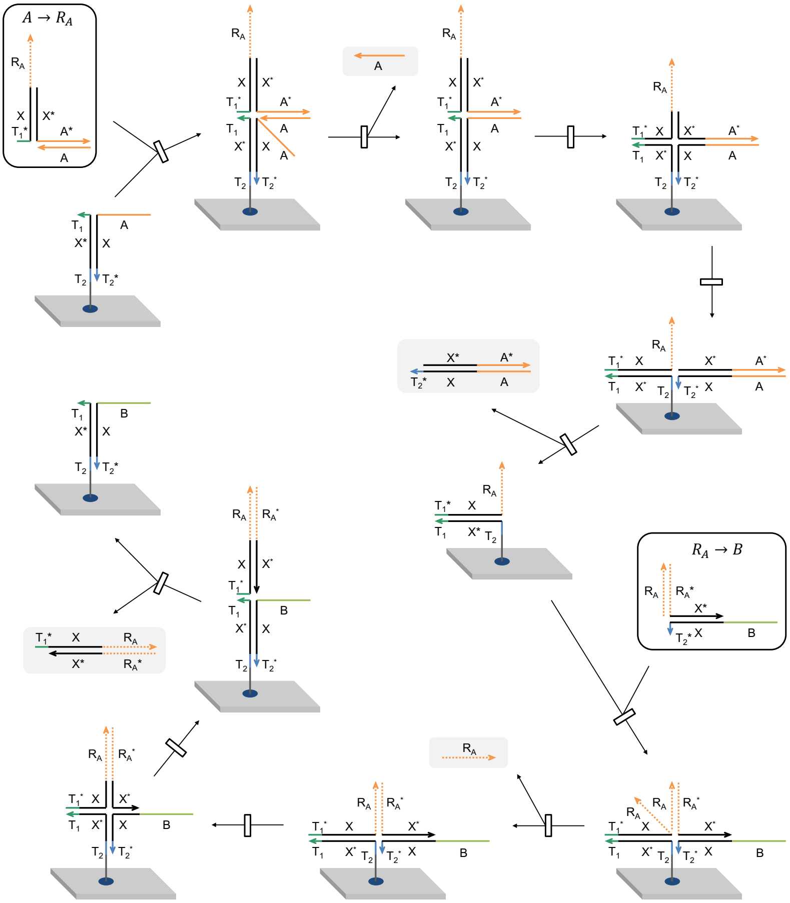


Fig. 8. Detailed mechanism of formal unimolecular reaction $A \rightarrow B$ on a surface

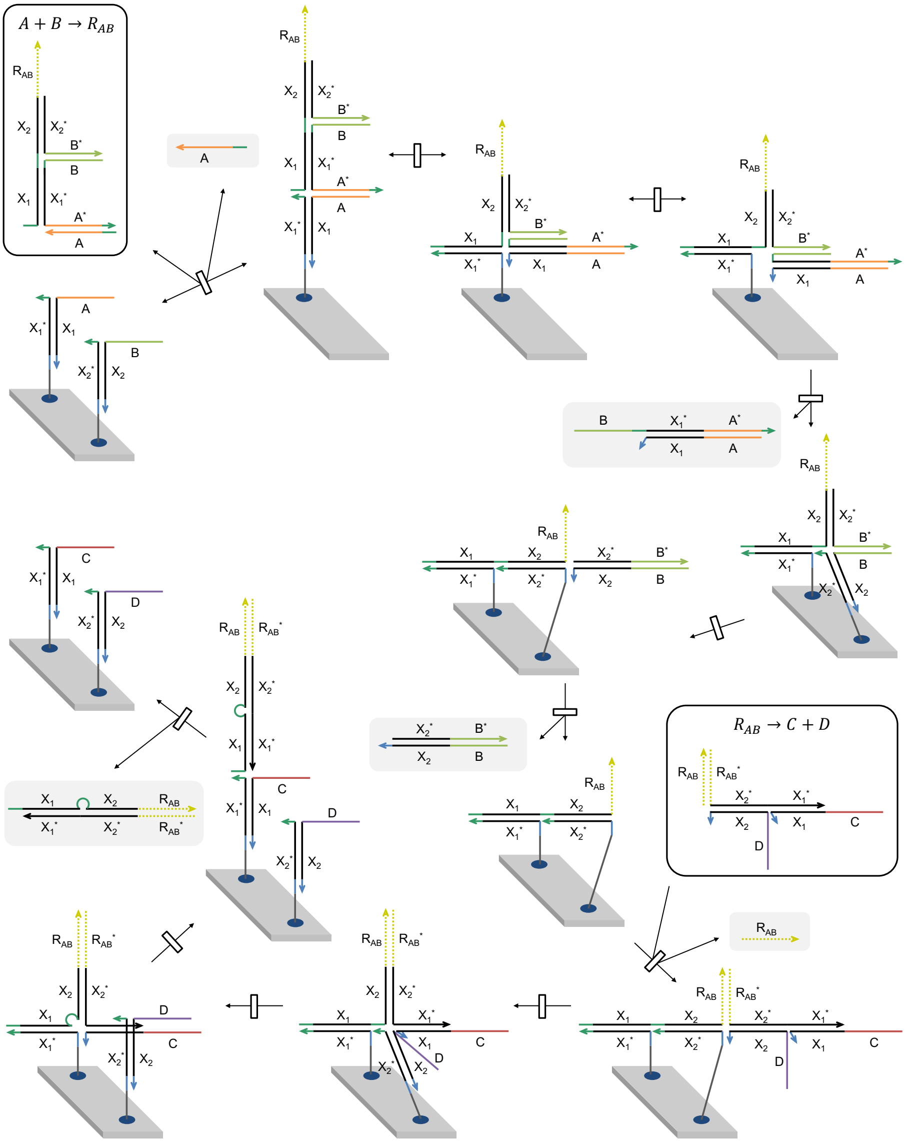


Fig. 9. Detailed mechanism of formal bimolecular reaction $A+B \rightarrow C+D$ on a surface

Abstract Modelling of Tethered DNA Circuits

Matthew R. Lakin[1], Rasmus Petersen[2], Kathryn E. Gray[2,3], and Andrew Phillips[2]

[1] Department of Computer Science, University of New Mexico, Albuquerque, NM, USA
[2] Microsoft Research, Cambridge, UK
[3] Computer Laboratory, University of Cambridge, Cambridge, UK
mlakin@cs.unm.edu, aphillip@microsoft.com

Abstract. Sequence-specific DNA interactions are a powerful means of programming nanoscale locomotion. These systems typically use a DNA track that is tethered to a surface, and molecular interactions enable a signal or cargo to traverse this track. Such low copy number systems are highly amenable to mechanized analyses such as probabilistic model checking, which requires a formal encoding. In this paper we present the first general encoding of tethered DNA species into a formal language, which allows the interactions between tethered species to be derived automatically using standard reaction rules. We apply this encoding to a previously published tethered DNA circuit architecture based on hairpin assembly reactions. This work enables automated analysis of large-scale tethered DNA circuits and, potentially, synthesis of optimized track layouts to implement specific logic functions.

1 Introduction

Nanoscale locomotion, driven by motor proteins such as kinesin and myosin, is a key component of many cellular processes [1]. Recent attempts to implement synthetic analogs of such systems typically rely on DNA components that are physically tethered to a surface, e.g., a DNA origami tile, to form a track. The intuition here is that tethered components can only interact if they are tethered in close proximity to one another, so that when a component is attached to a particular tethered track location it can only move to nearby available track locations. This approach has been used to implement a range of DNA walkers [2, 3], molecular-scale assembly lines [4], and localized DNA logic circuits [5, 6].

Since these systems typically involve small numbers of molecules, they are highly suited to formal analysis using methods such as probabilistic model checking [7]. Probabilistic model checking has previously been applied to solution-phase DNA strand displacement circuits [8] but its practical utility is limited by the state space explosion caused by large species populations. Previous work on model checking of DNA walkers [9] used a manually constructed representation of the state space, which is not scalable to larger track sizes with more reachable states. Therefore, it is important to define a general mechanism for deriving the interactions of tethered DNA species in large-scale systems. In this paper we present such a mechanism, by extending the DSD language [10, 11] with new syntactic constucts and reaction rules for encoding of tethered DNA strand displacement systems on tiles. This encoding could be used to formalize, simulate and analyze large-scale tethered DNA circuits.

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 132–147, 2014.

Domain lists without tethers	$\mathtt{S}$	$::= D_1 \cdots D_n$
Left domain lists	$\mathtt{L}$	$::= \mathtt{tether}(\mathsf{a}_1,\ldots,\mathsf{a}_n)\ \mathtt{S} \mid \mathtt{S}$
Right domain lists	$\mathtt{R}$	$::= \mathtt{S}\ \mathtt{tether}(\mathsf{a}_1,\ldots,\mathsf{a}_n) \mid \mathtt{S}$
Strands	$\mathtt{A}$	$::= \langle\mathtt{L}\rangle \mid \langle\mathtt{R}\rangle \mid \{\mathtt{L}\} \mid \{\mathtt{R}\}$
Segments (no hairpins)	$\mathtt{M}_{NH}$	$::= \{\mathtt{L}'\}\langle\mathtt{L}\rangle[\mathtt{S}]\langle\mathtt{R}\rangle\{\mathtt{R}'\}$
Segments (left hairpin)	$\mathtt{M}_{LH}$	$::= \langle\mathtt{S}'\}[\mathtt{S}]\langle\mathtt{R}\rangle\{\mathtt{R}'\}$
Segments (right hairpin)	$\mathtt{M}_{RH}$	$::= \{\mathtt{L}'\}\langle\mathtt{L}\rangle[\mathtt{S}]\{\mathtt{S}'\rangle$
Segment join operators	$\sim$	$::= \ : \mid ::$
Gates (no hairpins)	$\mathtt{G}_{NH}$	$::= \mathtt{M}_{NH} \mid \mathtt{M}_{NH} \sim \mathtt{G}_{NH}$
Gates (left hairpin)	$\mathtt{G}_{LH}$	$::= \mathtt{M}_{LH} \mid \mathtt{M}_{LH} \sim \mathtt{G}_{NH}$
Gates (right hairpin)	$\mathtt{G}_{RH}$	$::= \mathtt{M}_{RH} \mid \mathtt{G}_{NH} \sim \mathtt{M}_{RH}$
Gates (two hairpins)	$\mathtt{G}_{TH}$	$::= \mathtt{M}_{LH} \sim \mathtt{G}_{RH}$
Gates	$\mathtt{G}$	$::= \mathtt{G}_{NH} \mid \mathtt{G}_{LH} \mid \mathtt{G}_{RH} \mid \mathtt{G}_{TH}$
Species	$\mathtt{X}$	$::= \mathtt{A} \mid \mathtt{G}$
Tethered species	$\mathtt{XT}$	$::= \mathtt{X} \quad (\text{if } \textit{tethered}(\mathtt{X}))$
Untethered species	$\mathtt{XU}$	$::= \mathtt{X} \quad (\text{if } \textit{untethered}(\mathtt{X}))$
Tethered systems	$\mathtt{T}$	$::= \mathtt{XT} \mid (\mathtt{T}_1 \parallel \mathtt{T}_2)$
Mixed systems	$\mathtt{I}$	$::= \mathtt{X} \mid (\mathtt{I}_1 \parallel \mathtt{I}_2)$
Systems	$\mathtt{U}$	$::= \mathtt{XU} \mid [[\mathtt{T}]] \mid (\mathtt{U}_1 \parallel \mathtt{U}_2)$

Fig. 1. Extended syntax for DSD with tethered species, hairpins, and DNA tiles. The predicate *tethered*(X) is satisfied if X contains at least one tether, and the predicate *untethered*(X) is satisfied if X contains no tethers.

2 Abstract Specifications of Tether Locations

In an initial design of a large-scale tethered DNA circuit, the designer will not necessarily have precise locations in mind for each of the individual tethered species. Hence, in this early design phase it may be simpler to represent the relationship between tethered species abstractly, by simply specifying which tethered species are located sufficiently close to react with which other tethered species.

Here we adopt this abstract approach to the specification of the locations of tethered species. We represent physical proximity using *location tags*, which are chosen from an alphabet $\mathbb{A} = \{\mathsf{a},\mathsf{b},\mathsf{c},\ldots\}$. Each tether in a species will be annotated with a finite number of tags, and we assume that all of the species which share a particular tag are tethered such that they are close enough together to react with each other (but not with any species that do not share that tag). Hence, two tethered species can interact if they have at least one tag in common. Below, we will associate each tag with a local concentration, so that certain tethered species may interact at a faster rate than others.

3 DSD Syntax for Tethered Species

To model tethered species in the DSD language, we introduce the reserved keyword $\mathtt{tether}(\mathsf{a}_1,\ldots,\mathsf{a}_n)$ to represent a tether point that attaches the species to a surface, where $\mathsf{a}_1,\ldots,\mathsf{a}_n$ is a finite, non-empty list of location tags. This keyword is somewhat like a domain, except that it cannot be complemented and its occurrences in structures are syntactically restricted. A species with no tethers is free to diffuse in solution.

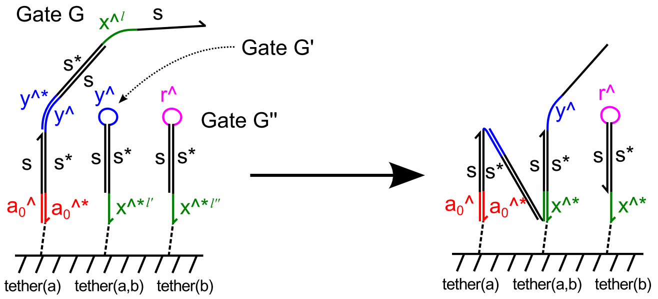
Figure 1 defines a grammar for tethered DSD systems. (For brevity, we omit module definitions and local domain declarations, which are present in the standard DSD syntax.) Here and henceforth, D ranges over domains *excluding* tethers. This enables us to syntactically limit the occurrences of tethers in the segment syntax. Strands are divided into "upper" strands ($\langle\mathtt{L}\rangle$ or $\langle\mathtt{R}\rangle$, which are rendered 5' to 3' from left to right), and "lower" strands ($\{\mathtt{L}\}$ or $\{\mathtt{R}\}$, which are rendered 3' to 5' from left to right). Note that this grammar limits single strands to at most one tether point.

Multi-strand complexes are known as "gates" in the DSD language, and are composed of one or more concatenated "segments", which have a double-stranded portion, possibly with single-stranded overhangs. From our previous work [10, 11], the general form of a segment is $\{\mathtt{L}'\}\langle\mathtt{L}\rangle[\mathtt{S}]\langle\mathtt{R}\rangle\{\mathtt{R}'\}$. Here, $\mathtt{S}$ is the double-stranded portion and the other domain sequences denote the upper and lower single-stranded overhangs, which are distinguished based on the brackets as in the case of single strands. Multiple segments may be composed using the segment join operator (:) for concatenation of the lower strand, or (::) for concatenation of the upper strand.

Here, we extend the DSD syntax with hairpin loops, which can occur at either end of a gate. This is an important extension because metastable hairpins are widely used as a fuel supply in the design of DNA nanomachines [2, 12–14], but they are not representable in the previously published DSD syntax [11]. We write $\langle\mathtt{S}\}$ for a hairpin loop at the left-hand end of a gate and $\{\mathtt{S}\rangle$ for a hairpin loop at the right-hand end of a gate. Within a hairpin loop, we list domains from 5' to 3', that is, clockwise (since, by convention, the upper strand runs 5' to 3' from left to right). The grammar includes multiple syntactic categories for gates, to ensure that hairpins can only appear at the *ends* of gate structures. Note that we omit empty overhangs when writing down gate structures and, as standard in DSD, we assume that the only single-stranded complementary domains are toeholds.

We let the metavariables $\mathtt{XT}$ and $\mathtt{XU}$ range over species (i.e., strands or gates) with and without tethers, respectively. We then define *tethered systems* $\mathtt{T}$ that consist entirely of tethered species, and *systems* $\mathtt{U}$ that consist of untethered species and *tiles* $[[\mathtt{T}]]$, which represent a DNA tile with the tethered species $\mathtt{T}$ attached to it. Hence, a system corresponds to a DSD program, in which tethered species can only occur within a tile construct. This provides a syntactic means of delimiting the occurrences of tethers in a program, and allows us to encode and simulate solutions containing many tethered circuits on many tiles. (We also define *mixed systems* $\mathtt{I}$ that may contain both tethered and untethered species—these are not considered well-formed and only appear during intermediate computations of reactions between tiles and untethered species.)

For simplicity, the grammar in Figure 1 admits gate structures with tethers in unrealistic locations at the joins between gate segments. Instead, We assume that such gates are disallowed by a subsequent well-formedness check on grammatical structures which requires that, if two neighbouring gate segments are joined along a particular strand, the domains adajcent to the join operator cannot be tethers. To simplify the semantics, we assume that hairpins are sufficiently short that nothing will bind to a domain in one of these structures. These extensions to the DSD syntax will enable us to model



```
[[ {tether(a)}[a0^ s]:[y^* s*]<x^ s>
 | {tether(a,b) x^*}[s]{y^>
 | {tether(b) x^*}[s]{r^> ]]
```

```
[[ {tether(a)}[a0^ s]:[y^* s*]::
     {tether(a,b)}[x^ s]{y^ s}
 | {tether(b) x^*}[s]{r^> ]]
```

inftags(l,G) = {a}
inftags(l',G') = {a,b}
inftags(l'',G") = {b}

interact(l,G,l',G') = true
interact(l,G,l'',G") = false

Fig. 2. Example of computing the sets of tags that influence several exposed toeholds in tethered gates G, G′ and G″, together with evaluations of the *interact* predicate to determine whether pairs of gates may interact. The gate structures are derived from the transmission line design from [6], and corresponding DSD code is presented. In this example, the domains labelled ℓ and ℓ'' have no tags in common and are therefore deemed too far apart to interact, whereas the domains labelled ℓ' and ℓ'' are both influenced by the location tag a and can therefore interact. Thus the design enforces that only neighbouring structures in the transmission line can interact.

tethered DNA circuits using hairpin fuels, as shown below. Appendix A presents additional extensions to the DSD syntax and semantics to encode internal loops and bulges (the Appendices are available for download from the corresponding authors' websites).

4 Computing Interactions between Tethered Species

In order to derive bimolecular interactions involving tethered species, we must calculate whether the domains involved are close enough to interact. Thus we must determine which tether points are exerting influence over which domains, in order to determine whether those domains are tethered close enough to interact.

4.1 Labels

To identify the particular domains involved in an interaction, we fix a countably infinite set Λ of *labels* $\ell_1, \ell_2, \ldots$ and assume that every occurrence of every domain is associated with a *globally unique* label. For example, the gate $\{\mathtt{T}^{\wedge*}\}[\mathtt{X}]\langle\mathtt{X}\rangle$ might be labelled as $\{\mathtt{T}^{\wedge*\ell_1}\}[\mathtt{X}^{\ell_2}]\langle\mathtt{X}^{\ell_3}\rangle$. Domain labels are not part of the user-visible language syntax, rather, they are an internal mechanism to distinguish between multiple instances of the same domain when calculating which tether is exerting influence over that domain. Hence, the particular assignment of labels to domain occurrences is not important, provided that they are globally unique. We do not state labels explicitly unless they are required in a reaction rule.

4.2 Computing Bimolecular Interactions with Tethered Species

The key operation in the tethered semantics is to compute the set of tethers that are currently exerting influence on a particular domain labelled with ℓ. We write $\mathit{inftags}(\ell, \mathtt{G})$ for the set of location tags that influence the domain labelled by ℓ in the gate $\mathtt{G}$. To compute this we traverse the structure of the species, starting from the labelled domain in question and moving outwards, and assume that the first tethers that we find in either direction (written as $\mathit{LHtags}(\ell, \mathtt{G})$ and $\mathit{RHtags}(\ell, \mathtt{G})$) are those exerting influence on the position of the labelled domain:

$$\mathit{inftags}(\ell, \mathtt{G}) \triangleq \bigcup(\mathit{LHtags}(\ell, \mathtt{G}) \cup \mathit{RHtags}(\ell, \mathtt{G}))$$

In this definition, the inner union is over sets of tag lists, while the outer union combines the resulting set of tag lists into a single set of location tags. The $\mathit{inftags}(\ell, \mathtt{G})$ function, and the case for tethered strands, can be fully defined as follows.

Given a segment $\mathtt{M}$, we write $\mathit{LHtags}(\mathtt{M})$ and $\mathit{RHtags}(\mathtt{M})$ for the sets of tag lists $\mathtt{a}_1, \ldots, \mathtt{a}_n$ such that $\mathtt{tether}(\mathtt{a}_1, \ldots, \mathtt{a}_n)$ appears on the left or right overhang of the segment $\mathtt{M}$, respectively. Furthermore, we write $\mathit{tags}(\mathtt{M})$ for the set of *all* tag lists $\mathtt{a}_1, \ldots, \mathtt{a}_n$ such that $\mathtt{tether}(\mathtt{a}_1, \ldots, \mathtt{a}_n)$ appears *anywhere* in $\mathtt{M}$.

We now define functions $\mathit{LHtags}(\mathtt{G})$ and $\mathit{RHtags}(\mathtt{G})$, which return the leftmost and rightmost tag sets found by searching a gate $\mathtt{G}$ segment-wise, respectively. These functions can be defined by recursion on the structure of gates, as follows.

$$\mathit{LHtags}(\mathtt{M}) \triangleq \mathit{tags}(\mathtt{M}) \qquad \mathit{LHtags}(\mathtt{M} \sim \mathtt{G}) \triangleq \begin{cases} \mathit{tags}(\mathtt{M}) & \text{if } \mathit{tags}(\mathtt{M}) \neq \varnothing \\ \mathit{LHtags}(\mathtt{G}) & \text{otherwise.} \end{cases}$$

$$\mathit{RHtags}(\mathtt{M}) \triangleq \mathit{tags}(\mathtt{M}) \qquad \mathit{RHtags}(\mathtt{G} \sim \mathtt{M}) \triangleq \begin{cases} \mathit{tags}(\mathtt{M}) & \text{if } \mathit{tags}(\mathtt{M}) \neq \varnothing \\ \mathit{RHtags}(\mathtt{G}) & \text{otherwise.} \end{cases}$$

We now define the first tag sets found by searching outwards from a particular labelled domain in a gate structure. Suppose that the domain in question has label ℓ, and that the gate $\mathtt{G}$ has the form $\mathtt{G}_L \sim \mathtt{M} \sim \mathtt{G}_R$, where $\mathtt{M}$ is the segment containing the domain with label ℓ. Then, we define functions *LHtags* and *RHtags* that compute the first tag sets found in a segment-wise search outward from the segment in $\mathtt{G}$ containing ℓ, as follows.

$$\mathit{LHtags}(\ell, \mathtt{G}_L \sim \mathtt{M} \sim \mathtt{G}_R) \triangleq \begin{cases} \mathit{LHtags}(\mathtt{M}) & \text{if } \mathit{LHtags}(\mathtt{M}) \neq \varnothing \\ \mathit{RHtags}(\mathtt{G}_L) & \text{otherwise.} \end{cases}$$

$$\mathit{RHtags}(\ell, \mathtt{G}_L \sim \mathtt{M} \sim \mathtt{G}_R) \triangleq \begin{cases} \mathit{RHtags}(\mathtt{M}) & \text{if } \mathit{RHtags}(\mathtt{M}) \neq \varnothing \\ \mathit{LHtags}(\mathtt{G}_R) & \text{otherwise.} \end{cases}$$

In the case where $\mathtt{G}$ has the form $\mathtt{M} \sim \mathtt{G}_R$, where $\mathtt{M}$ is the segment containing the domain with label ℓ, the definitions are as follows.

$$\begin{aligned}
\mathit{LHtags}(\ell, \mathtt{M} \sim \mathtt{G}_R) &\triangleq \mathit{LHtags}(\mathtt{M}) \\
\mathit{RHtags}(\ell, \mathtt{M} \sim \mathtt{G}_R) &\triangleq \begin{cases} \mathit{RHtags}(\mathtt{M}) & \text{if } \mathit{RHtags}(\mathtt{M}) \neq \varnothing \\ \mathit{LHtags}(\mathtt{G}_R) & \text{otherwise.} \end{cases}
\end{aligned}$$

Finally, in the case where $\mathtt{G}$ has the form $\mathtt{G}_L \sim \mathtt{M}$, where $\mathtt{M}$ is the segment containing the domain with label ℓ, the definitions are as follows.

$$\begin{aligned}
\mathit{LHtags}(\ell, \mathtt{G}_L \sim \mathtt{M}) &\triangleq \begin{cases} \mathit{LHtags}(\mathtt{M}) & \text{if } \mathit{LHtags}(\mathtt{M}) \neq \varnothing \\ \mathit{RHtags}(\mathtt{G}_L) & \text{otherwise.} \end{cases} \\
\mathit{RHtags}(\ell, \mathtt{G}_L \sim \mathtt{M}) &\triangleq \mathit{RHtags}(\mathtt{M}).
\end{aligned}$$

These functions are used to define $\mathit{inftags}(\ell, \mathtt{G})$, as shown above. Furthermore, since single strands may also be tethered, we must also define a similar function for strands: assuming that the label ℓ appears in the strand $\mathtt{A}$, we simply let $\mathit{inftags}(\ell, \mathtt{A})$ return the union of all tag lists $\mathtt{a}_1, \ldots, \mathtt{a}_n$ such that $\mathtt{tether}(\mathtt{a}_1, \ldots, \mathtt{a}_n)$ appears in $\mathtt{A}$. Our well-formedness conditions on the occurrences of tethers mean that $\mathit{inftags}(\ell, \mathtt{A})$ must contain at tags from at most one tag set, as the syntax only allows a tether at one end of a single strand.

The $\mathit{inftags}(\ell, \mathtt{X})$ function, where $\mathtt{X}$ could be a gate $\mathtt{G}$ or a strand $\mathtt{A}$, will be used below to define interaction rules for tethered species. Figure 2 presents the result of computing the sets of tags that influence exposed toeholds in an example interaction between a tethered gate and a tethered strand.

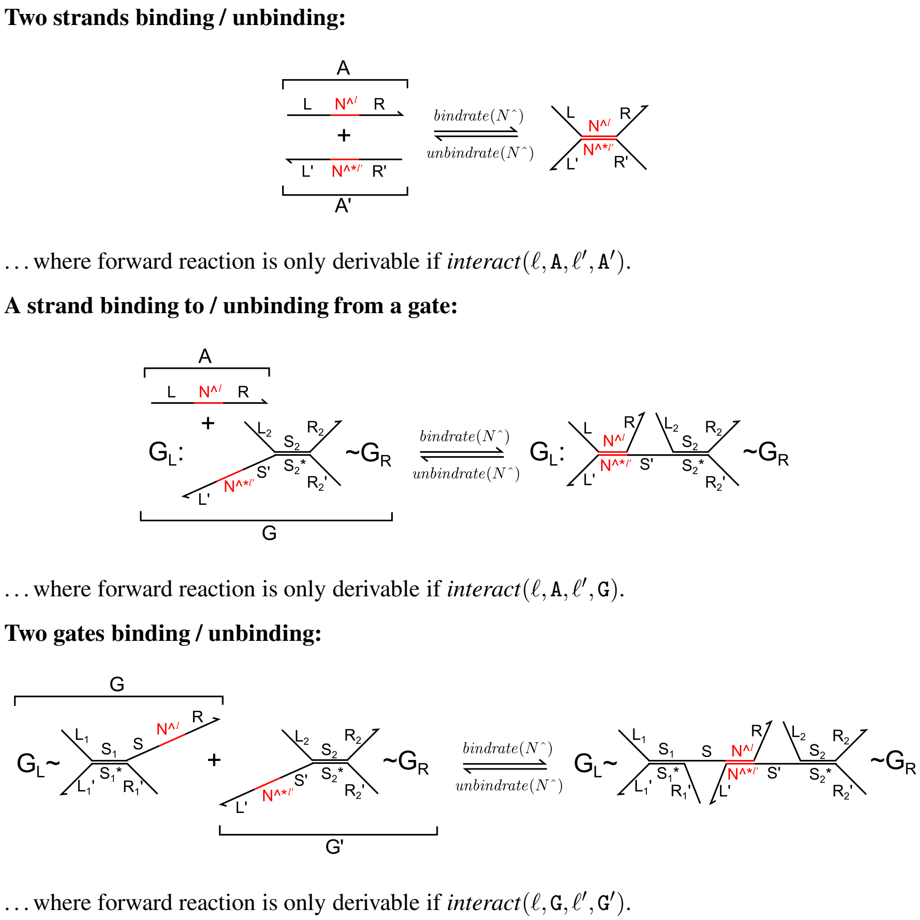
We can now define the additional tests, expressed in terms of the *inftags* function, that govern bimolecular interactions involving species that may be tethered. If species $\mathtt{X}_1$ and $\mathtt{X}_2$ may interact via toeholds with labels ℓ_1 and ℓ_2, the interaction is possible if the predicate $\mathit{interact}(\mathtt{X}_1, \ell_1, \mathtt{X}_2, \ell_2)$ is satisfied, which is defined as follows.

$$\begin{aligned}
\mathit{interact}(\mathtt{X}_1, \ell_1, \mathtt{X}_2, \ell_2) \triangleq\ & (\mathit{inftags}(\mathtt{X}_1, \ell_1) \cap \mathit{inftags}(\mathtt{X}_2, \ell_2)) \neq \varnothing \\
& \vee \mathit{inftags}(\mathtt{X}_1, \ell_1) = \varnothing \vee \mathit{inftags}(\mathtt{X}_2, \ell_2) = \varnothing
\end{aligned}$$

The first clause of the definition covers the case when the reactants $\mathtt{X}_1$ and $\mathtt{X}_2$ are both tethered, and when the interacting toehold domains have one or more location tags in common. This means that the species are tethered close enough together to interact. The two remaining clauses cover the cases when one or both reactants contain no tethers, and are therefore freely diffusing. In these cases, the reaction is always possible because a freely diffusing species can always find any other species to interact with. This definition will be used below to formalize the reaction rules for tethered species.

5 Reaction Rules for Tethered Species

We write $\mathtt{G}_{\mathfrak{L}}$ for any gate capable of serving as a left-hand context, that is, either $\mathtt{G}_{NH}$ (no hairpins) or $\mathtt{G}_{LH}$ (hairpin present only on the left-hand side), or an empty context.



Two strands binding / unbinding:

. . . where forward reaction is only derivable if *interact*$(\ell, \mathtt{A}, \ell', \mathtt{A}')$.

A strand binding to / unbinding from a gate:

. . . where forward reaction is only derivable if *interact*$(\ell, \mathtt{A}, \ell', \mathtt{G})$.

Two gates binding / unbinding:

. . . where forward reaction is only derivable if *interact*$(\ell, \mathtt{G}, \ell', \mathtt{G}')$.

Fig. 3. Bimolecular DSD reaction rules for tethered species

Similarly, we write $\mathtt{G}_{\mathfrak{R}}$ for any gate capable of serving as a right-hand context, that is, either $\mathtt{G}_{NH}$ (no hairpins) or $\mathtt{G}_{RH}$ (hairpin present only on the right-hand side), or an empty context. In this section we present rules that define the possible reactions between species, including permissible structural contexts. Each reaction rule is labelled with the reaction rate constant: we assume the existence of functions *bindrate* and *unbindrate* that map each toehold domain $N\hat{}$ (and its complement $N\hat{}^*$) to the associated binding rate constant *bindrate*$(N\hat{})$ and unbinding rate constant *unbindrate*$(N\hat{})$ respectively, and rate constants ρ_F for "fast" unimolecular reactions (e.g., branch migration) and ρ_S for "slow" unimolecular reactions (e.g., formation of internal loops, which involves internal diffusion).

Figure 3 presents bimolecular binding rules for strands and gates, and the corresponding unimolecular unbinding rules. Since these species may be tethered, the bimolecular rules use the *interact* predicate defined in Section 4.2 as a crucial additional test, so that two tethered species may only bind if they are tethered close enough to-

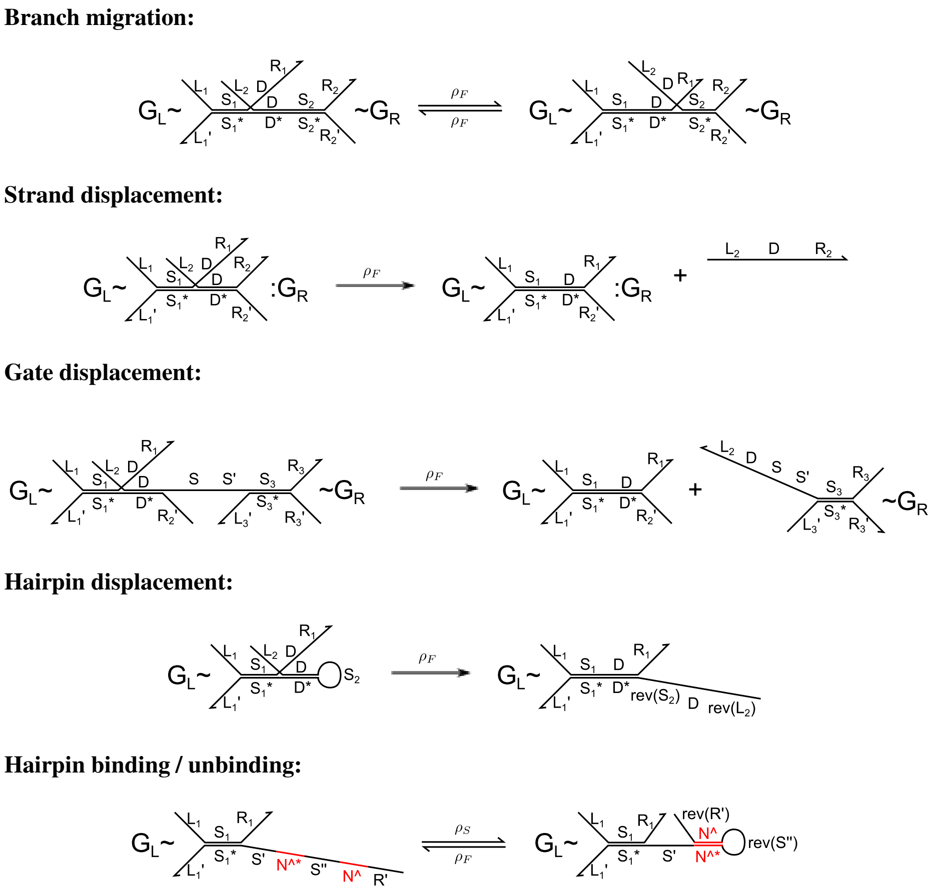

Branch migration:
G_L~ L_1 L_2 D R_1 S_1 D S_2 R_2 S_1* D* S_2* L_1' R_2' ~G_R
ρ_F
ρ_F
G_L~ L_1 L_2 D R_1 R_2 S_1 D S_2 S_1* D* S_2* L_1' R_2' ~G_R
Strand displacement:
G_L~ L_1 L_2 D R_1 R_2 S_1 D S_1* D* L_1' R_2' :G_R
ρ_F
G_L~ L_1 S_1 D R_1 S_1* D* L_1' R_2' :G_R
+
L_2 D R_2
Gate displacement:
G_L~ L_1 L_2 D R_1 S_1 D S S' S_3 R_3 S_1* D* S_3* L_1' R_2' L_3' R_3' ~G_R
ρ_F
G_L~ L_1 S_1 D R_1 S_1* D* L_1' R_2'
+
L_2 D S S' S_3 R_3 S_3* L_3' R_3' ~G_R
Hairpin displacement:
G_L~ L_1 L_2 D R_1 S_1 D S_2 S_1* D* L_1'
ρ_F
G_L~ L_1 S_1 D R_1 S_1* D* rev(S_2) D rev(L_2) L_1'
Hairpin binding / unbinding:
G_L~ L_1 S_1 R_1 S_1* S' N^* S" N^ R' L_1'
ρ_S
ρ_F
G_L~ L_1 S_1 R_1 rev(R') N^ rev(S") S_1* S' N^* L_1'

Fig. 4. Unimolecular DSD reaction rules, including additional rules to model hairpins. Note that the DSD convention is to list domains from left to right on the page, which corresponds to 5' to 3' for "upper" strands but 3' to 5' for "lower" strands. The inclusion of hairpins in the syntax muddies this distinction somewhat, and we must use the "rev" keyword to reverse the appropriate domain sequences in hairpin reactions.

gether. Figure 4 recaps the basic unimolecular reaction rules from the DSD semantics and presents additional rules to define intramolecular hairpin opening (displacement) and (un)binding reactions. (ASCII representations of all rules, using the DSD syntax, are presented in Appendix B.) Note that the formation rule for hairpins is an instance of the *remote toehold* design concept [15]. To formalize the interactions of DNA tiles as tethering surfaces in the DSD language, we require the following rules, to turn the reactions of tethered species into reactions involving the corresponding tile species.

$$(\textsc{tInt})\ \frac{\mathtt{T} \xrightarrow{\rho} \mathtt{T}'}{[[\mathtt{T}]] \xrightarrow{\rho} [[\mathtt{T}']]} \qquad (\textsc{tBnd})\ \frac{\mathtt{XU} \parallel \mathtt{T} \xrightarrow{\rho} \mathtt{T}'}{\mathtt{XU} \parallel [[\mathtt{T}]] \xrightarrow{\rho} [[\mathtt{T}']]} \qquad (\textsc{tUbnd})\ \frac{\mathtt{T} \xrightarrow{\rho} \mathtt{T}' \parallel \mathtt{XU}}{[[\mathtt{T}]] \xrightarrow{\rho} [[\mathtt{T}']] \parallel \mathtt{XU}}$$

Rule (tInt) handles direct interactions between tethered species on the tile, since all reactants and products are tethered to the tile (see syntax definitions above). Rule (tBnd) handles the case where an incoming diffusing species XU (which could be a strand or a gate) binds to a tile. Similarly, rule (tUbnd) covers the case where a reaction on a tile produces an untethered species that is now free to diffuse. Note that the premisses of rules (tBnd) and (tUbnd) are instances of *mixed systems* of tethered and untethered species, but the final derived reactions in all cases involve well-formed *systems* in which all and only tethered species are encapsulated within tile constructs. These rules do not allow any crosstalk between two tiles—a species that is tethered to a tile can only interact with another species tethered to the same tile, or with a freely diffusing species. This is a reasonable assumption because of the slow diffusion rate of large DNA tiles compared to non-tile species, and means that two tile-based circuits can only communicate via a freely diffusing signal. Hence, all interactions taking place inside a tile are modeled as unimolecular reactions. Furthermore, the entire tile in a particular configuration must be used as the reactant species, to enable accurate modelling and simulation of populations of tiles. Finally, some additional contextual rules are required to complete the definition of the semantics: these are presented in Appendix C.

6 Calculating the Propensities of Tethered Interactions

For simulations or probabilistic model checking of tethered circuits, we must compute the propensity of every possible interaction in the system, including tethered interactions. In mass action kinetics, the propensity, p, of a bimolecular reaction with reactants $\mathtt{X}_1$ and $\mathtt{X}_2$ and rate constant, k, is given by $p \triangleq k \times [\mathtt{X}_1] \times [\mathtt{X}_2]$, where $[\mathtt{X}_i]$ is the concentration of species $\mathtt{X}_i$. In tethered DSD systems, we use this expression for the propensity of bimolecular reactions in which both reactants are freely diffusing or precisely one reactant is tethered. In the latter case, we justify the use of this expression because the tiles to which the tethered species are attached are assumed to be well-mixed in the solution.

For bimolecular reactions involving two tethered reactants, however, this expression is not valid because tethered species do not satisfy the well-mixed assumption of mass action kinetics. To compute the propensities of bimolecular interactions between two tethered species, we use the concept of "local concentration" developed in previous work on the kinetics of biomolecular interactions between tethered species [5,15]. This approach approximates the corresponding rates by computing the volume swept out by flexible tethered strands, to estimate the probability that the two species will be close enough to interact at a given point in time. For example, Genot *et al.* [15] computed local concentrations of $\sim 1 \times 10^5$ nM for localized strand displacement reactions.

To incorporate this theory into our tethered DSD framework, we assume the existence of a function lc that maps every location tag a to the local concentration for interactions occurring between species influenced by that tag. A higher value for the local concentration means that species sharing that tag are tethered relatively close to each other and will therefore interact at a faster rate. Then, for a bimolecular reaction between two tethered species $\mathtt{X}_1$ and $\mathtt{X}_2$ that interact via domains labelled ℓ_1 and ℓ_2 with rate constant k, we compute the reaction propensity, p, as $p \triangleq k \times \max(lc(\mathsf{a}_1), \ldots, lc(\mathsf{a}_n))$, where

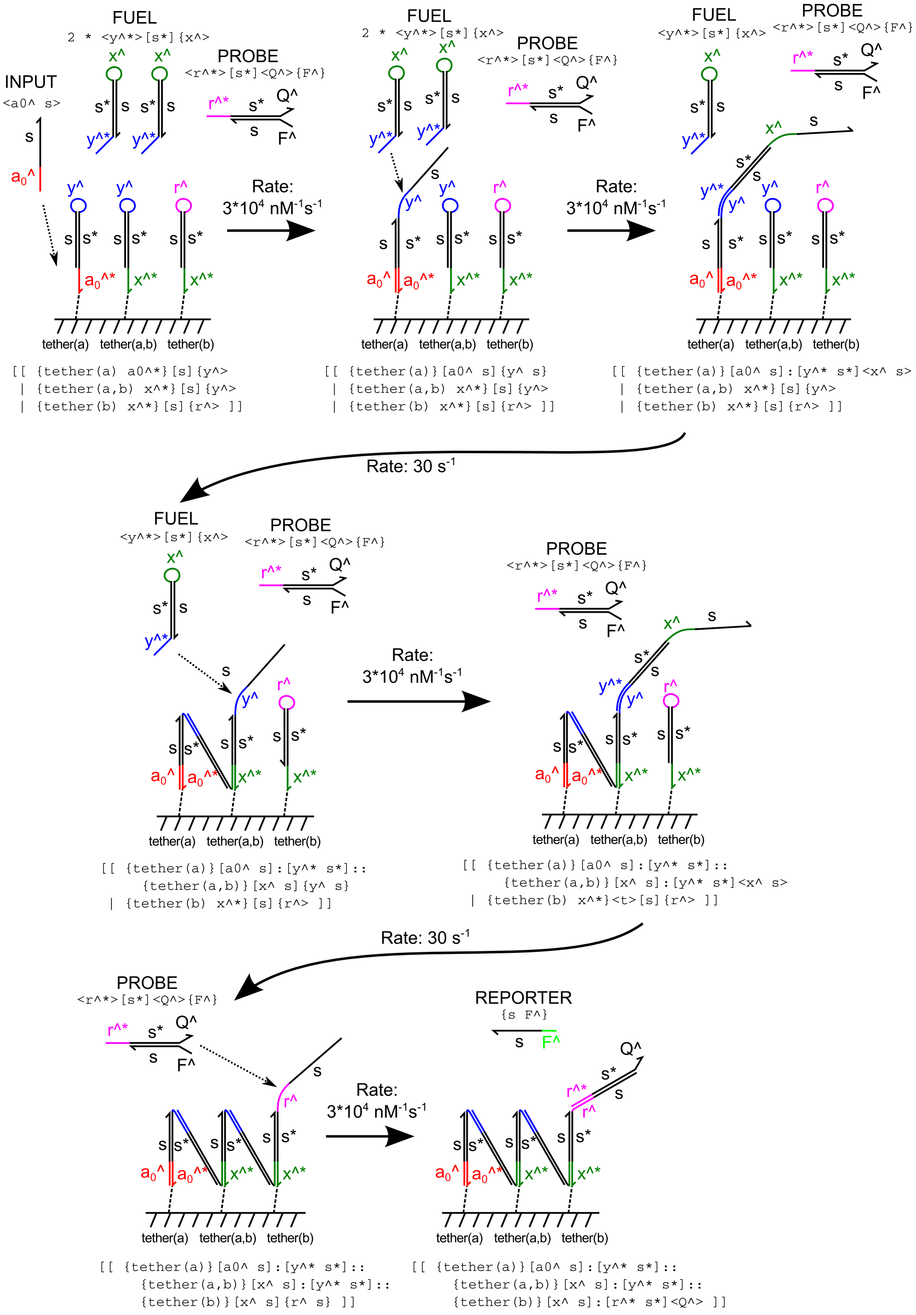
$\mathtt{a}_1,\ldots,\mathtt{a}_n = \mathit{inftags}(\ell_1,\mathtt{X}_1) \cap \mathit{inftags}(\ell_2,\mathtt{X}_2)$. According to the rules from Figure 3, the bimolecular reaction can only occur if $\mathit{inftags}(\ell_1,\mathtt{X}_1) \cap \mathit{inftags}(\ell_2,\mathtt{X}_2)$ is non-empty. If there are multiple shared tags in this intersection, we use the largest of the corresponding local concentrations. We take this design decision because multiple shared tags do not enable additional mechanisms for a given reaction to occur—instead, they simply impose further constraints on how tethered species could be placed on a tile so that they will interact with the specified local concentrations. Hence the largest local concentration is the dominant one when computing the rate of a given interaction. Thus we are able to model the rates of bimolecular interactions between tethered species, enabling simulation and probabilistic model checking of solutions of tile-based circuits.

7 Examples

As an example application of our abstract modelling framework for tethered DNA circuits, we encoded the hairpin-based tethered circuit architecture from [6] into our extended DSD language. Figure 5 presents the DSD code and reduction sequence for the three-stator transmission line system from [6]. Note that the stators are all contained within a syntactic tile construct, and that all of the tags are assigned the same local concentration, i.e., the signal is passed between each pair of stators at the same rate. Furthermore, the distribution of location tags prevents the fuel bound to the first stator from binding directly to the third stator—hence, the signal must be passed sequentially along the stators with none being missed. Importantly, this causal dependence between the binding reactions can be deduced automatically by the DSD compiler, thanks to the use of location tags. Finally, a freely-diffusing strand displacement probe produces an increase in bulk fluorescence to indicate that the signal has reached the last stator. Figure 6 encodes a threshold-based spatial AND gate design from [6] by using different local concentration values for different location tags. The resulting reaction propensities mean that there is a high probability that the first input will bind to the threshold rather than the output stator. If this happens, the second input is required to trigger the output, achieving the desired AND logic. However, there is a non-zero probability that the first input will erroneously activate the output without the second input. We have implemented our syntax and semantics for tethered systems in the Visual DSD software tool [16], and Appendix D presents simulation and state space analysis results from encoding the examples from this section in the latest version of Visual DSD.

8 Discussion

To summarize, we have defined an encoding of tethered DNA circuits on tiles in the DSD language, which uses *location tags* to abstractly specify the pattern of tethering, and therefore the pattern of possible interactions between tethered species. We have extended the DSD syntax to include hairpins, which are often used as fuel for DNA nanomachines, and also to include DNA tiles, which colocalize tethered species in solution. We have demonstrated a formalization of the hairpin-based tethered circuit design from [6]. Our abstract representation strategy removes the need to explicitly formalize the layout of the track and the structure of the supporting surface, which could



```
[[ {tether(a) a0^*}[s]{y^>
 | {tether(a,b) x^*}[s]{y^>
 | {tether(b) x^*}[s]{r^> ]]
```

```
[[ {tether(a)}[a0^ s]{y^ s}
 | {tether(a,b) x^*}[s]{y^>
 | {tether(b) x^*}[s]{r^> ]]
```

```
[[ {tether(a)}[a0^ s]:[y^* s*]<x^ s>
 | {tether(a,b) x^*}[s]{y^>
 | {tether(b) x^*}[s]{r^> ]]
```

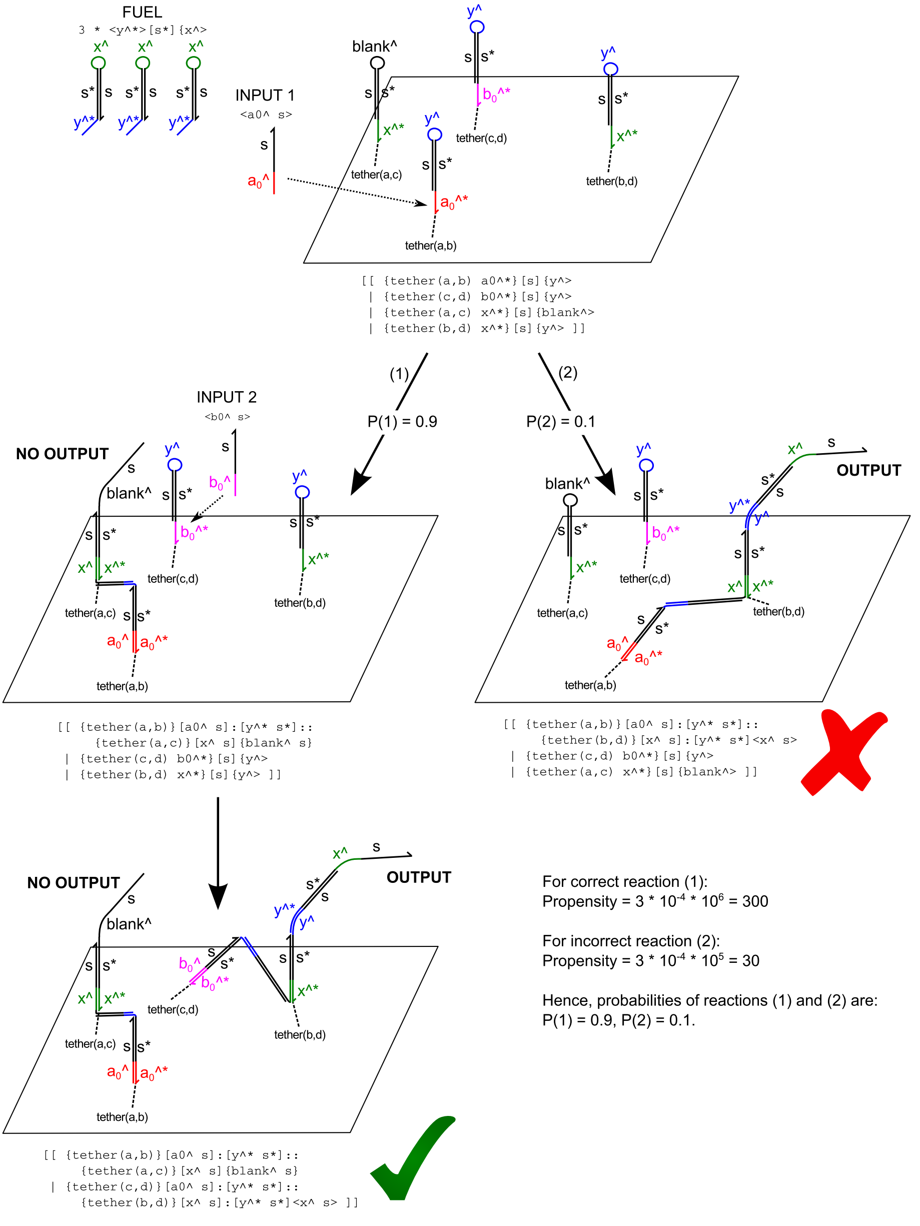
```
[[ {tether(a)}[a0^ s]:[y^* s*]::
     {tether(a,b)}[x^ s]{y^ s}
 | {tether(b) x^*}[s]{r^> ]]
```

```
[[ {tether(a)}[a0^ s]:[y^* s*]::
     {tether(a,b)}[x^ s]:[y^* s*]<x^ s>
 | {tether(b) x^*}<t>[s]{r^> ]]
```

```
[[ {tether(a)}[a0^ s]:[y^* s*]::
     {tether(a,b)}[x^ s]:[y^* s*]::
     {tether(b)}[x^ s]{r^ s} ]]
```

```
[[ {tether(a)}[a0^ s]:[y^* s*]::
     {tether(a,b)}[x^ s]:[y^* s*]::
     {tether(b)}[x^ s]:[r^* s*]<Q^> ]]
```

Fig. 5. DSD encoding of a variant on the full three-stator transmission line system from Figure 7 of [6]. To derive the reaction rate constants, we assume that all toeholds bind at the DSD default rate (3×10^{-4} nM^{-1} s^{-1}) and that $lc(a) = lc(b) = 1 \times 10^5$ nM, giving a tethered interaction rate of $30\,\mathrm{s}^{-1}$.



```
3 * <y^*>[s*]{x^>
```

```
<a0^ s>
```

```
[[ {tether(a,b) a0^*}[s]{y^>
 | {tether(c,d) b0^*}[s]{y^>
 | {tether(a,c) x^*}[s]{blank^>
 | {tether(b,d) x^*}[s]{y^> ]]
```

```
<b0^ s>
```

```
[[ {tether(a,b)}[a0^ s]:[y^* s*]::
     {tether(a,c)}[x^ s]{blank^ s}
 | {tether(c,d) b0^*}[s]{y^>
 | {tether(b,d) x^*}[s]{y^> ]]
```

```
[[ {tether(a,b)}[a0^ s]:[y^* s*]::
     {tether(b,d)}[x^ s]:[y^* s*]<x^ s>
 | {tether(c,d) b0^*}[s]{y^>
 | {tether(a,c) x^*}[s]{blank^> ]]
```

```
[[ {tether(a,b)}[a0^ s]:[y^* s*]::
     {tether(a,c)}[x^ s]{blank^ s}
 | {tether(c,d)}[a0^ s]:[y^* s*]::
     {tether(b,d)}[x^ s]:[y^* s*]<x^ s> ]]
```

For correct reaction (1):
Propensity = 3 * 10^{-4} * 10^{6} = 300

For incorrect reaction (2):
Propensity = 3 * 10^{-4} * 10^{5} = 30

Hence, probabilities of reactions (1) and (2) are:
P(1) = 0.9, P(2) = 0.1.

Fig. 6. DSD encoding of threshold-based spatial AND gate system from Figure 9 of [6]. We assume that input 1 arrives first, followed by input 2. The two possible trajectories for the system are outlined: one where the first input correctly binds to the threshold, and one where the first input erroneously triggers the output. To derive the reaction rate constants, we assume that all toeholds bind at the DSD default rate ($3 \times 10^{-4}\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$), and that $lc(a) = lc(c) = 1 \times 10^{6}\,\mathrm{nM}$ and $lc(b) = lc(d) = 1 \times 10^{5}\,\mathrm{nM}$. The differing local concentrations produce a thresholding effect that gives AND logic.

be a complex DNA nanostructure that is non-trivial to represent in a formal language. The inclusion of DNA tiles in the language provides a means of encapsulating tethered species such that multiple tethered circuits can be simulated in a single solution. The result is a powerful tool for modelling and verifying more sophisticated tethered systems, e.g., to analyze the possible routes taken by walkers in a multi-track system [9].

8.1 Abstractions for Tethered Circuit Design

For detailed design of tethered DNA circuits, a coordinate-based system for specifying the absolute positions of tethered components on a surface, e.g., a DNA origami tile, would be required. Ideally, this would be paired with a graphical design tool, so that the user could draw out the desired tether geometry directly, and could be integrated with existing DNA origami design tools such as caDNAno [17].

However, the coordinate-based approach requires a highly general biophysical model to predict the interaction rates of arbitrary DSD-representable structures with arbitrary toehold and tether locations. Previous calculations [5, 15] have derived expressions for tethered interaction rates for particular structures and tethering geometries, whereas to cope with the full generality of the DSD metalanguage a far more comprehensive physical model would be required.

In the absence of such a model, we chose a level of abstraction that is similar to the “channel-based” approach to specifying inter-process interactions in modelling languages such as the stochastic π-calculus [18, 19]. In the channel-based approach, all possible interactions between processes must be provided explicitly by the design via the mechanism of channel sharing. In this paper we have moved away from that idea to some extent by using the DSD reduction semantics to derive certain interactions between species, however, we still rely on a channel-like approach to specify which tethered species are close enough to react according to the reduction rules, via the mechanism of shared location tags.

By requiring the modeller to directly associate location concentrations with the various location tags, we shift the burden of computing the local concentrations to the modeller, who can perform structure-specific analyses to determine a reasonable value for the local concentration value, or alternatively fit these rate constants directly to experimental data, if available. Hence, a fully general biophysical model of the dynamics of tethered toehold interactions is not required. This approach gives a high degree of modelling flexibility, allowing measured rates to be included directly where available, or to be estimated using a biophysical model [5, 15]. However, the need to specify all possible interactions between tethered species means that the user must have some idea of the desired track geometry before encoding the system in DSD. Furthermore, potentially undesired interactions, such as track-jumping behaviour in molecular walker systems [9], cannot be inferred automatically by the compiler.

Hence, we envision that this method for the specification of tethered circuit behaviour will form but one layer of an abstraction hierarchy for the design and simulation of tethered DNA circuits, similar to our approach to the semantics of strand displacement reactions [11]. We see this model as sitting atop a more detailed coordinate-level specification, as described above. A geometric interpretation of location tags is that each tag corresponds to a point, whose physical coordinate is the average of the physical coordinates of the tether locations that share that tag. If we assume the existence of a detailed, realistic model of tethered reaction kinetics, this geometric interpretation could form the basis of a compilation phase that takes a tethered system specified abstractly using location tags and computes possible physical coordinates for each tether location, producing a concrete design suitable for experimental implementation. This may involve an iterative optimization routine to find tether coordinates that satisfy the constraints specified in the abstract model.

Alternatively, coordinate-level specifications could be translated back into the abstract domain for ease of analysis—this process could automatically generate the location tag-based encoding without further input from the user. Furthermore, these translations could be combined so that a tethered circuit design specified in the abstract domain can be compiled into a detailed, coordinate-based representation and subsequently lifted back into the abstract domain. This process would exploit the detailed model of tethered species to detect any spurious interactions between components in an abstractly specified circuit, and could be iterated to refine the tether geometry to minimize or eliminate the spurious interactions. This abstraction hierarchy could be extended further by implementing automated layout algorithms that directly compile logical specifications into geometrically arranged tracks that execute the corresponding computation with minimal spurious interactions.

8.2 Molecular Spiders

Another potential approach to implementing nanoscale locomotion is via multivalent catalytic walkers known as *molecular spiders*. These comprise multiple "legs" each of which is a catalytically active DNAzyme [20], all attached to a rigid body such as a streptavidin molecule. Molecular spiders move in a biased random walk due to cleavage of substrates displayed on a surface, and have been realized experimentally [21,22]. Previous work on computational analysis of molecular spider behaviour has used models ranging from the physically detailed [23, 24] to the more abstract [25, 26]. These simpler models have enabled computational studies of various effects, including cooperative nanoscale search due to self-avoidance [27] and maze navigation [28]. The latter is of particular interest with regard to our emphasis on the verification of track designs for molecular walkers. However, to model these systems in our framework would require us to extend the DSD framework further to model DNAzyme-catalyzed substrate cleavage reactions. Furthermore, the mechanism of spider motion implies that, for a given body position, any unattached leg has the choice of a number of potential attachment points. Encoding this mechanism using the approach proposed in this paper would require a very large number of interaction tags, because a separate tag would be needed for each pair of displayed substrates S_1 and S_2 that are sufficiently close together that a spider with a leg attached to S_1 can attach another leg to S_2. Hence, the resulting system would be rather cumbersome to simulate. Therefore, we believe that existing methods of simulating molecular spider dynamics using custom Monte Carlo simulation routines [23–26] are more practical than encoding them in our framework. Our work is more suited to encoding of tethered circuits whose interactions are constrained by the geometry of the track layout [5, 6].

8.3 Representable Structures

As the set of DSD-representable structures grows, we will gain increased power and flexibility for the design of tethered reaction systems. In particular, by enabling automatic, integrated compilation of enzymatic reactions, such as restriction enzyme and nickase reactions, we hope to model an important class of DNA walkers powered by enzymatic reactions, e.g., as in [3]. We also hope to combine this work with a formalization of dendritic DSD structures to enable simulation of tethered logic circuits with fan-in where both inputs must bind simultaneously [5], although several examples, such as the threshold-based AND circuit described in Section 7, do not require this extension. Finally, including four-way branch migration would enable us to encode additional published DNA walker designs [2].

Acknowledgments. The authors thank Filippo Polo for his work on the DSD implementation of the tethered semantics. This material is based upon work supported by the National Science Foundation under grants 1028238 and 1318833. M.R.L. gratefully acknowledges support from the New Mexico Cancer Nanoscience and Microsystems Training Center (NIH/NCI grant 5R25CA153825).

References

1. Vale, R.D.: The molecular motor toolbox for intracellular transport. Cell 112(4), 467–480 (2003)
2. Muscat, R.A., Bath, J., Turberfield, A.J.: A programmable molecular robot. Nano Lett. 11(3), 982–987 (2011)
3. Wickham, S.F.J., Bath, J., Katsuda, Y., Endo, M., Hidaka, K., Sugiyama, H., Turberfield, A.J.: A DNA-based molecular motor that can navigate a network of tracks. Nature Nanotech. 7, 169–173 (2012)
4. Gu, H., Chao, J., Xiao, S.-J., Seeman, N.C.: A proximity-based programmable DNA nanoscale assembly line. Nature 465, 202–205 (2010)
5. Chandran, H., Gopalkrishnan, N., Phillips, A., Reif, J.: Localized hybridization circuits. In: Cardelli, L., Shih, W. (eds.) DNA 17. LNCS, vol. 6937, pp. 64–83. Springer, Heidelberg (2011)
6. Muscat, R.A., Strauss, K., Ceze, L., Seelig, G.: DNA-based molecular architecture with spatially localized components. In: Proceedings of ISCA 2013 (2013)
7. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theor. Comput. Sci. 319(3), 239–257 (2008)
8. Lakin, M.R., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of DNA strand displacement devices using probabilistic model checking. J. R. Soc. Interface 9(72), 1470–1485 (2012)
9. Dannenberg, F., Kwiatkowska, M., Thachuk, C., Turberfield, A.J.: DNA walker circuits: Computational potential, design, and verification. In: Soloveichik, D., Yurke, B. (eds.) DNA 19. LNCS, vol. 8141, pp. 31–45. Springer, Heidelberg (2013)
10. Phillips, A., Cardelli, L.: A programming language for composable DNA circuits. J. R. Soc. Interface 6(suppl. 4), S419–S436 (2009)
11. Lakin, M.R., Youssef, S., Cardelli, L., Phillips, A.: Abstractions for DNA circuit design. J. R. Soc. Interface 9(68), 470–486 (2012)

12. Turberfield, A.J., Mitchell, J.C., Yurke, B., Mills Jr., A.P., Blakey, M.I., Simmel, F.C.: DNA fuel for free-running nanomachines. Phys. Rev. Lett. 90(11), 118102 (2003)
13. Seelig, G., Yurke, B., Winfree, E.: Catalyzed relaxation of a metastable DNA fuel. J. Am. Chem. Soc. 128, 12211–12220 (2006)
14. Green, S.J., Bath, J., Turberfield, A.J.: Coordinated chemomechanical cycles: A mechanism for autonomous molecular motion. Phys. Rev. Lett. 101, 238101 (2008)
15. Genot, A.J., Zhang, D.Y., Bath, J., Turberfield, A.J.: Remote toehold: A mechanism for flexible control of DNA hybridization kinetics. J. Am. Chem. Soc. 133(7), 2177–2182 (2011)
16. Lakin, M.R., Youssef, S., Polo, F., Emmott, S., Phillips, A.: Visual DSD: a design and analysis tool for DNA strand displacement systems. Bioinformatics 27(22), 3211–3213 (2011)
17. Douglas, S.M., Marblestone, A.H., Teerapittayanon, S., Vazquez, A., Church, G.M., Shih, W.M.: Rapid prototyping of three-dimensional DNA-origami shapes with caDNAno. Nucleic Acids Res. 37, 5001–5006 (2009)
18. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Inform Process Lett. 80, 25–31 (2001)
19. Phillips, A., Cardelli, L.: Efficient, correct simulation of biological processes in the stochastic pi-calculus. In: Calder, M., Gilmore, S. (eds.) CMSB 2007. LNCS (LNBI), vol. 4695, pp. 184–199. Springer, Heidelberg (2007)
20. Li, Y., Breaker, R.R.: Deoxyribozymes: new players in the ancient game of biocatalysis. Curr. Opin. Struct. Biol. 9, 315–323 (1999)
21. Lund, K., Manzo, A.J., Dabby, N., Michelotti, N., Johnson-Buck, A., Nangreave, J., Taylor, S., Pei, R., Stojanovic, M.N., Walter, N.G., Winfree, E., Yan, H.: Molecular robots guided by prescriptive landscapes. Nature 465, 206–210 (2010)
22. Pei, R., Taylor, S.K., Stefanovic, D., Rudchenko, S., Mitchell, T.E., Stojanovic, M.N.: Behavior of polycatalytic assemblies in a substrate-displaying matrix. J. Am. Chem. Soc. 128(39), 12693–12699 (2006)
23. Olah, M.J.: Multivalent Random Walkers: A computational model of superdiffusion at the nanoscale. PhD thesis, University of New Mexico (2012)
24. Olah, M.J., Stefanovic, D.: Superdiffusive transport by multivalent molecular walkers moving under load. Phys. Rev. E 87, 62713 (2013)
25. Semenov, O.: Abstract Models of Molecular Walkers. PhD thesis, University of New Mexico (2013)
26. Semenov, O., Mohr, D., Stefanovic, D.: First passage properties of molecular spiders. Phys. Rev. E 88, 012724 (2013)
27. Semenov, O., Olah, M.J., Stefanovic, D.: Cooperative linear cargo transport with molecular spiders. Natural Computing 12(2), 259–276 (2013)
28. Stefanovic, D.: Maze exploration with molecular-scale walkers. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) TPNC 2012. LNCS, vol. 7505, pp. 216–226. Springer, Heidelberg (2012)

On Decidability and Closure Properties of Language Classes with Respect to Bio-operations*

Oscar H. Ibarra

Department of Computer Science,
University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract. We present general results that are useful in showing closure and decidable properties of large classes of languages with respect to biologically-inspired operations. We use these results to prove new decidability results and closure properties of some classes of languages under hairpin-inversion, pseudo-inversion, and other bio-operations. We also provide a new approach for proving the undecidability of problems involving bio-operations for which the usual method of reduction to the undecidability of the Post Correspondence Problem (PCP) may not be easy to apply. Our closure and decidability results strengthen or generalize previous results.

Keywords: hairpin-inversion, pseudo-inversion, pushdown automaton, counters, reversal-bounded, decidable, undecidable, closure properties.

1 Introduction

The DNA literature has a large collection of papers that connect biological operations to formal languages and automata theory. Many studies have investigated bio-properties in terms of well-known notions in formal languages. In particular, researchers have looked at closure questions, decision problems, and other problems concerning formal languages under biological operations, see, e.g., [2,3,4,5,8,12,13,14,15].

Our work was motivated in part by a recent paper [1] that introduced a new bio-inspired operation called pseudo-inversion, which a variant of an important and well-studied operation called inversion. In a pseudo-inversion of a string, only the outermost parts of the string are reversed, while the middle part is not reversed. Inversion on the other hand, completely reverses the string. In looking at some of the results concerning pseudo-inversion in [1], we found that we are able to extend the findings in [1] to large classes of formal languages and to other bio-operations, like hairpin-inversion, that have been investigated in the literature (see, e,g., [2,3]).

* Supported in part by NSF Grants CCF-1143892 and CCF-1117708.

S. Murata and S. Kobayashi (Eds.): DNA 2014, LNCS 8727, pp. 148–160, 2014.

In this paper, we give general results that are useful in showing closure and decidable properties of classes of languages with respect to bio-operations. Formally, let let Σ be an alphabet (finite nonempty set of symbols), $L \subseteq \Sigma^*$, $k \geq 1$, $p = (p_1, \ldots, p_k)$ be a permutation of $(1, \ldots, k)$, and $c = (c_1, \ldots, c_k)$ where each c_i is R (denoting "reverse") or _ (denoting blank, i.e., "not reverse"). Let $F(x_1, \ldots, x_k)$ be a Presburger relation on strings $x_1, \ldots, x_k$ over Σ^* (e.g., linear relationships on the lengths of the strings, specification as what symbols occur in some positions of the strings, etc.) We will give a precise definition of F below. Define the language:

$$L(k, p, c, F) = \{x_{p_1}^{c_1} \cdots x_{p_k}^{c_k} \mid x_1 \cdots x_k \in L, F(x_1, \ldots, x_k)\}$$

An example of $L(k, p, c, F)$ is the following:

> $L(4, (4, 1, 2, 3), (R, \_, R, R), F) = \{x_4^R x_1 x_2^R x_3^R) \mid x_1 x_2 x_3 x_4 \in L, x_1$ begins and ends with the same symbol, the first symbol of x_2 is the same as the last symbol of $x_3, |x_1| < |x_4|, |x_1| + |x_3| > |x_2| + |x_4|\}$.

Clearly, the constraints on the x_i's can be specified by a Presburger relation F.

We say that a class of languages $\mathcal{L}$ is closed under (k, p, c, F)-inversion if for every $L \in \mathcal{L}$, $L(k, p, c, F)$ is also on $\mathcal{L}$. We exhibit large classes of languages $\mathcal{L}$ with decidable emptiness problem that are closed under (k, p, c, F)-inversion. An example is the class of finite-crossing two-way NFAs augmented with a finite number of reversal-bounded counters (precise definition to be given later).

Using closure under (k, p, c, F)-inversion, we are then able to prove decidable properties for $\mathcal{L}$. For example, we show that it is decidable, given a language accepted by a two-way DFA with one reversal-bounded counter, whether it is $L(k, p, c, F)$-inversion-free (precise definition to be given later).

The proofs of many undecidable properties concerning bio-operations use the undecidability of the Post Correspondence Problem (PCP). Here we show a technique for proving undecidability for which PCP is not applicable. Our construction is a reduction to the undecidability of the halting problem for two-counter machines.

2 Preliminaries

In a recent paper [1], the notions of *pseudo-inverse* of a string and of a language were introduced. Let Σ be an alphabet (of symbols). For a string $w \in \Sigma^*$, define $P(w) = \{v^R x u^R \mid u, x, v \in \Sigma^*, w = uxv$, and $vu \neq \varepsilon\}$ (The null string is denoted by ε.) The definition can be extended to languages in the natural way. So for a language L, $P(L)$ is the set of all pseudo-inverses of strings in L.

Let $d \geq 1$. The *d-iterated pseudo-inverse* of a language L is defined as follows: $P^1(L) = P(L)$, and for $d > 1$, $P^d(L) = P(P^{d-1}(L))$. The *iterated pseudo-inverse* of L, denoted by $P^*(L)$, is the union of all the $P^d(L)$ over all $d \geq 1$.

A language L is *pseudo-inversion-free* if there is no string in $P(L)$ that is a substring of some string in L, i.e., $\Sigma^* P(L) \Sigma^* \cap L = \varnothing$.

Another important bio-operation is *hairpin-inversion*, For $w \neq \varepsilon$, the hairpin-inverse of w is $H(w) = \{xuy^R u^R z \mid w = xuyu^R z, x, u, y, z \in \Sigma^*, u \neq \varepsilon\}$. If L is a language, $H(L)$ is the set of all hairpin-inversions of strings in L. For $d \geq 1$, *d-iterated hairpin-inversion* is defined in the obvious way: $H^1(L) = H(L)$, and for $d > 1$, $H^d(L) = H(H^{d-1}(L))$. Again, $H^*(L)$ is the union of all the $H^d(L)$ over all $d \geq 1$. L is *hairpin-inversion-free* if there is no string in $H(L)$ that is a substring of some string in L, i.e., $\Sigma^* H(L) \Sigma^* \cap L = \varnothing$.

In this paper, we generalize the notions above to (k, p, c, F)-*inversion.* Let L_1 and L_2 be two languages over alphabet Σ. We say that L_1 is (k, p, c, F)-*inversion-free* for L_2, if there is no string in $L_1(k, p, c, F)$ that is a substring of some string in L_2, i.e., $\Sigma^* L_1(k, p, c, F) \Sigma^* \cap L_2 = \varnothing$. In the special case when $L = L_1 = L_2$, we simply say L is (k, p, c, F)-*inversion-free.*

We will use the following notation throughout the paper: NPDA for nondeterministic pushdown automaton; DPDA for deterministic pushdown automaton; NCA for an NPDA that uses only one stack symbol in addition to the bottom of the stack, which is never altered; DCA for deterministic NCA; NFA for nondeterministic finite automaton; DFA for deterministic finite automaton; 2NFA, 2DFA, etc. are the two-way versions (whose input tape has left and right end markers). We refer the reader to [9] for the formal definitions of these devices.

A *counter* is an integer variable that can be incremented by 1, decremented by 1, left unchanged, and tested for zero. It starts at zero and cannot store negative values. Thus, a counter is a pushdown stack on unary alphabet, in addition to the bottom of the stack symbol which is never altered.

An automaton (NFA, NPDA, NCA, etc.) can be augmented with a finite number of counters, where the "move" of the machine also now depends on the status (zero or non-zero) of the counters, and the move can update the counters. It is well known that a DFA augmented with two counters is equivalent to a TM [16].

In this paper, we will restrict the augmented counter(s) to be reversal-bounded in the sense that each counter can only reverse (i.e., change mode from nondecreasing to nonincreasing and vice-versa) at most r times for some given r. In particular, when $r = 1$, the counter reverses only once, i.e., once it decrements, it can no longer increment. Note that a counter that makes r reversals can be simulated by $\lceil \frac{r+1}{2} \rceil$ 1-reversal counters.

We call a relation $F(x_1, \ldots, x_k)$ on k strings over an alphabet Σ a *Presburger relation* if the language $L_F = \{x_1 \# \cdots \# x_k \mid (x_1, \ldots, x_k) \text{ satisfies } F\}$ can be accepted (i.e., verified) by an NFA with reversal-bounded counters, where # is a symbol not in Σ.

However, there is one place in the paper (section 4), where we define F to be Presburger if L_F can be accepted by a 2DFA with one reversal-bounded counter.

We will use the following result:

Theorem 1. *[10] It is decidable, given an NPDA M augmented with reversal-bounded counters, whether $L(M) = \varnothing$ (resp., whether $L(M)$ is finite).*

3 Closure Results

We begin with the following theorem concerning NCAs with reversal-bounded counters. (Note that such a machine has one unrestricted counter in addition to the reversal-bounded counters.)

Theorem 2. *Let L be a language accepted by an NCA M with reversal-bounded counters. Let $L' = \{x_3x_2x_1 \mid x_1x_2x_3 \in L\}$. Then L' can also be accepted by an NCA M' with reversal-bounded counters.*

Proof. Assume that M has disjoint state set and input alphabet. Let c be the unrestricted counter of M. Suppose M has n reversal-bounded counters. Let 0, 1, # be new symbols. Define a language L'' consisting of strings of the form:

$w = q_3 1^{i_3} 0 1^{j_{31}} 0 \cdots 0 1^{j_{3n}} x_3 q_4 1^{i_4} 0 1^{j_{41}} 0 \cdots 0 1^{j_{4n}} \# q_2 1^{i_2} 0 1^{j_{21}} 0 \cdots 0 1^{j_{2n}} x_2 q_3 1^{i_3} 0 1^{j_{31}} 0$
$\cdots 0 1^{j_{3n}} \# q_1 1^{i_1} 0 1^{j_{11}} 0 \cdots 0 1^{j_{1n}} x_1 q_2 1^{i_2} 0 1^{j_{21}} 0 \cdots 0 1^{j_{2n}}$

where q_1 is the initial state of M, q_4 is an accepting state of M, the i's and j's are nonnegative integers with $i_1 = j_{11} = \cdots = j_{1n} = 0$, and $x_1x_2x_3 \in L$.

L'' can be accepted by an NCA M'' with reversal-bounded counters (using many more reversal-bounded counters than M). M'', when given input w, operates in three phases:

Phase 1: M'' reads $q_3 1^{i_3} 0 1^{j_{31}} 0 \cdots 0 1^{j_{3n}}$ and remembers q_3 in the finite control and stores i_3 in its unrestricted counter c and $i_3, j_{31}, \ldots, j_{3n}$ in two sets of $n+1$ reversal-bounded counters. Then M'' simulates M on input segment x_3 starting in state q_3 using c (which has value i_3) and one set of n reversal-bounded counters which contain $j_{31}, \ldots, j_{3n}$. At the end of the simulation, it checks that the state is q_4 and the values of the unrestricted counter c and reversal-bounded counters are $i_4, j_{41}, \ldots, j_{4n}$, respectively which are represented on the input after x_3. Note that $i_3, j_{31}, \ldots, j_{3n}$ have been stored in another set of counters.

Phase 2: As in phase 1, M'' reads $q_2 1^{i_2} 0 1^{j_{21}} 0 \cdots 0 1^{j_{2n}}$ and simulates M on x_2 and carries out the same process as in Phase 1, but it also checks at the end of the phase that the state and values of c and reversal-bounded counters are identical to the values stored in the second set of $n+1$ counters in Phase 1, i.e., the configuration (state and counter values) at the end of Phase 2 is identical to the configuration that Phase 1 started with.

Phase 3: As in Phase 2, M'' reads $q_1 1^{i_1} 0 1^{j_{11}} 0 \cdots 0 1^{j_{1n}}$ and simulates M on x_1 and carries out the same process as Phase 2 and at the end of the phase checks that the configuration at the end of Phase 3 is identical to the configuration that Phase 2 started with.

It is easily verified that M'' accepts L''. Finally, we construct from M'' an NCA with reversal-bounded counters M' that accept L' (since languages accepted by these machines are clearly closed under homomorphism). □

In the special case when M is an NFA, we have:

Corollary 1. *Let L be a language accepted by an NFA M. Let $L' = \{x_3x_2x_1 \mid x_1x_2x_3 \in L\}$. Then L' can also be accepted by an NFA M' whose size is polynomial in the size of M.*

Proof. We need only note that in the proof of Theorem 2, the language L'' will now consist of strings of the form $w = q_3x_3q_4\#q_2x_2q_3\#q_1x_1q_2$ (i.e., no counter values to be stored). Hence the size of M'' is polynomial in the size of M. Then the NFA M' to accept L' (which is simply erasing the q's and the $\#$'s from the input string) would still be polynomial in the size of M. □

The reason why the construction in Theorem 2 is a bit complicated is because we needed to construct an NCA M' with reversal-bounded counters for L' which has *one-way* input, since we need the fact that the emptiness problem for NCAs with reversal-bounded counters is decidable. It is known that the emptiness problem for 2DCAs (i.e., the machines have only an unrestricted counter and no reversal-bounded counters) is undecidable, even when the input head makes only 3 turns on the input tape [10]. The construction of the intermediate machine M'' above works, because in executing the three phases, we are able to "temporarily" store the value of the unrestricted counter c in a reversal-bounded counter in going from phase to phase in simulating the NCA M with reversal-bounded counters, since in the simulation, the input to M'' is not given in the same order as the input to M. This idea does not work when the unrestricted counter is replaced by a pushdown stacks, since we would need another stack to temporarily store the contents of the stack and the resulting machine would then have two stacks (the emptiness would be undecidable).

We observe that Theorem 2 and Corollary 1 would still hold for the case when some of x_1, x_2, x_3 are reversed. For example, $L' = \{x_3x_2^Rx_1 \mid x_1x_2x_3 \in L\}$ can be accepted by a NCA with reversal-bounded counters. In this case, the input segment:

$$q_2 1^{i_2} 0 1^{j_{21}} 0 \cdots 0 1^{j_{2n}} x_2 q_3 1^{i_3} 0 1^{j_{31}} 0 \cdots 0 1^{j_{3n}}$$

will now be displayed in reverse, i.e., the segment will now be:

$$1^{j_{3n}} 0 \cdots 0 1^{j_{31}} 0 1^{i_3} q_3 x_2^R 1^{j_{2n}} 0 \cdots 0 1^{j_{21}} 0 1^{i_2} q_2$$

Then in Phase 2, the simulation of M on input segment x_2 will be carried out by M'' in reverse.

Using the ideas above, we can prove a general result. Let $L \subseteq \Sigma^*$, $k \geq 1$, $p = (p_1, \ldots, p_k)$ a permutation of $(1, \ldots, k)$, and $c = (c_1, \ldots, c_k)$ where each c_i is R (denoting "reverse") or _ (denoting blank, i.e., "not reverse"). Let $F(x_1, \ldots, x_k)$ be a Presburger relation on strings $x_1, \ldots, x_k$ over Σ^*. Define the language:

$$L(k, p, c, F) = \{x_{p_1}^{c_1} \cdots x_{p_k}^{c_k} \mid x_1 \cdots x_k \in L, F(x_1, \ldots, x_k)\}$$

We say that a class of languages $\mathcal{L}$ is closed under (k, p, c, F)-inversion if for every $L \in \mathcal{L}$, $L(k, p, c, F)$ is also on $\mathcal{L}$. An example of $L(k, p, c, F)$ is the following:

$$L(3,(3,2,1),(R,\_,R)) = \{x_3^R x_2 x_1^R \mid x_1x_2x_3 \in L, |x_1|+|x_3|>0\}.$$

Clearly, the constraints on the x_i's can be specified by a Presburger relation F that can be verified by an NFA with reversal-bounded counters. Note that $L(3,(3,2,1),(R,\_,R))$ is the pseudo-inversion of L.

Theorem 3. *The class of languages accepted by NCAs with reversal-bounded counters is closed under (k,p,c,F)-inversion.*

Proof. Let M be an NCA with reversal-bounded counters accepting L. We construct an NCA M' with reversal-bounded counters accepting $L(k,p,c,F)$ following the ideas in the proof of Theorem 2 and discussion above and the fact that the Presburger relation F can be verified by an NFA with reversal-bounded counters (by definition), which can run in parallel with M'' (see proof of Theorem 2). □

In the special case when M is an NFA and F can be verified by an NFA (i.e., without reversal-bounded counters):

Corollary 2. *If L is accepted by an NFA M and F can be verified by an NFA M_F, then $L(k,p,c,F)$ can also be accepted by an NFA M' whose size is polynomial in the k and the sizes of M and M_F.*

The above corollary generalizes the results in [1] and [2] which showed that regular sets are closed under pseudo-inversion and hairpin-inversion, respectively.

We think that Theorem 3 and Corollary 2 can serve as powerful tools for showing closure properties and decidability of formal languages with respect to bio-operations. We illustrate some applications on positive closure properties here. In the next section, we will prove some decidable properties.

From Theorem 3 and the example above, we have:

Corollary 3. *The class of languages accepted by NCAs with reversal-bounded counters is closed under pseudo-inversion and d-iterated-pseudo-inversion.*

Corollary 4. *Let $L \subseteq \Sigma^*$ be a language accepted by an NCA with reversal-bounded counters, and $d \geq 1$. Then $H^d(L)$ is also accepted by a NCA with reversal-bounded counters.*

Proof. It was shown in [2] that the hairpin-inversion definition of a string w is equivalent to $H(w) = \{xay^Raz \mid w = xayaz, a \in \Sigma, x,y,z \in \Sigma^*\}$. We then apply Theorem 3 with $k=5, p=(1,2,3,4,5), c=(\_,\_,R,\_,\_)$, and Presburger constraint $|x_2|=|x_4|=1$ and x_1 and x_2 contain the same symbol. □

There are other interesting and well-known bio-operations such as *synchronized insertion, synchronized deletion*, and *synchronized bi-deletion* (see, e.g., [2,3]). Theorem 3 and the easily proved closure of languages accepted by NPDAs (resp., NCAs, NFAs) with reversal-bounded counters under intersection with regular sets, homomorphism, inverse homomorphism and reversal, can be used to improve or give shorter proofs of closure properties that have appeared in the literature (see, e.g., [2]).

Now we look at languages accepted by NPDAs (resp., NPDAs with reversal-bounded counters). Let $0, 1, a, b, \#_1, \#_2, \#_3$ be distinct symbols, and consider the language $L = \{\#_1 x_1 \#_2 x_2 \#_3 \mid x_1 \in (0+1)^+, x_2 \in (a+b)^+, x_1$ when $0, 1$ are mapped to a, b, respectively, is the reverse of $x_2\}$. Clearly, L can be accepted by a DPDA. Suppose the pseudo-inversion of L, $P(L)$, can be accepted by an NPDA (resp., NPDA with reversal-bounded counters). Then $L' = P(L) \cap \#_2(a+b)^+\#_3(0+1)^+\#_1$ can also be accepted by an NPDA (resp., NPDA with reversal-bounded counters), since these families are closed under intersection with regular sets. But $L' = \{\#_2 x \#_3 y \#_1 \mid y$ when $0, 1$ are mapped to a, b is equal to x. Clearly, L' is not context-free (i.e., cannot be accepted by an NPDA). It is also unlikely that L' can be accepted by an NPDA with reversal-bounded counters. We conclude that the class of languages accepted by NPDAs is not closed under pseudo-inversion, and it is very likely that the class accepted by NPDAs with reversal-bounded counters is also not closed under pseudo-inversion. The observation that the class of languages accepted by NPDAs is not closed under pseudo-inversion was already made in [1], but the NPDA language L used in [1] is such that $P(L)$ can be accepted by a NPDA with reversal-bounded counters.

There is another powerful class of machines with decidable emptiness problem that one can use for showing closure properties. Consider a 2NFA with reversal-bounded counters (recall that '2' means the input head is two-way, and the input has left and right end markers). It is known that the emptiness problem for such machines is undecidable, even when there are only two reversal-bounded counters [10]. But suppose it is *finite-crossing* in the sense that there is some given integer c such that the input head crosses the boundary between any two adjacent input symbols at most c times. The following was shown in [6]:

Theorem 4. *It is decidable, given a finite-crossing 2NFA M augmented with reversal-bounded counters, whether $L(M) = \varnothing$ (resp., whether $L(M)$ is finite).*

One can also show the following proposition using the results in [10,6]:

Proposition 1. *The class of languages accepted by finite-crossing 2NFAs with reversal-bounded counters is closed under union, intersection, reversal, and homomorphism.*

Languages accepted by finite-crossing 2NFAs with reversal-bounded counters can easily be shown to be closed under bio-operations like pseudo-inversion, hairpin-inversion, etc. This is because the input head is allowed to have two-way capability. In fact, we can prove a result similar to Theorem 3:

Theorem 5. *The class of languages accepted by finite-crossing 2NFAs with reversal-bounded counters is closed under (k, p, c, F)-inversion.*

Proof. By definition, Let M be a finite-crossing 2NFA accepting L. $L(k, p, c, F) = \{x_{p_1}^{c_1} \cdots x_{p_k}^{c_k} \mid x_1 \cdots x_k \in L, F(x_1, \ldots, x_k)\}$. Let $\#$ be a new symbol.

Let $L' = \{x_{p_1}^{c_1}\# \cdots \#x_{p_k}^{c_k} \mid x_1 \cdots x_k \in L, F(x_1, \ldots, x_k)\}$. Thus, L' is $L(k, p, c, F)$ with markers.

We show that L' can also be accepted by a finite-crossing 2NFA M' with reversal-bounded counters. M', when given input w', simulates the computation the computation of M on input $w = x_1 \cdots x_k$. Even though the x_i's in the input w' to M' are not displayed in the same way, M' is able to simulate M easily because the boundaries of the input segments are marked by #'s. M' accepts if M accepts and $(x_1, \ldots, x_k)$ satisfies the relation F. Then from Proposition 1, M' can be converted to a finite-crossing 2NFA with reversal bounded counters to accept $L(k, p, c, F)$ (which is just a homomorphic image of L', which erases the #'s). □

Finally, we note that since the emptiness problem for finite-crossing two-way NPDAs and two-way NCAs (even when the machines only make three sweeps of the input) is undecidable, finite-crossing for these devices would not be useful.

4 Decidable Results

Let L_1 and L_2 be two languages over alphabet Σ. Recall that L_1 is (k, p, c, F)-inversion-free for L_2 if $\Sigma^* L_1(k, p, c, F)\Sigma^* \cap L_2 = \varnothing$. In the special case when $L = L_1 = L_2$, we simply say L is $(k, p.c, F)$-inversion-free.

Theorem 6. *It is decidable, given $L_1, L_2 \subseteq \Sigma^*$, whether L_1 is (k, p, c, F)-inversion-free for L_2 for the following cases:*

1. *L_1 is accepted by an NCA (hence, also NFA) with reversal-bounded counters and L_2 is accepted by an NFA with reversal-bounded counters.*
2. *L_1 is accepted by an NFA with reversal-bounded counters and L_2 is accepted by an NPDA with reversal-bounded counters.*
3. *L_1 and L_2 are accepted by finite-crossing 2NFAs with reversal-bounded counters.*

Proof. Consider part 1. Let L_1 be accepted by an NCA M_1 with reversal-bounded counters, and L_2 be accepted by an NFA M_2 with reversal-bounded counters. (Note that M_1 has one unrestricted counter in addition to reversal-bounded counters.) From Theorem 3, we construct from M_1, an NCA M_1' with reversal-bounded counters to accept $L_1(k, p, c, F)$. Then we construct from M_1' another NCA M_1'' with reversal-bounded counters to accept $\Sigma^* L_1(k, p, c, F)\Sigma^*$. Finally, we construct an NCA M with reversal-bounded counters to accept $L(M_1'') \cap L(M_2)$. (M simulates M_1'' and M_2 in parallel.) Clearly, L_1 is (k, p, c, F)-inversion-free for L_2 if and only if $L(M) = \varnothing$. The result follows, since emptiness of languages accepted by NPDAs (hence also NCAs) with reversal-bounded counters is decidable by Theorem 1.

Now consider part 2. As in part 1, $\Sigma^* L_1(k, p, c, F)\Sigma^*$ can be accepted by an NFA M_1'' with reversal-bounded counters (similar construction as in part 1). Then we can construct an NPDA M with reversal-bounded counters to accept $L(M_1'') \cap L(M_2)$. Again, the result follows since emptiness for NPDAs with reversal-bounded counters is decidable (Theorem 1).

For part 3, let L_1 and L_2 be accepted by finite-crossing 2NFAs with reversal-bounded counters M_1 and M_2, respectively. From Theorem 5, we construct from M_1, a finite-crossing 2NFA M_1' with reversal-bounded counters to accept $L_1(k, p, c, F)$. Let # be a new symbol. Clearly, we can construct from M_1', a finite-crossing 2NFA M_2 with reversal-bounded counters to accept $\Sigma^* \# L_1(k, p, c, F) \# \Sigma^*$. Then by Proposition 1, $\Sigma^* L_1(k, p, c, F) \Sigma^*$ can also be accepted by a finite-crossing 2NFA M_1'' with reversal-bounded counters (i.e., the #'s are deleted). Finally, from Proposition 1 again, we can construct from M_1'' and M_2 a finite-crossing 2NFA M with reversal-bounded counters to accept $\Sigma^* L_1(k, p, c, F) \Sigma^* \cap L_2$. The result follows, since emptiness of languages accepted by finite-crossing 2NFAs with reversal-bounded counters is decidable by Theorem 4. □

For the special cases of pseudo-inversion and hairpin-inversion, we get the following corollary from Theorem 6:

Corollary 5. *It is decidable, given $L_1, L_2 \subseteq \Sigma^*$, whether L_1 is pseudo-inversion-free (resp., hairpin-inversion-free) for L_2 for the following cases:*

1. *L_1 is accepted by an NCA (hence, also NFA) with reversal-bounded counters and L_2 is accepted by an NFA with reversal-bounded counters.*
2. *L_1 is accepted by an NFA with reversal-bounded counters and L_2 is accepted by an NPDA with reversal-bounded counters.*
3. *L_1 and L_2 are accepted by finite-crossing 2NFAs with reversal-bounded counters.*

Part 1 of Corollary 5 does not hold if L_2 is also accepted by an NCA with reversal-bounded counters. In fact, as we shall see in the next section, it is already undecidable, given a language L accepted by an NCA, whether it is pseudo-inversion-free.

Theorem 7. *Let $c, r \geq 0$ be fixed. The problem of deciding, given an NFA with c counters, each of which makes at most r-reversals, whether $L(M)$ is pseudo-inverse-free is in NLOGSPACE, hence also in PTIME.*

Proof. It is known that for fixed c, r, the emptiness problem for NFAs with c r-reversal counters is in NLOGSPACE [6]. The result follows from the constructions in the proofs of Theorems 3 and 6, i.e., M in proof of Theorem 6 will have a fixed number of counters depending only on r and c, and the reversal bound for each of the counters will also only depend on r and c. □

Theorem 7 improves a result in [1] which showed that it decidable in PTIME whether a language accepted by an NFA is pseudo-inversion-free. Theorem 7 also holds when "pseudo-inverse-free" is replaced by "hairpin-inverse-free".

Remark. Suppose L_1 is not pseudo-inversion-free for L_2. Can we decide if $\Sigma^* L_1(k, p, c, F) \Sigma^* \cap L_2$ is finite? The answer is "yes" for the three cases in Theorem 6 and Corollary 5, because it is decidable if an NPDA with reversal-bounded counters is finite (Theorem 1).

As we mentioned earlier, 2DFAs with two reversal-bounded counters have an undecidable emptiness problem. However, the emptiness problem for 2DFAs with one reversal-bounded counter is decidable [11]. Note that a 2DFA with one reversal-bounded counter is quite powerful. For example, it can accept the language $L = \{a^i b^j \mid i, j \geq 1, i \text{ is divisible by } j\}$.

Here we assume that the Presburger relations can be verified by 2DFAs with one reversal-bounded counter.

Theorem 8. *It is decidable, given $L_1, L_2 \subseteq \Sigma^*$ that are accepted by 2DFAs with one reversal-bounded counter, whether L_1 is (k, p, c, F)-inversion-free for L_2.*

Proof. Given M_1 accepting L_1 and k, p, c, F, let M_F be a 2DFA with one reversal-bounded counter that verifies F. Let $L_1' = \{\#x_{p_1}^{c_1}\# \cdots \#x_{p_k}^{c_k}\# \mid x_1 \cdots x_k \in L_1, F(x_1, \ldots, x_k)\}$, where $\#$ is a new symbol not in Σ. Thus L_1' is $L_1(k, p, c, F)$ with markers. We construct a 2DFA M_1' with one reversal-bounded counter to accept L_1'. Then from M_1', we construct another 2DFA M_1'' with one reversal-bounded counter to accept $L_1'' = \Sigma^* L_1' \Sigma^*$. Let M_2 be a 2DFA with one reversal-bounded counter that accepts L_2. Let $L_2' = \{y_0\#y_1\# \cdots \#y_k\#y_{k+1} \mid y_0 y_1 \cdots y_k y_{k+1} \in L_2\}$. Thus L_2' is L_2 with markers. Clearly, we can construct from M_2 a 2DFA M_2' with one reversal-bounded counter to accept L_2'. Finally, we construct from M_1'' and M_2' a 2DFA M with one reversal-bounded counter to accept $L(M_1'') \cap L(M_2')$. The result follows since $\Sigma^* L_1(k, p, c, F)\Sigma^* \cap L_2 = \varnothing$ if and only if $\Sigma^* L_1'' \Sigma^* \cap L_2' = \varnothing$, and the emptiness problem for 2DFAs with one reversal-bounded counters is decidable. □

5 Undecidable Results

In [1], it was shown that it is undecidable if a context-free grammar (or, equivalently, an NPDA) is pseudo-inversion-free. The proof involved the use of the undecidability of the Post Correspondence Problem (PCP), which does not seem easy to apply to NCAs.

Here, we show, by a different construction, that the problem is still undecidable for NCAs (i,e., the pushdown stack is restricted to use only one stack symbol, in addition to the bottom of the stack, which is never altered).

Theorem 9. *It is undecidable to determine, given an NCA, whether it is pseudo-inversion-free.*

Proof. The proof uses the undecidability of the halting problem for 2-counter machines. A close look at the proof in [16] of the undecidability of the halting problem for 2-counter machines, where initially one counter has value d_1 and the other counter is zero, reveals that the counters behave in a regular pattern. The 2-counter machine operates in phases in the following way. Let c_1 and c_2 be its counters. The machine's operation can be divided into phases, where each phase starts with one of the counters equal to some positive integer d_i and the other counter equal to 0. During the phase, the positive counter decreases, while the other counter increases. The phase ends with the first counter having value

0 and the other counter having value d_{i+1}. Then in the next phase the modes of the counters are interchanged. Thus, a sequence of configurations corresponding to the phases will be of the form:

$$(q_1, d_1, 0), (q_2, 0, d_2), (q_3, d_3, 0), (q_4, 0, d_4), (q_5, d_5, 0), (q_6, 0, d_6), \ldots$$

where the q_i's are states, with q_1 the initial state, and $d_1, d_2, d_3, \ldots$ are positive integers. Note that the second component of the configuration refers to the value of c_1, while the third component refers to the value of c_2. We assume, w.l.o.g., that $d_1 = 1$.

Let T be a 2-counter machine. We assume that if T halts, it does so in a unique state q_h. Let T's state set be Q, and 1 be a new symbol.

In what follows, α is any sequence of the form $I_1 I_2 \cdots I_{2m}$ (thus we assume that the length is even), where $I_i = 1^k q$ for some $k \geq 1$ and $q \in Q$, represents a possible configuration of T at the beginning of phase i, where q is the state and k is the value of counter c_1 (resp., c_2) if i is odd (resp., even).

Define L_{odd} to be the set of all strings α such that

1. $\alpha = \#I_1\#I_2 \cdots \#I_{2m}\$$;
2. $m \geq 1$;
3. $I_1 = 1^{d_1} q_1$, where $d_1 = 1$ and q_1 is the initial state, which we assume is never re-entered during the computation;
4. $I_{2m} = 1^v q_h$ for some positive integer v, where q_h is the only halting state;
5. for odd j, $1 \leq j \leq 2m - 1$, $I_j \Rightarrow I_{j+1}$, i.e., if T begins in configuration I_j, then after one phase, T is in configuration I_{j+1};

Similarly, define L_{even} analogously except that the condition "$I_j \Rightarrow I_{j+1}$" now applies to *even* values of j, $2 \leq j \leq 2m - 2$.

Now, let $L = L_{odd} \cup L_{even}^R$ (note that R denotes reverse). Let Σ be the alphabet over which L is defined.

We can construct an NCA M accepting $L \subseteq \Sigma^*$ as follows. Given input x, M executes (1) or (2) below:

(1) M checks that x is in L_{odd} deterministically by simulating the 2-counter machine T as follows: M reads $x = \#I_1\#I_2 \cdots \#I_{2m}\$ = \#1^{d_1}q_1\#1^{d_2}q_2 \cdots \#1^{d_{2m}}$ $q_{2m}\$$ and verifies that $d_1 = 1$, q_1 is the initial state, q_{2m} is the halting state q_h, and for odd j, $1 \leq j \leq 2m - 1$, $I_j \Rightarrow I_{j+1}$. To check that $I_j \Rightarrow I_{j+1}$, M reads the segment $1^{d_j} q_j$ and stores 1^{d_j} in its counter (call it c) and remembers the state q_j in its finite control. This represents the configuration of T when one of its two counters, say c_1, has value d_j, the other counter, say c_2, has value 0, and its state is q_j. Then, starting in state q_j, M simulates the computation of T by decrementing c (which is simulating counter c_1 of T) and reading the input segment $1^{d_{j+1}}$ until c becomes zero and at which time, the input head of M should be on q_{j+1}. Thus, the process has just verified that counter c_2 of T has value $1^{d_{J+1}}$, counter c_1 has value 0, and the state is q_{j+1}.

(2) In a similar way, M checks that x^R is in L^R_{even}, i.e., for even j, $2 \leq j \leq 2m-2$, $I_j \Rightarrow I_{j+1}$. But M does the checking in reverse, since I_{j+1} is read before I_j; so M does the checking in a nondeterministic way.

One can verify that M is not pseudo-inversion-free if and only if T halts. We omit the details. □

Finally, we note that using a construction similar to that in the proof of Theorem 9, we can show that it is undecidable to determine, given an NCA, whether it is hairpin-inversion-free.

6 Conclusion

We have presented general results that are useful in showing closure and decidable properties of large classes of languages with respect to biologically-inspired operations. We have also provided a technique for proving the undecidability of problems involving bio-operations for which the PCP is not applicable.

It was recently shown in [7] that the emptiness problem for NPDAs augmented with reversal-bounded counters (where the number of counters and the bound r on reversals are not fixed, but r is assumed to be given in unary) is NP-complete, even when the counters can be compared against and incremented/decremented by constants that are given in binary. This result can be used to show that some of the decision problems we discussed in Section 4 are NP-complete.

References

1. Cho, D.-J., Han, Y.-S., Kang, S.-D., Kim, H., Ko, S.-K., Salomaa, K.: Pseudo-inversion on formal languages. In: Ibarra, O.H., Kari, L., Kopecki, S. (eds.) UCNC 2014. LNCS, vol. 8553, pp. 93–104. Springer, Heidelberg (2014)
2. Daley, M., Ibarra, O.H., Kari, L.: Closure and decidability properties of some languages classes with respect to ciliate bio-operations. Theoret. Comput. Sci. 306(1-3), 19–38 (2003)
3. Daley, M., Kari, L., McQuillan, I.: Families of languages defined by ciliate bio-operations. Theoret. Comput. Sci. 320(1), 51–69 (2004)
4. Deaton, R., Garzon, M., Murphy, R.C., Rose, J.A., Franceschetti, D.R., Stevens Jr., S.E.: Genetic search of reliable encodings for DNA-based computation. In: Proc. of 1st Conf. on Genetic Programming, pp. 9–15 (1996)
5. Garzon, M., Deaton, R., Nino, L.F., Stevens, E., Wittner, M.: Encoding genomes for DNA computing. In: Genetic Programming 1998: Proc. of 3rd Annual Conf., pp. 684–690 (1998)
6. Gurari, E., Ibarra, O.H.: The complexity of decision problems for finite-turn multicounter machines. J. Comput. Syst. Sci. 22, 220–229 (1981)
7. Hague, M., Lin, A.W.: Model checking recursive programs with numeric data types. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 743–759. Springer, Heidelberg (2011)
8. Hussini, S., Kari, L., Konstantinidis, S.: Coding properties of DNA languages
9. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata, Languages and Computation. Addison-Wesley (1978)

10. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. of the ACM 25(1), 116–133 (1978)
11. Ibarra, O.H., Jiang, T., Tran, N.Q., Wang, H.: New decidability results concerning two-way counter machines. SIAM J. Comput. 24(1), 123–137 (1995)
12. Jonoska, N., Kari, L., Mahalingam, K.: Involution solid and join codes. Fundamenta Informaticae 86(1,2), 127–142 (2008)
13. Jonoska, N., Mahalingam, K., Chen, J.: Involution codes: with application to DNA coded languages. Natural Computing 4(2), 141–162 (2005)
14. Kari, L., Lossevsa, E., Konstantinidis, S., Sosik, P., Thierrin, G.: A formal language analysis of DNA hairpin structures. Fundamenta Informaticae 71(4), 453–475 (2006)
15. Kari, L., Mahalingam, K.: DNA codes and their properties. In: Mao, C., Yokomori, T. (eds.) DNA12. LNCS, vol. 4287, pp. 127–142. Springer, Heidelberg (2006)
16. Minsky, M.: Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines. Ann. of Math. (74), 437–455 (1961)

Author Index