28th International Conference on DNA Computing and Molecular Programming

**DNA 28, August 8–12, 2022, University of New Mexico,
Albuquerque, New Mexico, USA**

Edited by

Thomas E. Ouldridge
Shelley F. J. Wickham

LIPIcs – Vol. 238 – DNA 28 www.dagstuhl.de/lipics

Editors

Thomas E. Ouldridge
Imperial College London, UK
t.ouldridge@imperial.ac.uk

Shelley F. J. Wickham
University of Sydney, Australia
shelley.wickham@sydney.edu.au

ACM Classification 2012
Theory of computation → Models of computation; Applied computing → Molecular structural biology; Applied computing → Biological networks; Information systems → Information storage systems

ISBN 978-3-95977-253-2

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-253-2.

Publication date
August, 2022

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.DNA.2022.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

In memory of Ned Seeman, for his many contributions to our research community.

Contents

Preface
Thomas E. Ouldridge and Shelley F. J. Wickham 0:ix

Organization

Steering Committee
. . . . 0:xi

Program Committee
. . . . 0:xii

Additional Reviewers for Tracks A and B
. . . . 0:xiii

Organizing Committee for DNA 28
. . . . 0:xiv

Sponsors
. . . . 0:xv

Papers

Fast and Robust Strand Displacement Cascades via Systematic Design Strategies
Tiernan Kennedy, Cadence Pearce[1]*, and Chris Thachuk* 1:1–1:17

Universal Shape Replication via Self-Assembly with Signal-Passing Tiles (Extended Abstract)
Andrew Alseth, Daniel Hader, and Matthew J. Patitz 2:1–2:24

A Coupled Reconfiguration Mechanism for Single-Stranded DNA Strand Displacement Systems
Hope Amber Johnson and Anne Condon 3:1–3:19

Exploring Material Design Space with a Deep-Learning Guided Genetic Algorithm
Kuan-Lin Chen and Rebecca Schulman 4:1–4:14

Modelling and Optimisation of a DNA Stack Nano-Device Using Probabilistic Model Checking
Bowen Li, Neil Mackenzie, Ben Shirt-Ediss, Natalio Krasnogor, and Paolo Zuliani 5:1–5:22

On Turedo Hierarchies and Intrinsic Universality
Samuel Nalin and Guillaume Theyssier 6:1–6:18

Computing Real Numbers with Large-Population Protocols Having a Continuum of Equilibria
Xiang Huang and Rachel N. Huls 7:1–7:22

The Structural Power of Reconfigurable Circuits in the Amoebot Model
Andreas Padalkin, Christian Scheideler, and Daniel Warner 8:1–8:22

Fault-Tolerant Shape Formation in the Amoebot Model
Irina Kostitsyna, Christian Scheideler, and Daniel Warner 9:1–9:22

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at DNA 28: the 28th International Conference on DNA Computing and Molecular Programming. The conference was held during August 8–12, 2022, at the University of New Mexico, Albuquerque, New Mexico, USA, and was organized under the auspices of the International Society for Nanoscale Science, Computation, and Engineering (ISNSCE). The DNA conference series aims to draw together researchers from the fields of mathematics, computer science, physics, chemistry, biology, and nanotechnology to address the analysis, design, and synthesis of information-based molecular systems.

Papers and presentations were 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 folding and self-assembly of nanostructures; analysis and theoretical models of laboratory techniques; molecular motors and molecular robotics; information storage; studies of fault-tolerance and error correction; software tools for analysis, simulation,and design; synthetic biology and in vitro evolution; and applications in engineering, physics, chemistry, biology, and medicine.

Authors who wished to orally present their work 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 or theoretical results who plan to submit to a journal rather than publish in the conference proceedings. We received 38 submissions for oral presentations: 14 submissions to Track A and 24 submissions to Track B. Each submission was reviewed by at least four reviewers. The Program Committee accepted 9 papers for Track A (64%) and 14 papers for Track B (58%). We also received 50 submissions for Track C (poster), of which four were selected as additional oral presentations by the Program Committee. This volume contains the papers accepted for Track A.

We express our sincere appreciation to our invited speakers: Andrew Ellington, Jessica Flack, Elisa Franco and Matthew Patitz. We thank all of the authors who contributed papers to these proceedings, and those who presented papers and posters during the conference. Last, but by no means least, the editors are especially grateful to the members of the Program Committee and the additional invited reviewers for their hard work in reviewing the papers on a tight deadline and for providing insightful and constructive comments to the authors.

Thomas Ouldridge
Shelley Wickham

August 2022

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Steering Committee

Anne Condon (Chair)	University of British Columbia
Masami Hagiya	University of Tokyo
Natasha Jonoska	University of Southern Florida
Chengde Mao	Purdue University
Satoshi Murata	Tohoku University
John H. Reif	Duke University
Grzegorz Rozenberg	University of Leiden
Rebecca Schulman	Johns Hopkins University
Friedrich Simmel	Technical University Munich
David Soloveichik	The University of Texas at Austin
Andrew J. Turberfield	Physics, University of Oxford
Erik Winfree	California Institute of Technology
Damien Woods	Maynooth University
Hao Yan	Arizona State University

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Shelley Wickham (co-chair)	University of Sydney
Thomas Ouldridge (co-chair)	Imperial College London
Pekka Orponen	Aalto University
Shinnosuke Seki	The University of Electro-Communications
Trent Rogers	Maynooth University
Sung Ha Park	Sungkyunkwan University
Shalin Shah	Bloomberg
Dongsheng Liu	Tsinghua University
Abeer Eshra	Maynooth University
Damien Woods	Maynooth University
Rakesh Mukherjee	Imperial College London
Ho-Lin Chen	National Taiwan University
David Doty	University of California, Davis
David Soloveichik	The University of Texas at Austin
Wooli Bae	University of Surrey
James Lathrop	Iowa State University
Constantine Evans	Maynooth University
Masami Hagiya	The University of Tokyo
Robert Schweller	University of Texas Rio Grande Valley
Satoshi Murata	Tohoku University
Jaimie Stewart	California Institute of Technology
Matthew Patitz	University of Arkansas
Chenxiang Lin	Yale University
Lulu Qian	California Institute of Technology
Manoj Gopalkrishnan	Indian Institute of Technology Bombay
Joseph Schaeffer	Google Health
Grigory Tikhomirov	University of California, Berkeley
Jonathan Bath	University of Oxford
Petr Šulc	Arizona State University
Lorenzo Di Michele	Imperial College London
Stefan Badelt	University of Vienna
Luca Cardelli	University of Oxford
Andrew Turberfield	University of Oxford
Chris Thachuk	University of Washington
Fei Zhang	Rutgers University
Lorenzo Rovigatti	Sapienza University of Rome
William Shih	Harvard University
Dominic Scalise	Johns Hopkins University
Yuan-Jyue Chen	Microsoft
Cody Geary	Aarhus University
Satoshi Kobayashi	The University of Electro-Communications
Eyal Nir	Beer Sheva University of the Negev
Anne Condon	University of British Columbia

Additional Reviewers for Tracks A and B

Siddharth Agarwal	Univeristy of California, Los Angeles
Andrew Alseth	University of Arkansas
Olivier Bournez	École Polytechnique
Xiangxiang Guan	Chinese Academy of Science
Daniel Hader	University of Arkansas
Shogo Hamada	Tohoku University
Ibuki Kawamata	Tohoku University
Huajie Liu	Tongji University
Dawn Nye	Iowa State University
Yuki Suzuki	Mie University
Yusuke Takezawa	University of Tokyo
Tao Zhang	Yantai University
Chao Zhou	Chinese Academy of Sciences

Organizing Committee for DNA 28

Matthew Lakin (Co-Chair)	University of New Mexico
Darko Stefanovic (Co-Chair)	University of New Mexico
David Arredondo	University of New Mexico
Peter Davenport	University of New Mexico
Jeremy Edwards	University of New Mexico
Steven Graves	University of New Mexico
Tracy Mallette	University of New Mexico

Sponsors

International Society for Nanoscale Science, Computation, and Engineering
Department of Computer Science, University of New Mexico
Vice President for Research, University of New Mexico
School of Engineering, University of New Mexico
National Science Foundation
New Mexico Consortium
Santa Fe Institute
Integrated DNA Technologies
Illumina
New Mexico INBRE

Fast and Robust Strand Displacement Cascades via Systematic Design Strategies

Tiernan Kennedy[1]
Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA, USA

Cadence Pearce[1]
Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA, USA

Chris Thachuk[2]
Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA, USA

Abstract

A barrier to wider adoption of molecular computation is the difficulty of implementing arbitrary chemical reaction networks (CRNs) that are robust and replicate the kinetics of designed behavior. DNA Strand Displacement (DSD) cascades have been a favored technology for this purpose due to their potential to emulate arbitrary CRNs and known principles to tune their reaction rates. Progress on *leakless* cascades has demonstrated that DSDs can be arbitrarily robust to spurious "leak" reactions when incorporating systematic domain level redundancy. These improvements in robustness result in slower kinetics of designed reactions. Existing work has demonstrated the kinetic and thermodynamic effects of sequence mismatch introduction and elimination during displacement. We present a systematic, sequence modification strategy for optimizing the kinetics of leakless cascades without practical cost to their robustness. An in-depth case study explores the effects of this optimization when applied to a typical leakless translator cascade. Thermodynamic analysis of energy barriers and kinetic experimental data support that DSD cascades can be fast and robust.

2012 ACM Subject Classification Applied computing → Chemistry; Computer systems organization → Molecular computing

Keywords and phrases DNA strand displacement, Energy barriers, Kinetics

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.1

Funding This work was supported by an NSF grant (CCF 2106695) and a Faculty Early Career Development Award from NSF (CCF 2143227).

1 Introduction

One goal of molecular programming is to design chemical systems that not only store and process information, but are able to manipulate matter with nanometer precision, to sense (bio-) chemical signals from their environment, to perform robust, complex and energy-efficient computation, and to actuate a physical response. This is not an easy goal. Yet, inspiring demonstrations of complex, enzyme-free circuits based on DNA strand displacement (DSD) [26] show the promise of this technology to realize the equivalent of a chemical central processing unit. While engineered DSD reactions, inspired by strand displacement in genetic recombination[14, 15], were studied as early as the 1980s [3] the groundwork for modern engineering of DSD cascades and other more complicated dynamic systems followed nearly twenty years later [19, 25].

[1] Authors contributed equally and are listed alphabetically by last name.
[2] To whom correspondence may be sent.

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham; Article No. 1; pp. 1:1–1:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sequence-independent "domain level" strategies have emerged to improve the design of high fidelity DNA strand displacement reactions and cascades. For example, recent work on *leakless* DSD cascades [18, 20, 21] have shown how these systems can be made arbitrarily robust: the rate of spurious leak reactions – that produce output in the absence of correct input – can be decreased exponentially by a linear increase to the length of the cascade. However, this increased robustness comes at a cost: designed reaction pathways have slower kinetics. This slowdown of desired kinetics can result from longer cascades, from cascades that are a series of reversible displacements when toehold-sized clamps are employed [20], and from unproductive reactions attributed to the domains shared between multiple signal strands within the same cascade (see Figure 1b).

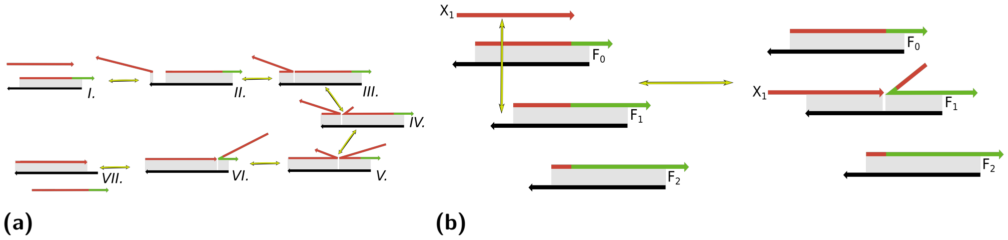


Figure 1 **(a)** A visual representation of a toehold mediated strand exchange reaction. *I.* Initial system: translator gate with invader strand (red), incumbent strand (red/green), and substrate strand (black). *II.* Initial base pair formed between translator toehold and signal strand. *III.* Toehold fully bound to signal strand. *IV.* First base pair in translator frays, incurring a free energy penalty ΔG_p for branch migration initiation [16, 10]. *V.* Branch migration occurs: base pairs between the translator strands are replaced by base pairs between the signal strand and the substrate strand (black). *VI.* Branch migration completes: incumbent strand is bound to toehold sized domain on substrate strand. *VII.* Incumbent strand dissociates from complex. **(b)** While systematic redundancy introduced in leakless cascades inhibits spurious displacement the resulting overlap in sequence space among complexes introduce new pathways for spurious invasion. Spurious invasion events do not necessarily lead to spurious displacement (*i.e.*, leak). However, these unproductive pathways sequester chemical species that are kinetically relevant to the designed reaction pathway decreasing its effective rate. The *fuel* complexes F_0, F_1 and F_2 comprise a displacement cascade that mediates the formal reaction $X_1 \rightarrow Y_1$. Illustrated is the (unproductive) spurious invasion of F_1 by the cascade input signal X_1; similarly X_1 can occlude the toehold of and spuriously invade F_2.

In parallel, sequence-dependent strategies that introduce and/or eliminate Watson-Crick-Franklin [23, 9] base-pairing violations, or "mismatches", have been studied as an effective means for tuning the kinetics and thermodynamics of DSD reactions. A study from as early as 1986 investigated the impact of a single base-pair mismatch as a parameter for engineering strand displacement reactions [3]. The effects of mismatch elimination (and introduction) in strand displacement have been widely studied [6, 1, 12] and include a thermodynamic driving force from mismatch elimination, as well as reaction rate changes that are highly dependent on the distance of the mismatch position from the site of invasion. Mismatches have been used both strategically [24, 7] and systematically [11, 13] to alter reaction kinetics [24] and thermodynamic driving forces [7, 12, 5], as a mechanism for single nucleotide polymorphism (SNP) identification [8, 17], and as a means to combat leak [13, 11, 7].

While strategies for systematic domain-level optimization of DSD circuits and those for sequence-level optimization of DSD circuits were developed primarily in parallel, they are not mutually exclusive. The drawbacks introduced to designed reaction pathways when employing domain level strategies such as leakless motifs and/or the use of clamp domains, namely slowed kinetics and/or a reduction in thermodynamic driving force, might be remedied by sequence-level strategies. Here we demonstrate a synthesis of these two strategies to arrive at cascade designs that are robust by virtue of their domain level redundancy, yet have fast kinetics due to a systematic sequence level optimization strategy based on strategic mismatch elimination and introduction. By composing the state-of-the-art for domain level and sequence level design of DSD systems we demonstrate a rich design space for fast and robust DSD cascades. We support the efficacy of this strategy with theory based on thermodynamic modeling and preliminary experimental verification.

2 Systematic mismatch strategies for leakless cascades

Due to the domain level redundancy, introduced by *leakless* cascades in order to combat leak [18, 20], unproductive invasion is possible – signal strands present initially as input, or produced via displacement of incumbent strands, can interact with multiple fuel complexes. For example, Figure 1b illustrates how the input strand X_1 can spuriously invade fuel complex F_1 – not shown is that X_1 can also spuriously invade fuel complex F_2. While overall these unproductive invasions are unlikely to cause leak events they do create kinetic slowdown; a signal strand (transiently) bound to the wrong fuel is signal not currently propagating through the cascade and is also a source of occlusion of its bound fuel and thus a source of inhibition of other signal propagation. A confounding factor in kinetic slowdown is the use of toehold-sized clamps. This design choice has been demonstrated to suppress leak to even lower rates in leakless cascades [20], but at the cost of thermodynamic driving force since each designed reaction is based on reversible toehold exchange (see Figure 1a and Figure 5).

To combat these problems we introduce a systematic sequence level modification strategy to introduce *sentinel* positions.

▶ **Definition 1** (sentinel position). *Any position i such that an intended invader forms a Watson-Crick-Franklin base pair and every spurious invader introduces a mismatch.*

Table 1 Sequence-level strategies to meet design goals. Reasonable choices for parameters α and β are discussed in Section 2.1.

Goal	Strategy
1. Barrier to spurious toehold binding	Sentinel position in each toehold domain
2. Additional barrier to spurious invasion	Sentinel positions “close” to helix ends
3. Additional thermodynamic drive	Sentinel positions in each fuel complex
4. Avoid increasing leak	No internal loops within α nucleotides of any helix end
	No internal loops within β nucleotides of each other

Our aim is to systematically create energy barriers that must be encountered early in any unproductive invasion pathway. Furthermore, these modifications should not be at the expense of cascade robustness and, when possible, should introduce kinetic improvements and additional thermodynamic driving force for designed reactions. While the strategy is generally applicable to any system with redundant domains across a cascade we will focus on translator cascades for the sake of clarity. Consider the leakless translator cascade with toehold-sized clamps to emulate the reaction $X_1 \rightarrow Y_1$ illustrated in Figure 2 (Step 0). These additional clamps provably increase the barrier to leak reactions [22], but individual steps in the cascade are reversible; similarly, the overall cascade is reversible and thus emulates the reaction $X_1 \leftrightarrow Y_1$ with forward and reverse rate constants dependent on the strength of forward and reverse toeholds. Can we keep the robustness benefit provided by the redundant domains *and* these additional clamps yet drive the cascade forward as if implemented as a series of "irreversible" reactions without clamps (*i.e.,* with heavily favored forward rates)? Can we significantly decrease the rates of unproductive reactions? As it turns out these goals can be simultaneously met.

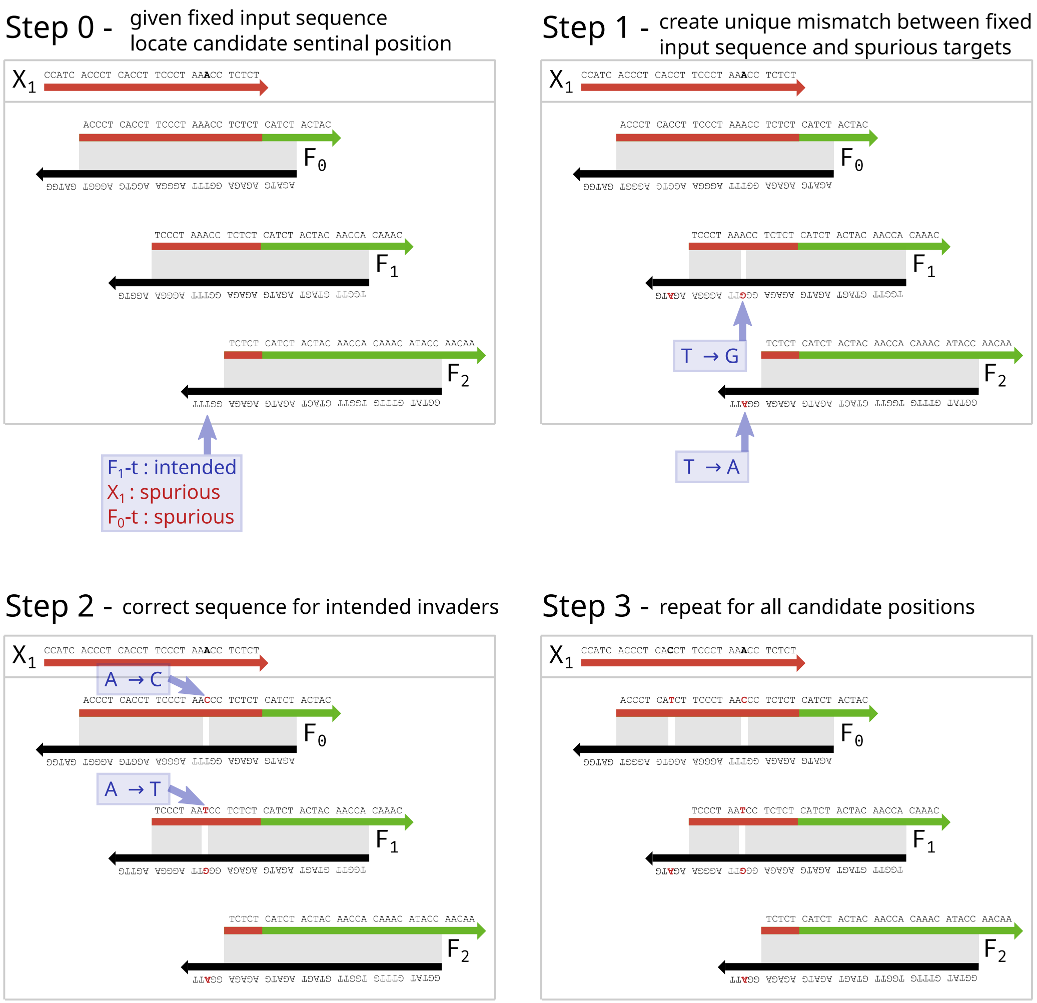

Step 0 - given fixed input sequence locate candidate sentinal position
Step 1 - create unique mismatch between fixed input sequence and spurious targets
Step 2 - correct sequence for intended invaders
Step 3 - repeat for all candidate positions
X_1
CCATC ACCCT CACCT TCCCT AAACC TCTCT
ACCCT CACCT TCCCT AAACC TCTCT CATCT ACTAC
F_0
TCCCT AAACC TCTCT CATCT ACTAC AACCA CAAAC
F_1
TCTCT CATCT ACTAC AACCA CAAAC ATACC AACAA
F_2
F_1-t : intended
X_1 : spurious
F_0-t : spurious
T → G
T → A
A → C
A → T
ACCCT CACCT TCCCT AACCC TCTCT CATCT ACTAC
TCCCT AATCC TCTCT CATCT ACTAC AACCA CAAAC
CCATC ACCCT CACCT TCCCT AAACC TCTCT
ACCCT CATCT TCCCT AACCC TCTCT CATCT ACTAC

Figure 2 A general strategy for creating sentinel positions given fixed input and output sequences of a cascade. (Step 0) A candidate position i is identified. (Step 1) Unique mismatches are introduced into substrate (bottom) strands that intersect i and are not designed to interact with the fixed input. (Step 2) Sequences are corrected for intended (top strand) invaders. (Step 3) The process is repeated for candidates intersecting the fixed input. (Step 4 – not depicted) Symmetric process run to introduce sentinel positions with respect to the output sequence to guard against spurious invasion in the reverse pathway; see Figure 3 for a complete example with fixed input and output sequences.

Sentinel positions will introduce mismatches within fuel complexes. Obvious candidate positions are those intersecting toeholds within a cascade since weakening the toehold binding of spurious invaders will decrease the rate of unproductive reactions (see Section 3). However, these positions should not be chosen if they intersect fuel complexes near the end of a helix as fraying could be exacerbated by introduced mismatches and result in increased rates of leak. In those cases, positions can be chosen that intersect further away from helix ends (*e.g.* at least a "toehold"-sized domain away). Figure 2 demonstrates how sentinel positions can be created with respect to a fixed input and output sequence of a cascade. The aim of this strategy is to internally optimize a cascade that implements a formal reaction without changing its interface to other system components (*i.e.*, the input sequence it accepts and the output sequence it produces). We note that any change to the sequences of signal strands has the potential to create spurious events with other cascades in the system, or create unwanted secondary structure. These can be incorporated as hard constraints when implementing the sentinel position design strategy.

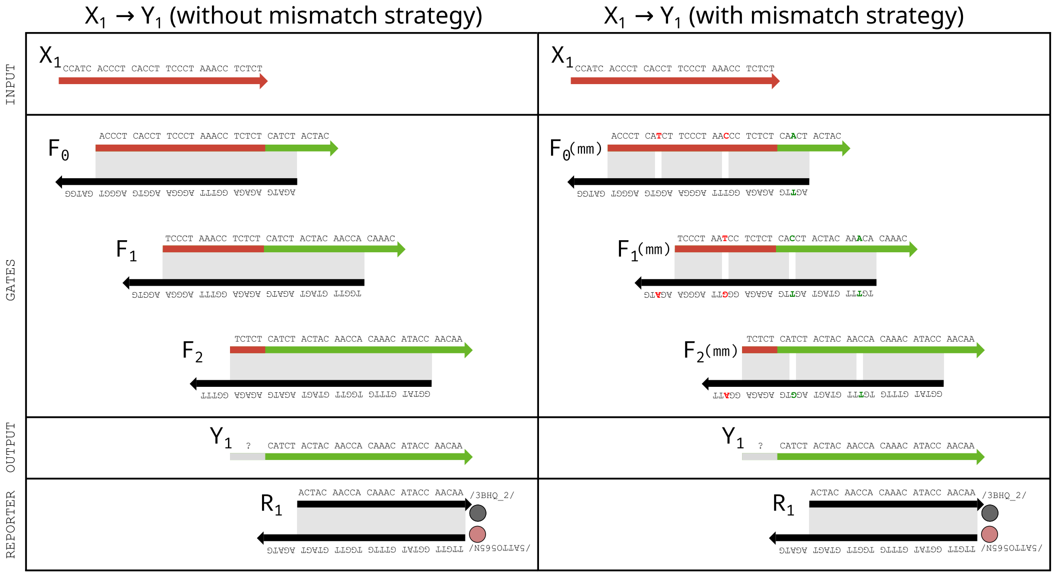

X1 → Y1 (without mismatch strategy)
X1 → Y1 (with mismatch strategy)
INPUT
GATES
OUTPUT
REPORTER
X1 CCATC ACCCT CACCT TCCCT AAACC TCTCT
F0 ACCCT CACCT TCCCT AAACC TCTCT CATCT ACTAC
F1 TCCCT AAACC TCTCT CATCT ACTAC AACCA CAAAC
F2 TCTCT CATCT ACTAC AACCA CAAAC ATACC AACAA
Y1 ? CATCT ACTAC AACCA CAAAC ATACC AACAA
R1 ACTAC AACCA CAAAC ATACC AACAA /3BHQ_2/
/5ATTO565N/
X1 CCATC ACCCT CACCT TCCCT AAACC TCTCT
F0(mm) ACCCT CATCT TCCCT AACCC TCTCT CAACT ACTAC
F1(mm) TCCCT AATCC TCTCT CACCT ACTAC AAACA CAAAC
F2(mm) TCTCT CATCT ACTAC AACCA CAAAC ATACC AACAA
Y1 ? CATCT ACTAC AACCA CAAAC ATACC AACAA
R1 ACTAC AACCA CAAAC ATACC AACAA /3BHQ_2/
/5ATTO565N/

Figure 3 Domain level and sequence level designs for a $X_1 \rightarrow Y_1$ translator without mismatch strategy (left) and with mismatch strategy (right). Both systems were designed to share the same input "trigger" strand and same output strand (and reporter). Locations and sequences of mismatch positions are indicated around gaps in DNA duplexes.

Figure 3 gives a complete example of introducing sentinel positions that optimizes with respect to both the forward and (undesirable) reverse reaction pathways given a fixed input and output sequence. This system will serve as our detailed case study in subsequent sections.

2.1 Fragile regions of design space

For a collection of sequence modifications of a given DNA cascade there exist many poor sequence designs. Modification within a particular cascade may introduce unwanted secondary structure or introduce spurious interactions with species of another cascade within the same system. These conflicts can be modelled as hard or soft constraints to be evaluated when considering a modification candidate. However, there are certain design choices that are likely to yield poor candidates and can be eliminated from consideration entirely. For example, choosing a sentinel position too close to the end of a helix will result in increased fraying that in turn could lead to increased leak rates (see Figure 4a). Similarly, placing neighboring sentinel positions too close could result in the intervening duplex structure being destabilized and merging into a large internal loop, again leading to increased leak rates (see Figure 4c). We parameterize these distances in our overall design considerations.

▶ **Definition 2** (α). *The minimum distance of any sentinel position from a helix end (including nicks).*

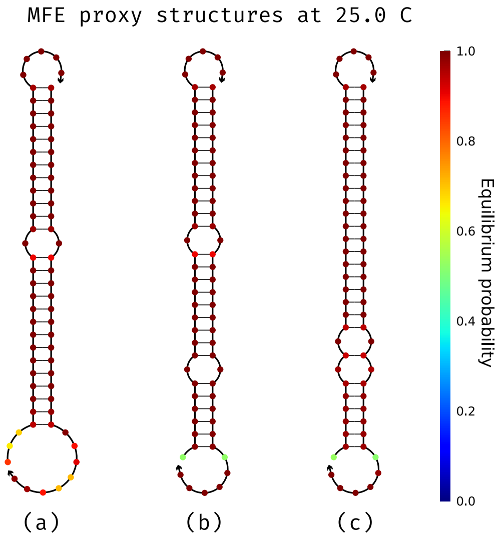


Figure 4 Stability of gate complexes and their propensity to fray at helix ends or merge neighboring mismatches into larger loops for candidate designs can depend on parameters α and β. Three candidate designs generated with (a) $\alpha = 2, \beta = 5$, (b) $\alpha = 5, \beta = 5$, and (c) $\alpha = 5, \beta = 2$.

▶ **Definition 3** (β)**.** *The minimum distance between neighboring sentinel positions.*

Both of these distance parameters can be chosen for a particular cascade or particular reaction condition based on simple criteria: at what distance do the intended secondary structures of fuel complexes have unacceptable ensemble defect [4]?

2.2 A naive sentinel design algorithm

Beginning from reference sequences the sentinel modification strategy naturally yields a naive algorithm: for every possible combination of positions that span the interval covered by a cascade, such that no position is within distance α of a helix end and no neighboring positions are within distance β of each other, evaluate all sequence modifications that yield valid sentinel positions and disfavor each spurious invasion relative to designed reactions. Any candidate design can be rejected for violating user-defined constraints such as propensity to form undesired intra- or inter-molecular secondary structure, or by containing forbidden sequence motifs. As we will see in Section 3.3 the space of candidate designs is sufficiently small in practice that it is tractable to enumerate and evaluate each one, yet the number of candidate designs that yield improvements, relative to the unmodified cascade, is large.

3 Thermodynamic modeling

The thermodynamic landscapes of conventional strand displacement and exchange reactions based on our translator system (Figure 3) were modeled using NUPACK [4, 2] at typical conditions (25 °C, 12.5 $\mathrm{mmol\,L^{-1}}$ $\mathrm{Mg^{2+}}$, 50 $\mathrm{mmol\,L^{-1}}$ $\mathrm{Na^{+}}$). Free energy values (ΔG) were plotted for each step in the conventional pathways for each reaction under study and incorporated a penalty for branch migration initiation $\Delta G_{\mathrm{p}} = 2\,\mathrm{kcal\,mol^{-1}}$ [10, 16]. Unless otherwise noted we use the structure free energy of an unstructured invader and fuel complex with a toehold-sized clamp (F_2), both at 10 $\mathrm{nmol\,L^{-1}}$ concentration, as the reference microstate defined to have $\Delta G = 0\,\mathrm{kcal\,mol^{-1}}$.

3.1 Toehold-mediated strand exchange (TMSE)

A common strategy to reduce leak in DSD systems is to introduce toehold-sized clamps. With respect to our translator gate F_2, when moving from a displacement reaction (Figure 5, blue) to an exchange reaction (Figure 5, black) the structural free energy of the initial microstate is improved, but at the cost of $\approx 6\,\mathrm{kcal\,mol^{-1}}$ forward thermodynamic driving force. This difference ($\Delta\Delta G_{\mathrm{exchange}} - \Delta\Delta G_{\mathrm{displacement}}$) translates directly from the loss in additional base-pairs formed via displacement.

While strand exchange is typically thermodynamically neutral, strand exchange in a translator utilizing the sentinel mismatch strategy is thermodynamically favorable. Consider the specific comparison in Figure 6. The initial structural free energy of $F_2(mm)$ is higher than that of F_2, since $F_2(mm)$ has fewer paired bases, as well as mismatches destabilizing its helix. Both systems pay a concentration-dependent entropic penalty (calculated based on experimental concentration of each reactant at $10\,\mathrm{nmol\,L^{-1}}$) when base 1 of the toehold pairs (microstate II). Throughout invasion of $F_2(mm)$ by $F_1(mm)$-t (red pathway) the system drops in energy each time a mismatch is replaced by a Watson-Crick-Franklin base pair. After the last mismatch has been replaced, both strand exchange systems follow essentially the same energy path throughout the remainder of the invasion and toehold dissociation (microstates VI-VII). Without the mismatch strategy the change in structural free energy $\Delta\Delta G_{\text{no mismatch}} = -0.71\,\mathrm{kcal\,mol^{-1}}$, while with the mismatch strategy $\Delta\Delta G_{\mathrm{mismatch}} = -10.71\,\mathrm{kcal\,mol^{-1}}$. Additionally, the reverse reaction of the translator using the mismatch strategy has a significant uphill energy activation barrier ($\Delta\Delta G_{\mathrm{rev(mismatch)}} = 19.75\,\mathrm{kcal\,mol^{-1}}$), while the activation energy for the reverse reaction of the strand exchange system without mismatches ($\Delta\Delta G_{\text{rev(no mismatch)}} = 9.75\,\mathrm{kcal\,mol^{-1}}$) is largely due to the entropic penalty of toehold binding ($9.04\,\mathrm{kcal\,mol^{-1}}$). This additional driving force has been studied in greater detail in other work [5, 10].

When applied systematically within a cascade, as in our sentinel position design strategy, mismatches raise the energy barrier for unproductive strand invasions in leakless systems. Sequence redundancies in the leakless system allow upstream signal strands to partially invade downstream translators via toehold binding, and subsequent displacement of matching sequence (see Figure 1). The systematic mismatch strategy ensures that signal strands will mismatch with every translator except their intended target. Figure 7 shows the conventional pathways of invasion of $F_2(mm)$ by its intended invader ($F_1(mm)$-t, blue) and two other spurious invaders ($F_0(mm)$-t, green and X_1, orange). Note that without systematic mismatches the conventional invasion pathway for each spurious invader would simply be a prefix of the intended conventional invasion pathway.

We investigated the effect of the mismatch strategy in greater detail on invasion pathways of F_2 and $F_2(mm)$ when reactants were both at low concentrations ($10\,\mathrm{nmol\,L^{-1}}$) and when both were at high concentrations ($10\,\mathrm{\mu mol\,L^{-1}}$). For unproductive reactions only the most favorable spurious invader was considered – that is, the corresponding spurious invader with the largest overlap with the fuel complex; F_0-t in the control case and $F_0(mm)$-t when using the mismatch strategy. In the no mismatch case at low concentration (Figure 8a) the energy pathway for both invaders is identical until base 20 when the spurious invader completes its invasion and recovers the branch migration initiation penalty $\Delta\Delta G_{\mathrm{p}}$ (Figure 8b, microstate IV). In contrast, when the mismatch strategy is employed at low concentration the spurious invasion pathway (i) necessarily deviates from that of the intended invasion, (ii) must overcome a higher energy barrier to invasion (Figure 8b microstate III vs microstate II), and (iii) terminates at a significantly less favorable resting microstate compared with the free energy of the intended reaction having made similar invasion progress (Figure 8b microstate IV).

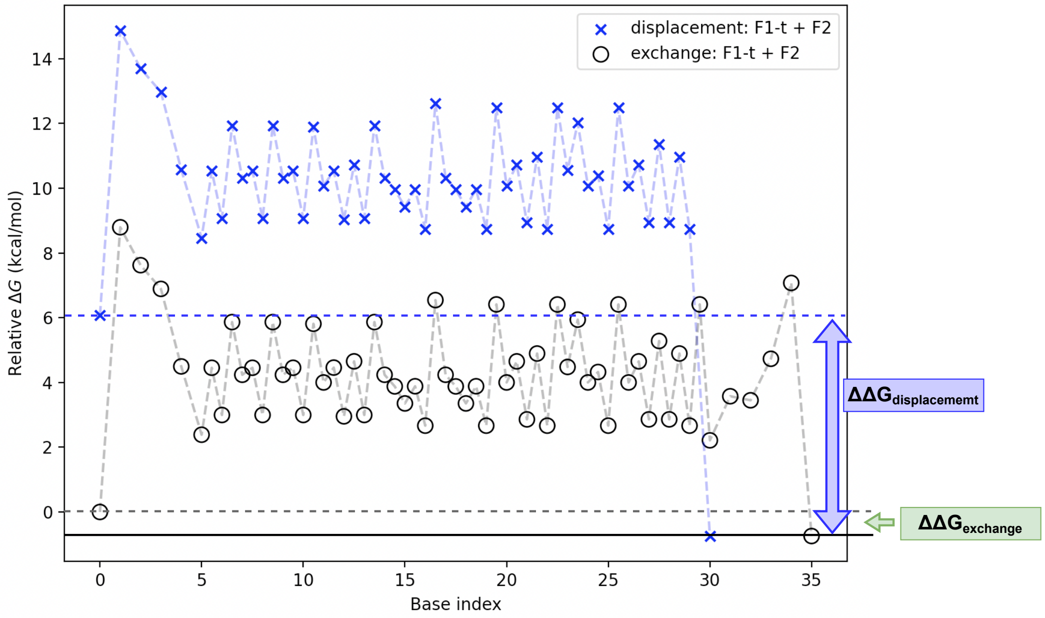


Figure 5 (black) The energy landscape for a conventional toehold mediated strand exchange (TMSE) between complex F_2 and its intended trigger strand F_1-t of Figure 3 (left). **(blue)** A similar landscape is shown for a modified F_2 complex that lacks a toehold-sized clamp thus resulting in a toehold mediated strand displacement (TMSD) reaction.

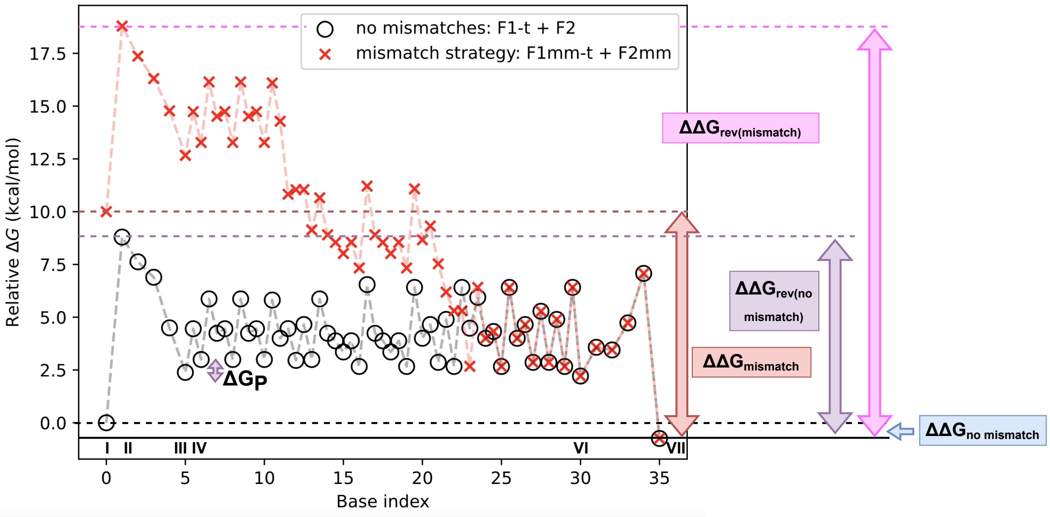


Figure 6 (red) The energy landscape for a conventional toehold mediated strand exchange (TMSE) between complex $F_2(mm)$ and its intended trigger strand $F_1(mm)$-t that results in two mismatch elimination events. **(black)** A similar landscape is shown for F_2 and invader F_1-t without a mismatch strategy. Roman numerals reference system microstates described in Figure 1a.

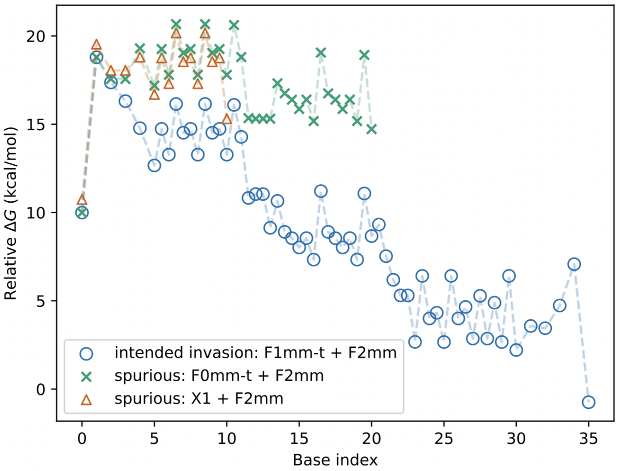
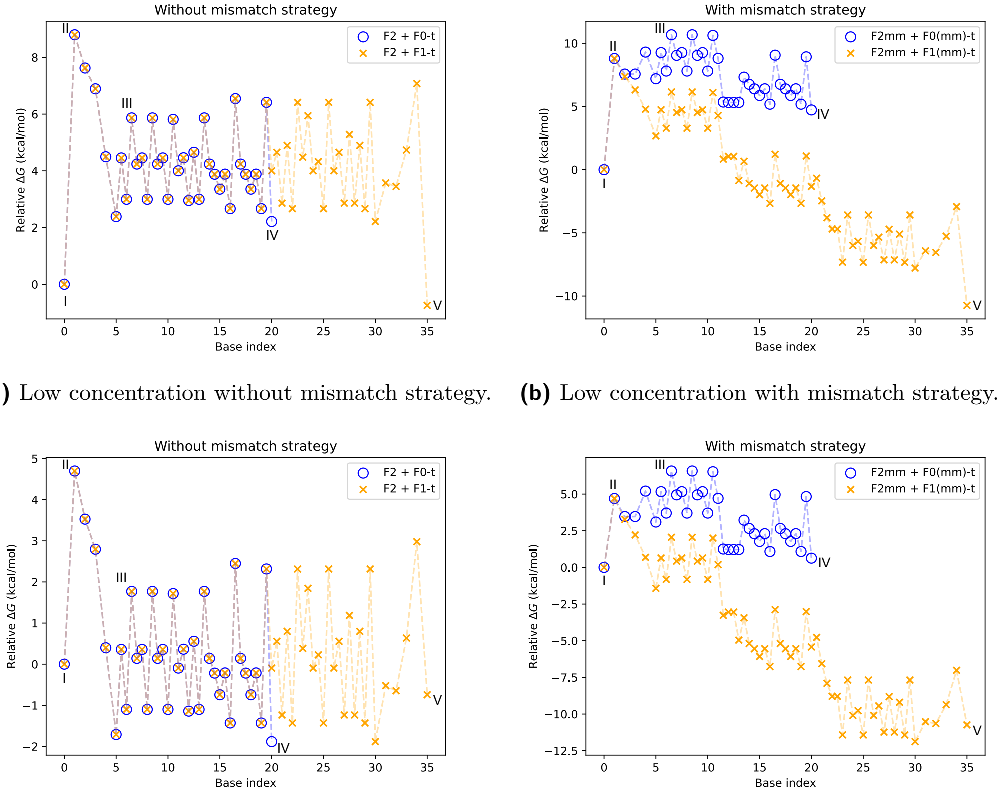


Figure 7 Mismatch strategy for TMSE in $F_2(mm)$ complex. Mismatches disfavor spurious invaders $F_0(mm)$-t (green) and X_1 (orange), and favor the intended invader $F_1(mm)$-t (blue).



(a) Low concentration without mismatch strategy. **(b)** Low concentration with mismatch strategy.

(c) High concentration without mismatch strategy. **(d)** High concentration with mismatch strategy.

Figure 8 Comparison of the conventional pathways for intended and spurious invasion without and with the mismatch strategy – left column and right column, respectively – and at low concentration ($10\,\mathrm{nmol\,L^{-1}}$) and at high concentration ($10\,\mathrm{\mu mol\,L^{-1}}$) – top row and bottom row, respectively. Deviating from prior convention, note that the free energies reported in **(c)** and **(d)** are relative to structure free energy of $F_2(mm)$ and an unstructured invader.

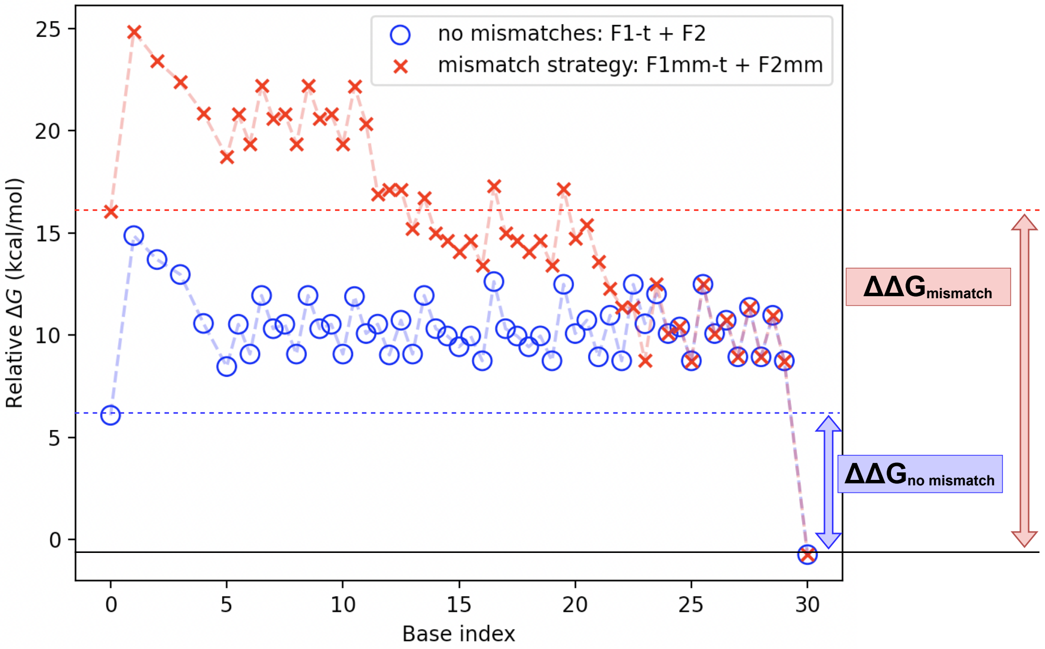


Figure 9 The thermodynamic driving force already present in TMSD **(blue)** can be significantly increased when intended invaders eliminate multiple mismatches **(red)**.

These same trends hold at higher concentrations (see Figure 8c and Figure 8d); however, there is an important distinction to consider. At the high micromolar concentrations typically used to increase the speed of leakless strand exchange ($1\,\mu\mathrm{mol\,L^{-1}}$–$10\,\mu\mathrm{mol\,L^{-1}}$) [22, 21] the entropic penalty of toehold binding is reduced (microstate II in Figures 8c and 8d). Modeling of translator gate F_2 at high concentration (Figure 8c) shows that not only is it equally as favorable for a spurious invader to initiate branch migration as the desired invader, but that many of the microstates during spurious invasion (*e.g.*, microstate IV) are energetically favorable to the initial configuration (*i.e.*, microstate I). Therefore, as the concentration of the non mismatch cascades increase – and thus the thermodynamic favorability of microstates with occluded toeholds and sequestered signal strands – so too do the kinetic inhibitions caused by (unproductive) spurious invasions. In contrast, even at high concentrations, the mismatch strategy remedies these issues by (i) providing an additional energy barrier to spurious invasion (Figure 8d, microstate III), (ii) ensuring all microstates of spurious invasion are unfavorable relative to the initial configuration (microstates II-IV compared with microstate I, Figure 8d), and (iii) maintaining a thermodynamic driving force ($\Delta\Delta G_{\mathrm{mismatch}} \approx -10\,\mathrm{kcal\,mol^{-1}}$) for the intended TMSE reaction (Figure 8d, orange, microstate I to microstate V).

3.2 Toehold mediated strand displacement (TMSD)

In Section 3.2 we consider invasions of a variant of the $F_2(mm)$ fuel complex that lacks a toehold sized clamp – the first five nucleotides from the $5'$ end of strand $F_2(mm)$-b are truncated. While TMSD already has a significant driving force due to the free energy improvement of a toehold's worth of additional base pairing ($\Delta\Delta G_{\text{no mismatch}} \approx -6\,\mathrm{kcal\,mol^{-1}}$, Figure 9) the mismatch strategy offers a means to increase this driving force further ($\Delta\Delta G_{\mathrm{mismatch}} \approx -16\,\mathrm{kcal\,mol^{-1}}$, Figure 9).

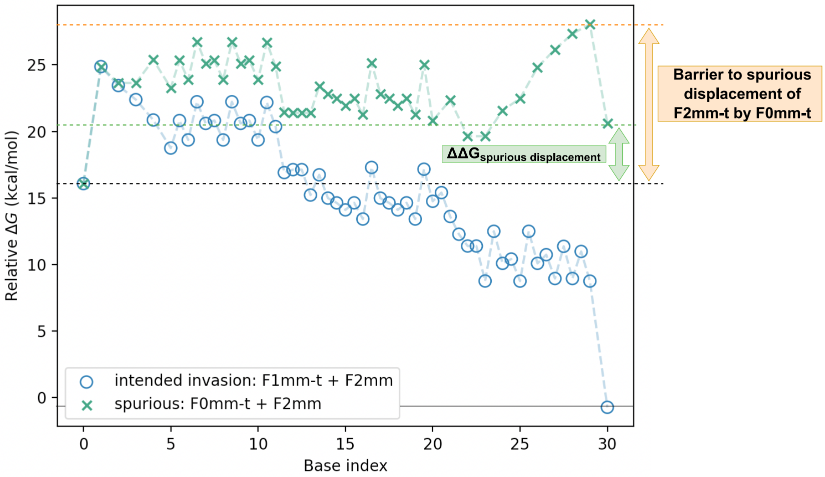


Figure 10 The sentinel mismatch strategy is compatible with leakless TMSD cascades.

Unlike our TMSE case study, where spurious invasions are unlikely to result in leak reactions, the spurious invasion of $F_2(mm)$ by invader $F_0(mm)$-t can reach a microstate where the incumbent strand $F_2(mm)$-t is bound by only ten nucleotides (base index 25, Figure 10) and could therefore be susceptible to experimentally relevant rates of dissociation. However, the sentinel mismatch strategy can result in designs that kinetically and thermodynamically disfavor leak in TMSD.For example, in the variant considered here (i) leak is thermodynamically unfavorable ($\Delta\Delta G_{\text{spurious displacement}} = 4.52\,\text{kcal mol}^{-1}$), and (ii) the kinetic barrier to spurious displacement is greater than $11\,\text{kcal mol}^{-1}$.

3.3 Design space of sentinel mismatch strategy

Beginning from the reference sequences of Figure 5 (left) we used the naive algorithm from Section 2.2 to generate each of the 685,307 candidate sentinel designs that were valid for $\alpha = \beta = 5$. Thermodynamic properties were evaluated for each candidate including its minimum among all possible $\Delta\Delta G_{\text{spurious}}$ – the net free energy of the microstate representing the maximum progress of a spurious invasion (*e.g.*, microstate IV in Figure 8b) – and $\Delta\Delta G_{\text{translator}}$ – the net free energy of the overall cascade via designed reactions (*i.e.*, a proxy for its thermodynamic driving force). In our analysis that follows and in the annotations of Figure 11, *our design* refers to the sentinel mismatch sequences of Figure 3 (right).

The full design space visualized as a density plot of $\Delta\Delta G_{\text{translator}}$ for each candidate design is shown in Figure 11a; this includes the reference sequence design, which contain no sentinel position, as it is valid by our definition. Note that *our design*, which we investigated the thermodynamic properties of in this section and experimentally evaluated and discuss in the next section, has worse thermodynamic driving force than the average across all valid candidate designs.

Optimizing for only the driving force of the overall cascade does not necessarily result in designs that mitigate spurious invasion. In practice, one may wish to optimize for some trade-off between multiple properties. Figure 11b plots the Pareto frontier of maximizing the minimum $\Delta\Delta G_{\text{spurious}}$ and minimizing $\Delta\Delta G_{\text{translator}}$. Our design is not expected to be on this front because it contains fewer sentinel positions than are permitted by the $\alpha = \beta = 5$ constraint. Overall, the sentinel mismatch strategy allows further room for

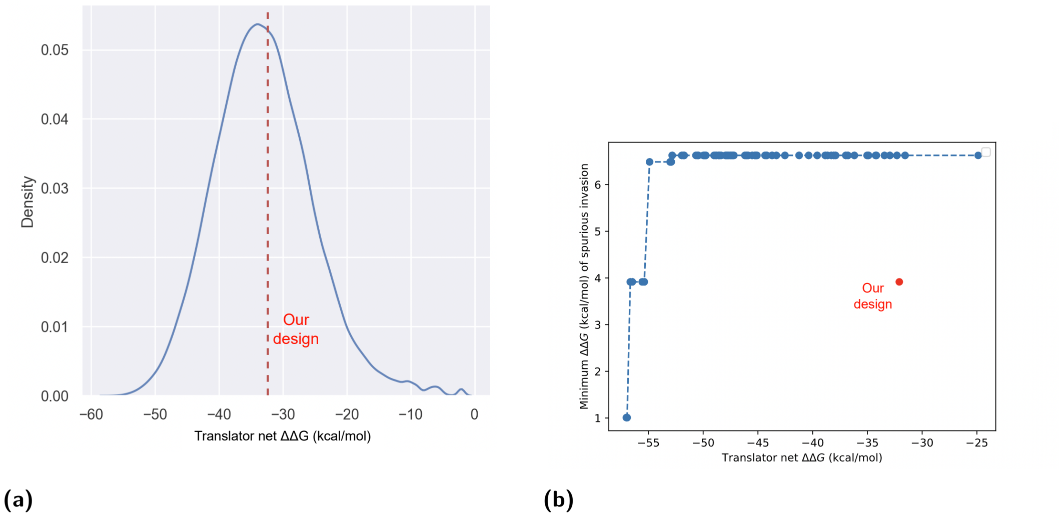


Figure 11 **(a)** Density distribution of all valid sentinel design candidates starting from the non mismatch reference sequences – see Figure 3 (left) – that obey $\alpha = \beta = 5$ and disfavor spurious invasion. **(b)** Pareto frontier for maximizing the minimum $\Delta\Delta G_{\text{spurious}}$ – the net free energy of any spurious invasion in a design – and minimizing $\Delta\Delta G_{\text{translator}}$ – the net free energy of the overall cascade via designed reactions.

translator optimization and can be broadly applied to generate a rich design space for any cascade with redundancy. A final design can be chosen based on multiple, user-defined criteria and constraints utilizing tools such as NUPACK [4] for evaluation.

4 Preliminary Experimental Verification

To demonstrate the efficacy of a systematic domain *and* sequence level design strategy for DSD cascades we compared a typical leakless translator as a control to a variant that incorporates sentinel positions. Our control is a cascade that requires three successive displacement reactions to translate an input to a sequence independent output and has a large entropic and enthalpic barrier to leak [20] due to its domain level redundancy and incorporation of toehold-sized clamps, making it reversible (see Figure 3 (left)). However, the redundancy within the control cascade creates a number of (unproductive) spurious invasion pathways. The variant tested incorporates the systematic mismatch strategy of sentinel positions described in Section 2 (see Figure 3 (right)). Sequences were designed using NUPACK [2, 4] to verify that all complexes were well-formed, all signal strands were unstructured, all desired toehold exchange reactions netted $\approx 0\,\text{kcal}\,\text{mol}^{-1}$ in structure free energy, and sequence motifs that might affect synthesis yield – such as homomeric repeats – were absent. To facilitate direct comparison between the translator systems with and without the sentinel strategy both systems were designed to use the same input strand as a trigger, produce identical sequences as outputs, and to use identical reporter complexes that consume this output sequence. All designed sequences were ordered from IDT with PAGE purification; annealed complexes were PAGE purified and concentrations were measured via UV absorbance and then normalized.

4.1 Leak reactions

Based on work by Wang *et al.* [21, 20], because of the levels of redundancy and presence of toehold-sized clamps, we expected negligible rates of leak in the non-mismatched cascade (our control). A recent experimental demonstration of a translator with the same domain-level design, but differing in sequence, had no significant rate of leak even at micromolar concentrations [22]. However, in order for the sentinel mismatch strategy to provide a valid remedy to the adverse kinetic effects introduced by domain level strategies that address robustness, it must itself not re-introduce significant leak. Thus we hoped to find similarly low levels of leak in both the non-mismatched and mismatched translator cascades.

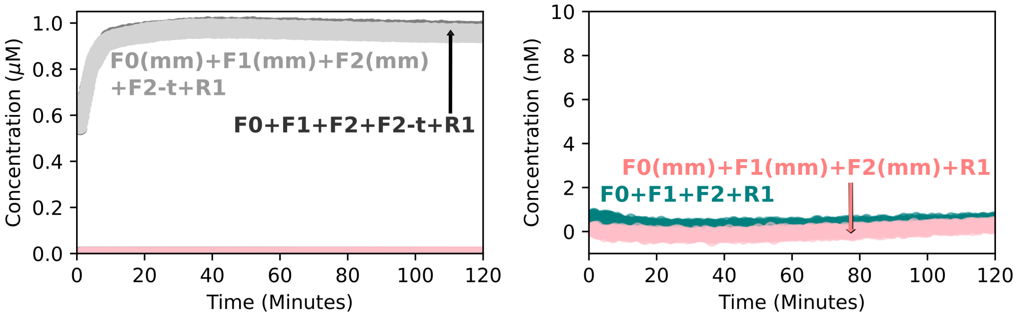


Figure 12 Testing the presence of output signal in the absence of valid input (leak) in leakless translators with and without the sentinel mismatch strategy. (Right) Time-course fluorescence results when incubating 1 µmol L^{-1} of each fuel complex in a given translator system with 1.5 µmol L^{-1} reporter and no input trigger strands are reported for the non-mismatched and mismatched translator systems in teal and pink respectively. (Left) Internal standards of the aforementioned systems spiked with final concentrations of 1 µmol L^{-1} of strand F_2-t (direct trigger to the reporter) are plotted for the same interval. All replicates are plotted for each of the experimental conditions tested. Both the mismatched and non-mismatched systems leaked a maximum of 0.054% of the total possible output in 2 hours, whereas the high controls triggered at the reporter with 1 µmol L^{-1} F_2-t reached over 50% completion in the 50 seconds between mixing of reactants and the first fluorescence reading.

To verify leak did not increase when using the mismatch strategy we prepared two sets of triplicate mixtures: one containing purified F_0, F_1, and F_2 at final nominal concentrations of 1 µmol L^{-1} and fluorescent output reporter R_1 at a final concentration of 1.5 µmol L^{-1}, and the other containing purified $F_0(mm)$, $F_1(mm)$, and $F_2(mm)$ at final concentrations of 1 µmol L^{-1} and R_1 at a final concentration of 1.5 µmol L^{-1}. Both of these sets of samples were compared to triplicate measurements of high controls of the same mixtures prepared with a final concentration of 1 µmol L^{-1} F_2-t that directly triggers the output reporter. Each reaction sample was quickly mixed and observed under time-course fluorescence measurements every 20 seconds for 2 hours. Raw fluorescence values were then normalized to concentration using internal low and high controls. With the exception of three data series which were discarded due to sample evaporation the results across all three replicates are shown in Figure 12.

From Figure 12 it appears that mismatch introduction did not significantly alter the rate of leak in our system. While the high control reactions reached above 50% completion in the 50s interval between the mixture of reaction components and the start of fluorescence measurements, the untriggered samples for both the non-mismatched and mismatched systems leaked at most 0.054% in 2 hours.

4.2 Designed reactions

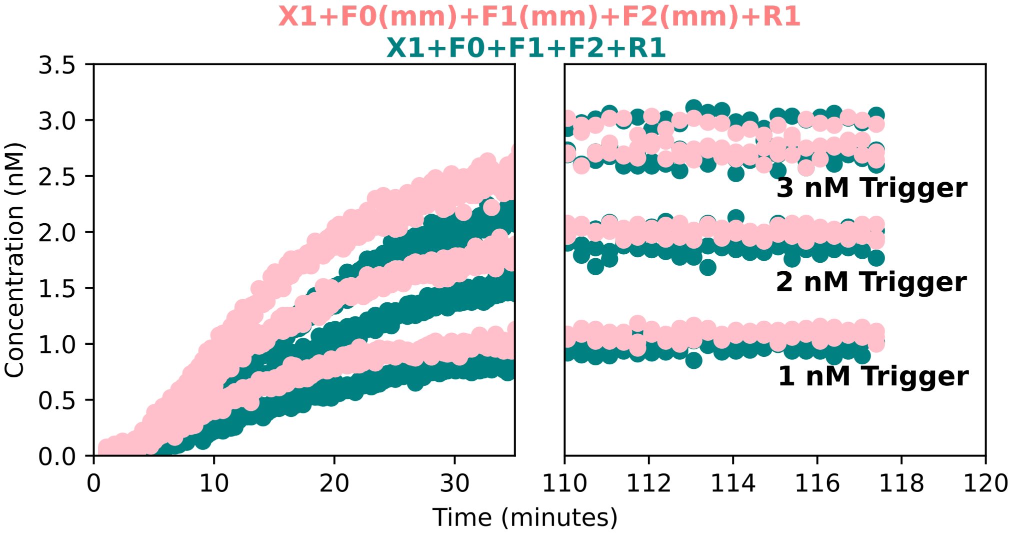


Figure 13 Comparing the triggering kinetics of leakless translator systems with and without the mismatch strategy. Each reaction was conducted with 10 nmol L^{-1} of each translator gate in the respective cascade and 15 nmol L^{-1} of output reporter R_1. The x-axis is broken to include only the beginning of the reaction and the equilibrated endpoints of the reaction. Raw fluorescence of each reaction was converted to concentration by internal normalization with a low control of untriggered cascade and high controls of cascade triggered with 10 nmol L^{-1} F_2-t at the reporter. The normalized output concentrations for each of the three replicates for each experimental condition were plotted and labeled with their reaction components as given in Figure 3. Clear differences in reaction half-times between the two systems at each concentration, even considering the variance across triplicate measurements, suggests that the introduction of mismatches increased the effective forward rate of the translator system compared to the non-mismatched variant.

In the previous section we presented evidence supporting that our strategy for introducing systematic sentinel mismatches into leakless translator cascades does not significantly increase leak relative to the non-mismatched case. This fact alone suggests some flexibility for sequence design in translator systems and supports a notion of compatibility between redundancy based leak-reduction and mismatch-based optimization strategies. However, we also theorized that the mismatch strategy would increase the observed reaction rate of a translator system by introducing kinetic barriers to unproductive reaction pathways that temporarily sequester reaction intermediates relevant to the desired reaction rate. To test this hypothesis, we measured the time-course fluorescence of triplicate samples of mismatched and non-mismatched translator cascades at 10 nmol L^{-1} gate concentration, 15 nmol L^{-1} reporter concentration, and input trigger concentrations of 1 nmol L^{-1}, 2 nmol L^{-1}, and 3 nmol L^{-1}. Additionally, we recorded triplicate measurements of high controls of each of non-mismatched and mismatched cascade with gates and reporters at the previous concentrations, and with 10 nmol L^{-1} F_2-t to directly trigger the reporter. Lastly, we measured untriggered low control cases of both the mismatched and non-mismatched translator cascades at the same concentrations previously used.

Based on the results presented in Figure 13, it appears that the sentinel mismatch strategy accomplished its goal of increasing the forward rate of the $X1 \rightarrow Y1$ translator reaction, without significantly increasing rates of leak. At each of the triggering concentrations, even considering the variance across replicates, the reaction halftimes of the mismatched cascades were clearly lower than those in the non-mismatched cascades. This suggests that the effective rate constants of reactions when using the sentinel mismatch strategy is larger than in the non-mismatched case. Additional experiments are required to accurately quantify the effect of this systematic strategy.

5 Conclusion

We have introduced, analyzed, and given preliminary experimental validation for a systematic mismatch strategy compatible with *leakless* DNA strand displacement cascades. By promoting desired reactions and simultaneously discouraging spurious and unproductive invasions, even at high concentrations, the combination of domain and sequence level strategies unlocks a thermodynamic landscape conducive to fast, robust circuits. If this strategy proves effective broadly, as we expect, it may provide a new design standard for effective DNA strand displacement reactions and networks. The additional driving force introduced by the systematic mismatches in a toehold exchange reaction is greater than a comparable toehold-mediated strand displacement reaction without the strategy. Future experimental work will include characterization of these thermodynamic and additional kinetic differences. While we focused our strategy in the context of leakless translators it is applicable to any system with redundancy of domains between signals in a common cascade; this includes proposals of leakless architectures capable of implementing arbitrary chemical reaction networks [18]. The early success of this approach motivates the need for algorithms and design software capable of elucidating sequence level modifications that best optimize energy barriers of concern in the context of an entire strand displacement system. We presented a naive algorithm that can find an optimal design for a particular cascade, but it is unlikely to remain tractable when considering global constraints across a large system. A natural question is how to best optimize these sequence level modifications in order to improve overall kinetic rates of desired reactions without increasing undesirable reaction rates beyond a user-defined value in an overall system. Finally, what other strategies can be developed to improve strand displacement systems when combining systematic domain and sequence level design principles?

References

1 Vadim V Demidov, Michael V Yavnilovich, Boris P Belotserkovskii, Maxim D Frank-Kamenetskii, and Peter E Nielsen. Kinetics and mechanism of polyamide ("peptide") nucleic acid binding to duplex DNA. *Proceedings of the National Academy of Sciences*, 92(7):2637–2641, 1995.

2 Robert M Dirks, Justin S Bois, Joseph M Schaeffer, Erik Winfree, and Niles A Pierce. Thermodynamic analysis of interacting nucleic acid strands. *SIAM review*, 49(1):65–88, 2007.

3 MS Ellwood, M Collins, EF Fritsch, JI Williams, SE Diamond, and JG Brewen. Strand displacement applied to assays with nucleic acid probes. *Clinical chemistry*, 32(9):1631–1636, 1986.

4 Mark E Fornace, Nicholas J Porubsky, and Niles A Pierce. A unified dynamic programming framework for the analysis of interacting nucleic acid strands: Enhanced models, scalability, and speed. *ACS Synthetic Biology*, 9(10):2665–2678, 2020.

5 Natalie EC Haley, Thomas E Ouldridge, Ismael Mullor Ruiz, Alessandro Geraldini, Ard A Louis, Jonathan Bath, and Andrew J Turberfield. Design of hidden thermodynamic driving for non-equilibrium systems via mismatch elimination during DNA strand displacement. *Nature communications*, 11(1):1–11, 2020.

6 Patrick Irmisch, Thomas E Ouldridge, and Ralf Seidel. Modeling DNA-strand displacement reactions in the presence of base-pair mismatches. *Journal of the American Chemical Society*, 142(26):11451–11463, 2020.

7 Yu Sherry Jiang, Sanchita Bhadra, Bingling Li, and Andrew D Ellington. Mismatches improve the performance of strand-displacement nucleic acid circuits. *Angewandte Chemie*, 126(7):1876–1879, 2014.

8 Dmitriy A Khodakov, Anastasia S Khodakova, David M Huang, Adrian Linacre, and Amanda V Ellis. Protected DNA strand displacement for enhanced single nucleotide discrimination in double-stranded DNA. *Scientific reports*, 5(1):1–8, 2015.

9 Aaron Klug. Rosalind franklin and the discovery of the structure of DNA. *Nature*, 219(5156):808–810, 1968.

10 Hao Liu, Fan Hong, Francesca Smith, John Goertz, Thomas Ouldridge, Molly M. Stevens, Hao Yan, and Petr Šulc. Kinetics of RNA and RNA:DNA hybrid strand displacement. *ACS Synthetic Biology*, 10:3066–3073, November 2021.

11 Drew Lysne, Kailee Jones, Alma Stosius, Tim Hachigian, Jeunghoon Lee, and Elton Graugnard. Availability-driven design of hairpin fuels and small interfering strands for leakage reduction in autocatalytic networks. *The Journal of Physical Chemistry B*, 124(16):3326–3335, 2020.

12 Robert RF Machinek, Thomas E Ouldridge, Natalie EC Haley, Jonathan Bath, and Andrew J Turberfield. Programmable energy landscapes for kinetic control of DNA strand displacement. *Nature communications*, 5(1):1–9, 2014.

13 Xiaoping Olson, Shohei Kotani, Jennifer E Padilla, Natalya Hallstrom, Sara Goltry, Jeunghoon Lee, Bernard Yurke, William L Hughes, and Elton Graugnard. Availability: A metric for nucleic acid strand displacement systems. *ACS synthetic biology*, 6(1):84–93, 2017.

14 Charles M Radding. Molecular mechanisms in genetic recombination. *Annual review of genetics*, 7(1):87–111, 1973.

15 Charles M Radding. Genetic recombination: strand transfer and mismatch repair. *Annual review of biochemistry*, 47(1):847–880, 1978.

16 Niranjan Srinivas, Thomas E. Ouldridge, Petr Šulc, Joseph M. Schaeffer, Bernard Yurke, Ard A. Louis, Jonathan P.K. Doye, and Erik Winfree. On the biophysics and kinetics of toehold-mediated DNA strand displacement. *Nucleic Acids Research*, 41:10641–10658, December 2013.

17 Weiyang Tang, Weiye Zhong, Yun Tan, Guan A Wang, Feng Li, and Yizhen Liu. DNA strand displacement reaction: a powerful tool for discriminating single nucleotide variants. *DNA Nanotechnology*, pages 377–406, 2020.

18 Chris Thachuk, Erik Winfree, and David Soloveichik. Leakless DNA strand displacement systems. In *International Workshop on DNA-Based Computers*, pages 133–153. Springer, 2015.

19 Andrew J Turberfield, JC Mitchell, Bernard Yurke, Allen P Mills Jr, MI Blakey, and Friedrich C Simmel. DNA fuel for free-running nanomachines. *Physical Review Letters*, 90(11):118102, 2003.

20 Boya Wang, Chris Thachuk, Andrew D Ellington, and David Soloveichik. The design space of strand displacement cascades with toehold-size clamps. In *International Conference on DNA-Based Computers*, pages 64–81. Springer, 2017.

21 Boya Wang, Chris Thachuk, Andrew D Ellington, Erik Winfree, and David Soloveichik. Effective design principles for leakless strand displacement systems. *Proceedings of the National Academy of Sciences*, 115(52):E12182–E12191, 2018.

22 Boya Wang, Chris Thachuk, and David Soloveichik. Speed and correctness guarantees for programmable enthalpy-neutral DNA reactions. *bioRxiv*, 2022. `doi:10.1101/2022.04.13.488226`.

23 James D Watson and Francis HC Crick. Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, 1953.

24 Dongbao Yao, Tingjie Song, Xianbao Sun, Shiyan Xiao, Fujian Huang, and Haojun Liang. Integrating DNA-strand-displacement circuitry with self-assembly of spherical nucleic acids. *Journal of the American Chemical Society*, 137(44):14107–14113, 2015.

25 Bernard Yurke and Allen P Mills Jr. Using DNA to power nanostructures. *Genetic Programming and Evolvable Machines*, 4(2):111–122, 2003.

26 David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, 2011.

Universal Shape Replication via Self-Assembly with Signal-Passing Tiles

Andrew Alseth
University of Arkansas, Fayetteville, AR, USA

Daniel Hader
University of Arkansas, Fayetteville, AR, USA

Matthew J. Patitz
University of Arkansas, Fayetteville, AR, USA

Abstract

In this paper, we investigate shape-assembling power of a tile-based model of self-assembly called the Signal-Passing Tile Assembly Model (STAM). In this model, the glues that bind tiles together can be turned on and off by the binding actions of other glues via "signals". In fact, we prove our positive results in a version of the model in which it is slightly more difficult to work (where tiles are allowed to rotate) but show that they also hold in the standard STAM. Specifically, the problem we investigate is "shape replication" wherein, given a set of input assemblies of arbitrary shape, a system must construct an arbitrary number of assemblies with the same shapes and, with the exception of size-bounded junk assemblies that result from the process, no others. We provide the first fully universal shape replication result, namely a single tile set capable of performing shape replication on arbitrary sets of any 3-dimensional shapes without requiring any scaling or pre-encoded information in the input assemblies. Our result requires the input assemblies to be composed of signal-passing tiles whose glues can be deactivated to allow deconstruction of those assemblies, which we also prove is necessary by showing that there are shapes whose geometry cannot be replicated without deconstruction. Additionally, we modularize our construction to create systems capable of creating binary encodings of arbitrary shapes, and building arbitrary shapes from their encodings. Because the STAM is capable of universal computation, this then allows for arbitrary programs to be run within an STAM system, using the shape encodings as input, so that any computable transformation can be performed on the shapes.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Algorithmic self-assembly, Tile Assembly Model, shape replication

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.2

Category Extended Abstract

Related Version *Full Version*: `https://arxiv.org/abs/2206.03908`

Funding This work was supported in part by NSF grant CAREER-1553166.

1 Introduction

Artificial self-assembling systems are most often designed with the goal of building structures "from scratch". That is, they are designed so that they will start from a disorganized set of relatively simple components (often abstractly called *tiles*) that autonomously combine to form more complex target structures. This process often begins from collections of only unbound, singleton tiles, or sometimes also includes so-called *seed assemblies* which may be small (in relation to the target structure) "pre-built" assemblies that encode some information which *seeds* the growth of larger assemblies. This growth occurs as additional tiles bind to those seed assemblies according to the rules of the system, allowing them to eventually grow into the desired structures. Examples have been shown in both experimental

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham; Article No. 2; pp. 2:1–2:24
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

settings (e.g. [14, 37, 22]), as well as in the mathematical domains of abstract models (e.g. [34, 32, 7, 12, 10]). However, in the subdomain of algorithmic self-assembly, in which systems are designed so that the tile additions implicitly follow the steps of pre-designed algorithms, other goals have also been pursued. These have included, for instance, performing computations (e.g. [24, 30]), identifying input assemblies that match target shapes [31], replicating patterns on input assemblies [23, 33], and replicating (the shapes of) input assemblies [6, 26, 1, 3, 17]. In this paper, we explore the latter, particularly the theoretical limits of systems within a mathematical model of self-assembling tiles to replicate shapes.

We use the term *shape replication* to refer to the goal of designing self-assembling systems that take as input seed assemblies and which produce new assemblies that have the same shapes as those seed assemblies [1]. In order for tile-based self-assembling systems to perform shape replication, dynamics beyond those of the original abstract Tile Assembly Model (aTAM), introduced by Winfree [36] and widely studied (e.g. [34, 32, 12, 24, 5, 27, 18, 25]), are required. In the aTAM, tiles attach to the seed assembly and the assemblies which grow from it, one tile at a tile, and tile attachments are irreversible. A generalization of the aTAM, the hierarchical assembly model known as the 2-Handed Assembly Model [5, 7], allows for the combination of pairs of arbitrarily large assemblies, but it too only allows irreversible attachments. However, for shape replication, it is fundamentally important that at least some tiles are able to bind to the input assemblies to gather information about their shapes which is then used to direct the formation of the output assemblies, since binding to an assembly is the only mechanism for interacting with it. These output assemblies eventually must not be connected to the input assemblies if they are to have the same shapes as the original input assemblies. This requires that at some point tile bindings can be broken. A number of theoretical models have been proposed with mechanisms for breaking tiles apart, for example: glues with repulsive forces [29, 26], subsets of tiles which can be dissolved at given stages of assembly [1, 11], tiles which can turn glues on and off [28, 20] (a.k.a. *signal-passing tiles*), and systems where the temperature can be increased to cause bonds to break [7, 35]. Within these models, previous results have shown the power of algorithmic self-assembling systems to perform shape replication. In [6], they used glues with repulsive forces, and in [1] they used the ability to dissolve away certain types of tiles at given stages during the self-assembly process, and each showed how to replicate a large class on two-dimensional shapes. In [17], signal-passing tiles were shown to be capable of replicating arbitrary hole-free two-dimensional shapes if they are scaled up by a factor of 2. The results of [3] deal with the replication of three-dimensional shapes, and will be further discussed below.

The results of this paper are the first which provide for shape replication of all 3-dimensional shapes with no requirement for scaling those shapes. Additionally, although in [3] all three-dimensional shapes can be replicated at the small scale factor of 2, there it is necessary for the input assemblies to have relatively complex information embedded within them (in the form of Hamiltonian paths through all of their points being encoded by their glues). In our results, the input assemblies require no such embedded information. Furthermore, the model used in [3] is more complex, allowing not only for hierarchical assembly and signal-passing tiles, but also for tiles of differing shapes, and glue bindings that are flexible and thus allow for assemblies to reconfigure by folding. For the results of this paper, we have not only limited the dynamics to those of the Signal-Passing Tile Assembly Model (STAM), but have even placed an additional restriction on the model. Rather than assigning fixed orientations to tiles, in the model we use and call the STAM^R (i.e. the "STAM with rotation") tiles and assemblies are allowed to rotate. This allows us to consider an even more general, and difficult, version of the shape replication problem. Namely, the

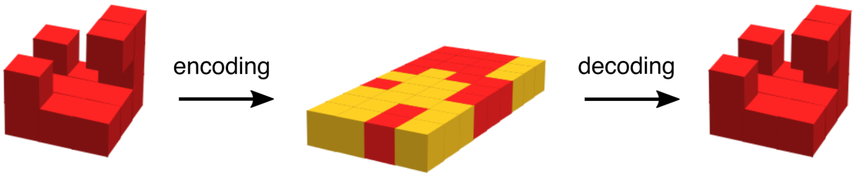


Figure 1 Schematic depiction of shape replication: (Left) An input assembly, (Middle) The assembly resulting from the encoding process which deconstructs the input assembly and encodes its shape, (Right) The assembly created by the decoding process, which uses the encoding as its input.

input assemblies in our constructions have glues of a single generic type covering their entire exteriors, and there is no distinction between a north-facing glue and an east-facing glue, for instance, as there is in the standard STAM. This makes several aspects of working with such generic input assemblies more difficult, but it is notable that our constructions need only trivial, simplifying modifications to work in the standard STAM and that our positive results thus also hold for the STAM. We show that there is a "universal shape replicator" which is a tileset in the STAM^R that can be used in conjunction with any set of generic input assemblies and will cause assemblies of every shape in the input set to be simultaneously produced in parallel. This is the first truly universal shape replicator for two or three dimensional shapes[1]. Furthermore, we break our construction into two major components, a "universal encoder" and a "universal decoder" (see Figure 1 for a depiction). The universal encoder is capable of taking generic input assemblies and creating assemblies that expose binary sequences that encode those shapes, and the universal decoder is capable of taking assemblies exposing those encodings and creating assemblies of the encoded shapes. Due to the Turing universality of this model, this also allows for the full range of all possible computational transformations to occur between the encoding and decoding, and thus enables the generation of any transformations of the shapes of the input assemblies, such as creating scaled versions or complementary shapes.

In order for our universal shape replication construction to operate, the input assemblies must be created from signal-passing tiles which are capable of turning off their glues and dissociating from the assemblies. This allows for the assemblies to be "deconstructed", and we prove that this is necessary in order to replicate arbitrary shapes, specifically those which have enclosed or narrow, curved cavities, and this is intuitively clear since otherwise there would be no way to determine which locations in the interior of an input shape are included in the shape, and which are part of an enclosed void. Our proof that it is also impossible to replicate shapes with curved, but not enclosed, cavities further exhibits the additional difficulty of working within the STAM^R model which allows tile rotations.

While our universal shape encoder, decoder, and replicator achieve the full goal of the line of research into shape replication, and also provide the ability to augment shape-building with arbitrary computational transformations, we note that the results are highly theoretical and serve more generally as an exploration of the theoretical limits of self-assembling systems. The tilesets are relatively large and require tiles with large numbers of signals, and although the input assemblies are not required to have complex information embedded within them, a trade-off that occurs compared with the results of [3] is that our constructions make use

[1] Note that while replicating two-dimensional shapes, which consist of points in a single plane, our construction will utilize three dimensions.

of a large amount of "fuel". That is, a large number of tiles are used during various phases but they are only temporary and aren't contained within the target assemblies and thus are "consumed" by the construction process. Despite the complexity of these theoretical constructions, we think that several modules and techniques developed may be of future use within other constructions (e.g. our "leader election" procedure which is guaranteed to uniquely select a single corner of an input assembly's bounding prism, to serve as a staring location for our encoding procedure within a constant number of assembly steps despite the lack of directional information provided by such an assembly), and also that these results may lead the way to similarly powerful but less complex constructions that may eventually achieve a level of being physically plausible to construct.

This paper is organized as follows. In Section 2 we provide an overview of the STAM^R and definitions. In Section 3 we state our main theorem and supporting lemmas, and discuss the constructions that prove them. In Section 4 we briefly describe some of the computational transformations that could be used to augment our constructions, and in Section 5 we prove deconstruction is necessary for shape replication of certain classes of shapes. Due to space constraints, full details can be found in [4] and the Appendix contains more details of the STAM^R and a series of subconstructions that appear throughout the constructions.

2 Definitions

In this section we provide definitions of the model used, and also for several of the terms used throughout the paper.

2.1 Overview of the STAM^R model

Here we provide only a high-level overview of the STAM^R model since it is so similar to the standard STAM model [28]. A full definition can be found in Section A.1 of the Appendix. The STAM^R is based on the STAM, which is based on the 2-Handed Assembly Model (a.k.a. Hierarchical Assembly Model), in which the fundamental components are *tiles* which have *glues* on their sides that allow them to bind together. Here, tiles are 3-dimensional unit cubes. Each glue has a string *label* and an integer *strength*, and glues can bind to each other when they are adjacent and have the same strength and have complementary labels (which we denote using "$*$", e.g. "l_1" and "l_1^*"). There is a system parameter $\tau \in \mathbb{Z}^+$ which determines the strength threshold that must be reached for tiles to bind together. Individual tiles may bind, and also any pair of *assemblies* (groups of previously bound tiles, also called *supertiles*) may bind if they can bind with at least τ strength summed across bonds and don't have overlapping tiles. Tiles and assemblies are free to rotate and don't have fixed orientations. The glues of tiles may be assigned *signals* which *fire* when those glues form bonds. These signals can turn glues on the same tile either `on` or `off`. The time between which a signal is fired and the glue it targets changes its state is nondeterministic. Each signal can fire only one time, no matter how many times its glue may form bonds. Turning glues `off` may cause previously τ-stable assemblies (ie. those where all subassemblies are connected by $\geq \tau$ strength) to break apart. When our constructions cause tiles to turn `off` glues and dissociate from assemblies, we often use the term *dissolve* since the tile is "removed" from the assembly, although it is not technically dissolved. A *system* in the STAM^R is a triple (T, S, τ) where T is a set of tile types, S is an initial state consisting of a multiset of tiles and *input* (or *seed*) assemblies and their counts (which may be infinite). τ is the binding threshold as previously discussed. An *assembly sequence* is a (possibly infinite) series of discrete steps beginning fron the initial state of a system that may include (super)tile bindings, changes in glue states, or

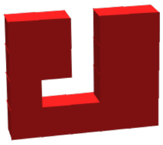


Figure 2 Example of a bent cavity, assuming that the planes on the sides into and out of the page were also filled in, leaving a single-cube-wide path into the interior of the shape.

breaking of assemblies. The *producible assemblies* of a system $\mathcal{T}$, denoted $\mathcal{A}[\mathcal{T}]$, are those which can result from the steps of some assembly sequence. The *terminal assemblies*, denoted $\mathcal{A}_\Box[\mathcal{T}]$ are those which can be produced but for which no other valid actions may occur.

▶ **Definition 1.** *Given an STAM^R system $\mathcal{T} = (T, S, \tau)$, we say that it* finitely completes *with respect to a set of terminal assemblies $\hat{\alpha}$ if and only if there exists some constant $c \in \mathbb{N}$ such that, if in the initial configuration S, each element of S was assigned count c, in every possible valid assembly sequence of $\mathcal{T}$, every element of $\hat{\alpha}$ is produced.*

A system which finitely completes with respect to assemblies $\hat{\alpha}$ is guaranteed to always produce those assemblies as long as it begins with enough copies of the (super)tiles in its initial configuration, i.e. it cannot follow any assembly sequence which would consume one or more (super)tiles needed to form those assemblies before making them.

▶ **Definition 2.** *A* shape *is a non-empty connected subset of $\mathbb{Z}^3$, i.e. a connected set of unit cubes each of which is centered at a coordinate $\vec{v} \in \mathbb{Z}^3$. A* finite shape *is a finite connected subset of $\mathbb{Z}^3$.*

In this paper, we consider shapes to be equivalent up to rotation and translation and unless stated otherwise explicitly, we will use the word *shape* to refer only to *finite shapes.*

▶ **Definition 3.** *Given a shape s, a* bounding box *is a rectangular prism in $\mathbb{Z}^3$ which completely contains s. The* minimum bounding box *is the smallest such rectangular prism.*

▶ **Definition 4.** *Given a shape s, we use the term* enclosed cavity *in s to refer to a set of connected points in $\mathbb{Z}^3$ that are not contained in s and for which no path in $\mathbb{Z}^3$ exists that does not intersect at least one point in s and gets infinitely far from all points in s.*

▶ **Definition 5.** *Given a shape s, we use the term* bent cavity *in s to refer to a set of connected points in $\mathbb{Z}^3$ contained inside of the minimum bounding box of s, b_s, but not contained within s itself, such that it includes some points which can be reached by straight lines in $\mathbb{Z}^3$ beginning from points in b_s, and some points which cannot be reached by straight lines in $\mathbb{Z}^3$ beginning from points in b_s.*

See Figure 2 for an example of a bent cavity.

▶ **Definition 6.** *We define a* shape encoding function *f_e as a function which, given as input an arbitrary shape s, returns a unique finite set E of binary strings, each unique for the shape s, such that there exists a* shape decoding function, *f_d and $f_d(e) = s$ for all $e \in E$.*

The shape encoding function we will define by construction in the proof of Lemma 14 will generate a set of binary strings for each input shape s such that each string encodes the points of the shape starting from a different reference corner and rotation of a bounding box. That can lead to up to 24 unique binary strings (for 3 rotations of each of 8 corners) for most shapes, but less for those with symmetry.

▶ **Definition 7.** *Given a shape S and a point $p = (x, y, z) \in S$, we define the* neighborhood *of p in S to be the set $S \cap \{(x+1, y, z), (x-1, y, z), (x, y+1, z), (x, y-1, z), (x, y, z+1), (x, y, z-1)\}$. We also say that neighborhoods are equivalent up to rotation, so there is 1 neighborhood containing 1 point, 2 with 2 points, 2 with 3 points, 2 with 4 points, 1 with 5 points, and 1 with 6 points.*

▶ **Definition 8.** *We define a* uniformly covered assembly *as an assembly α where every exposed side of every tile has the same strength 1 glue which is* `on`*. Additionally, if s is the shape of α, we require that for every 2 points $p, q \in s$ with the same neighborhood, a tile of the same type is located in both locations p and q in α.*

A uniformly covered assembly has the same glue all over its surface, with no glues marking special or unique locations, and has the same tile type in each location with the same neighborhood, so such an assembly can convey no information specific to particular locations, orientation, etc.

▶ **Definition 9.** *We define a* deconstructable assembly *as an assembly where (1) all neighboring tiles are bound to each other by one or more glues whose strengths sum to $\geq \tau$, and (2) each tile contains the glue(s) and signal(s) necessary to allow for all glues binding it to its neighbors to be turned* `off`*.*

In the following definitions, we will use the term *junk assembly* to refer to an assembly that is not a "desired product" of a system, but which is a small assembly composed of tiles which were used to facilitate the construction but are now terminal and cannot interact any further.

▶ **Definition 10** (Universal shape encoder). *Let $\mathcal{S}$ be the set of all finite shapes, let f_e be a shape encoding function, let $c \in \mathbb{N}$ be a constant, and let E be a tileset in the STAM^R. If, for every finite subset of shapes $S' \subset S$, there exists an STAM^R system $\mathcal{E}_{S'} = (E, \sigma_{S'}, \tau)$, where $\sigma_{S'}$ consists of infinite copies of assemblies of each shape $s \in S'$ and also infinite copies of the singleton tiles from E, such that (1) for every shape $s \in S'$ there exists at least one binary string $b_s \in f_e(s)$ and there exist infinite terminal assemblies of $\mathcal{E}_{S'}$ that contain glues in the* `on` *state on the exterior surfaces of those assemblies that encode b_s (which we refer to as an* assembly encoding *s), (2) every terminal assembly is either an assembly encoding some $s \in S'$ or a "junk assembly" whose size is bounded by c, and (3) no non-terminal assembly grows without bound, then we say that E is a* universal shape encoder *with respect to f_e.*

▶ **Definition 11** (Universal shape decoder). *Let $\mathcal{S}$ be the set of all finite shapes, let f_e be a shape encoding function, let $c \in \mathbb{N}$ be a constant, and let D be a tileset in the STAM^R. If, for every finite subset of shapes $S' \subset S$, there exists an STAM^R system $\mathcal{D}_{S'} = (D, \sigma_{S'}, \tau)$, where $\sigma_{S'}$ consists of infinite copies of assemblies each of which encode a shape $s \in S'$ with respect to f_e, and also infinite copies of the singleton tiles from D, such that (1) for every shape $s \in S'$ there exist infinite terminal assemblies of shape s, (2) every terminal assembly is either an assembly of the shape of some $s \in S'$ or a "junk assembly" whose size is bounded by c, and (3) no non-terminal assembly grows without bound, then we say that D is a* universal shape decoder *with respect to f_e.*

▶ **Definition 12** (Universal shape replicator). *Let $\mathcal{S}$ be the set of all finite shapes and let R be a tileset in the STAM^R, and let $c \in \mathbb{N}$ be a constant. If, for every finite subset of shapes $S' \subset S$, there exists an STAM^R system $\mathcal{R}_{S'} = (R, \sigma_{S'}, \tau)$, where $\sigma_{S'}$ consists of infinite copies of assemblies of each shape $s \in S'$ and also infinite copies of the singleton tiles from R, such that (1) for every shape $s \in S'$ there exist infinite terminal assemblies of shape s,* *(2) every terminal assembly is either an assembly of the shape of some $s \in S'$ or a "junk assembly" whose size is bounded by c, (3) the number of assemblies of each shape $s \in S'$ grows infinitely, and (4) no non-terminal assembly grows without bound, then we say that R is a* universal shape replicator*.*

3 3D Shape Replication

In this section, we state our main result, namely that there is a tileset in the STAM^R which is capable of replicating arbitrary shapes. This is formally stated in Theorem 13, and our proof works by providing modular constructions capable of encoding and decoding arbitrary sets of shapes which are given by Lemma 14 and Lemma 15, respectively, and then discussing how they can be combined to replicate shapes.

▶ **Theorem 13.** *There exists a tileset R in the STAM^R which is a universal shape replicator, such that for the systems using R (1) all input assemblies are uniformly covered, (2) the constant c which bounds the size of the junk assemblies equals 4, and (3) they finitely complete with respect to a set of terminal assemblies with the same shapes as the input assemblies.*

▶ **Lemma 14.** *There exist a shape encoding function f_e, and a tileset E in the STAM^R which is a universal shape encoder with respect to f_e, such that for the systems using E (1) all input assemblies are uniformly covered, (2) the constant c which bounds the size of the junk assemblies equals 4, and (3) they finitely complete with respect to a set of terminal assemblies which encode the shapes of the input assemblies.*

▶ **Lemma 15.** *There exist a shape decoding function f_d, and a tileset D in the STAM^R which is a universal shape decoder with respect to f_d, such that for the systems using D (1) the constant c which bounds the size of the junk assemblies equals 3 and (2) they finitely complete with respect to a set of terminal assemblies with the same shapes as those encoded by the input assemblies.*

3.1 Universal shape encoding

Here we describe the process by which a set of arbitrary shapes $S = \{s_1, \ldots, s_n\}$ can be encoded in the STAM^R using a universal shape encoding tileset E. We define our STAM^R system $\mathcal{E}_S$ to be the triple $(E, \Sigma_S, \tau = 2)$ where Σ_S is our initial system state consisting of all tiles in E, each with an infinite count, and additionally consists of a set $A = \{\alpha_1, \ldots, \alpha_n\}$ of uniformly covered, deconstructable assemblies such that the shape of α_i is s_i for $i = 1, \ldots, n$. The assemblies of A are called our *shape assemblies* and are made only of tiles from a fixed subset of E called *shape tiles*. Note that the glues and signals defined in these shape tiles are not used to encode any information regarding the structure of our shape assemblies; any shape specific information is inferred during the encoding process and the shape tiles simply contain the necessary glues and signals to perform basic tasks required for the encoding process, none of which are specific to any particular part of the shape assemblies.

The encoding process described below can largely be broken down into 3 steps. First, a bounding box is constructed around the shape assemblies using special tiles which are distinct from the shape tiles. Then, one of the corners of the box is elected non-deterministically to be the *leader corner* to provide an origin point which will represent the first tile location of our encoding. Finally, from the leader corner, the shape will be disassembled tile-by-tile during which an encoding assembly will be constructed, recording for each disassembled tile whether it is part of the shape or not (i.e. a "filler" tile used to assist the construction).

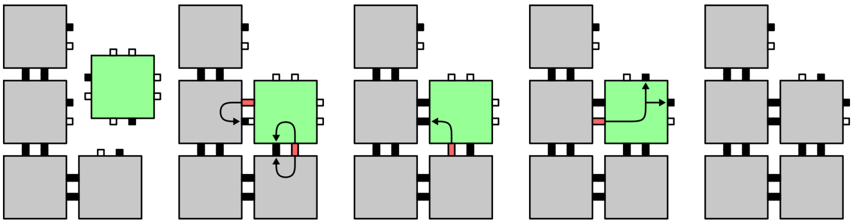


Figure 3 Filler tile binding to a concave site. Once a filler tile attaches cooperatively, signals activate glues on the filler tile and adjacent tiles. These glues ensure that the filler tile is attached with strength 2 on all sides. These glues are activated sequentially and once both are in the `on` state, signals activate output glues on all other sides of the filler tile. Once these signals activate, the supertile has 1 fewer concave site and the filler tile behaves as though it is just another tile on the supertile. While depicted in 2D for clarity, this occurs in 3D during our construction, but the idea is the same.

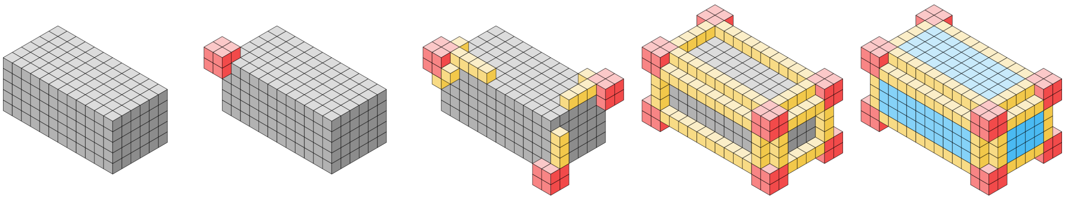


Figure 4 Growth of the inner shell around a bounding box (illustrated in gray). Growth begins by the attachment of corner gadgets (red). Cooperative binding with the corner gadgets and bounding box allow edge tiles to attach (yellow). Cooperation between the edge tiles and the bounding box then allows filler verification tiles (blue) to grow which are used to fill in the faces of the inner shell. The process by which these verification tiles bind to the bounding box ensures that there are no gaps or protrusions on the bounding box surface.

3.1.1 Bounding box assembly construction and verification

The first step in our encoding process begins by forming a bounding box assembly through the attachment of special tiles, called *filler tiles*, to a shape assembly. These filler tiles cooperatively bind to 2 diagonally adjacent tiles of our shape assembly in order to fill out any concave portions. (See Figure 3.) When a filler tile attaches to an assembly, signals are fired from the newly bound glues which activate additional glues between the filler tile and shape assembly. These new glues ensure that the filler tile is bound strongly on each face to the shape assembly. Glues are then activated on the other faces of the filler tile to further allow additional filler tiles to attach. Eventually, after sufficiently many filler tiles have attached, there will be no more locations in which another filler tile can attach. There are often many ways in which this can occur for any shape assembly, but the resulting bounding box assembly will always be a minimal bounding box of our shape. It should be noted that it's possible that not every location within the bounding box is filled. This can occur if the original shape had enclosed cavities, but can also occur because the attachment of filler tiles can create additional cavities as they attach. This is not a problem and it will always be possible for filler tiles to complete the outer surface of the bounding box.

In order to verify that the bounding box is fully formed, a shell of tiles is grown around our assembly. This shell, which we call the *inner shell*, is able to complete only if and only if the assembly is a fully formed bounding box. Figure 4 illustrates the high-level construction of the inner shell around a fully formed bounding box. Growth of the inner shell begins with the attachment of corner gadgets to our assembly. These corner gadgets then allow

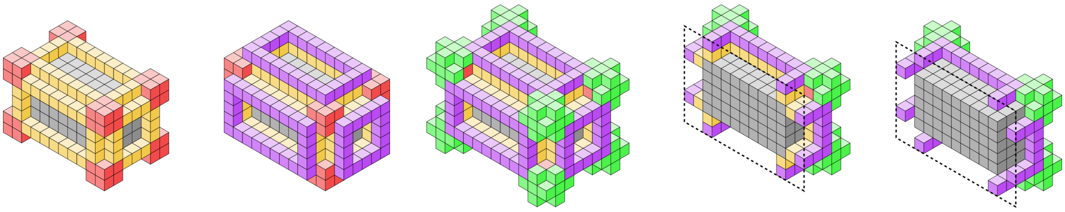


Figure 5 Once filler verification has successfully occurred on a surface of our bounding box, outer shell tiles attach to the edge tiles and corner gadgets on that surface to form a rectangle. Between the corners of these rectangles, outer corner gadgets can cooperatively bind. Once the corner gadgets have attached sufficiently to the outer shell tiles and the necessary connectivity conditions have been met, inner shell tiles are dissolved from between the outer shell and bounding box assembly. Illustrated using a cross-section view, the detachment of these tiles leaves us with a detached bounding box assembly that is too large to fit in the gaps of the outer shell, but too small to touch more than one interior corner of the outer shell simultaneously. Because of this, the bounding box assembly can then bind to an interior corner of the outer shell, but only on one corner, which is then elected leader.

for the cooperative attachment of edge tiles along the edges of the bounding box. Inside the region bounded by the edge tiles, special tiles called *filler verification tiles* cooperatively bind to the surface of the bounding box. If the surface of the bounding box assembly is not complete or contains gaps or protrusions to which additional filler tiles can bind, special gadgets called *detector gadgets* are able to bind to the erroneous edge tile or verification tile. This attachment causes a signal to fire which propagates along the erroneous tiles and causes them to deactivate their glues and detach.

3.1.2 Outer Shell Construction

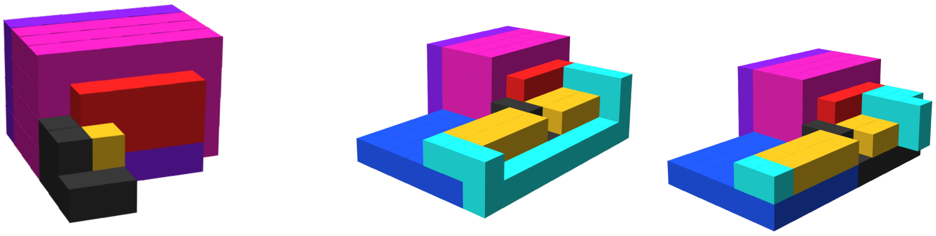
Whenever the filler verification process is completed on a surface of the bounding prism, signals activate glues on the corner gadgets of that surface which initiate the growth of an outer shell (see Figure 5). This outer shell consists of a rectangle of tiles which grows along the edge tiles of the inner shell underneath. Between these outer shell rectangles on each face of a bounding box, special corner gadgets called *outer corner gadgets* bind with 3 outer shell tiles on the corners of the assembly. Once an outer corner gadget attaches, signals are propagated along outer shell and outer edge tiles to adjacent outer corner gadgets. When enough of these signals are detected, the inner shell underneath the outer shell is dissolved and glues on the corners of the bounding box assembly called *candidate glues* are activated. These glues are complementary to glues presented on the inside corners of the outer shell. Because the inner shell dissolved, the bounding box assembly is free to move around inside of the outer shell, but since it is too big to fit through the gaps on the outer shell, it cannot escape. Additionally, since the outer shell is 1 tile bigger than the bounding box on each side, even though all corners of the two are complementary, only one of them will be able to bind. When one nondeterministically does, it is elected “leader” and the outer shell is dissolved.

3.2 Shape encoding

Following the process of leader election on a bounding box, we are presented with a single corner with unique glues exposed indicating a leader tile. Here we describe the tiles of E which allow for the universal shape encoding function f_e to be implemented on the shape contained in a bounding box. We use the term *voxels* to reference the locations of $\mathbb{Z}^3$ in the bounding box.

At a high level, the encoding of a shape $s_i \in S$ is generated by a process which visits each voxel inside of the bounding box sequentially, and transfers the information of whether the voxel is inhabited by a filler tile or a shape tile to an encoding assembly. Because leader election chooses a corner non-deterministically, there may be multiple encoding assemblies corresponding to each shape. We say that Φ_i is the finite set of encodings of shape s_i and define $\Phi = \Phi_1 \cup \cdots \cup \Phi_n$. For convenience we use ϕ_i to refer to any one encoding of s_i in Φ_i and note that the process is essentially identical for the others.

The first step in the process is for an *encoding corner gadget* to bind to the corner elected as leader, and then construct a set of helper tiles around the bounding box called the *shell*. Deconstruction is then carried out in *slices*, where each slice is the set of voxels contained in a 2D plane of the bounding box. The starting voxel contains the tile elected leader (see Figure 6a) and the orientation of the binding of the encoding corner gadget arbitrarily defines the orientation of the slices. For ease of explanation, once an orientation has been chosen by the attachment of the encoding corner gadget, we choose the x and y directions to correspond to the axes along a slice and the z direction to be the axis perpendicular to x and y into the bounding box from the leader. Specifically, each xy plane of the bounding box constitutes one slice. The end result of the encoding process is a rectangular prism assembly of height 1 where the each tile corresponds to a unique location of the bounding box in $\mathbb{Z}^3$, and whose glues represent whether or not each location contains a shape tile (represented by a 1), or empty space inhabited by a filler tile or otherwise (represented by 0). Additionally, information about the order in which tiles were deconstructed is included in ϕ_i for purposes of decoding and defining the width of a *row*. For a full description of the tiles and their functionality, see [4], and see Figure 6 for a schematic depiction of part of the process.



(a) The binding of the encoding gadget (black) allows for creation of the shell (fuchsia, purple) around the bounding box (red). Once the shell is complete, deconstruction can begin by the addition of recognizer tiles (yellow).

(b) (Left) Placement of recognizer tile allows for growth of path of tiles which extend the encoding structure and place both direction and encoding tile (Right) After tiles are placed on the encoding structure, tiles used to place the encoding tiles are dissolved into size 1 junk and the encoding process continues by placing a new recognizer in the order of encoding.

Figure 6 Tile type calculation and description of tile types used in shape decoding.

We provide a description of how the information provided by the location of tiles in a bounding box is encoded into binary values. Beginning with the origin point $(0, 0, 0)$, we read the tile type information for each tile in the first row sequentially by incrementing the x-coordinate; for example, the second tile read is in the voxel with coordinates $(1, 0, 0)$. Once all tiles in the current row have been read, we jump to the next row up. By this process of visiting every tile in a slice in a "zig-zag" pattern, we are able to encode the information regarding any slice of a bounding box sequentially. Figure 7 demonstrates the encoding of a $3 \times 4 \times 5$ $(x \times y \times z)$ bounding box.

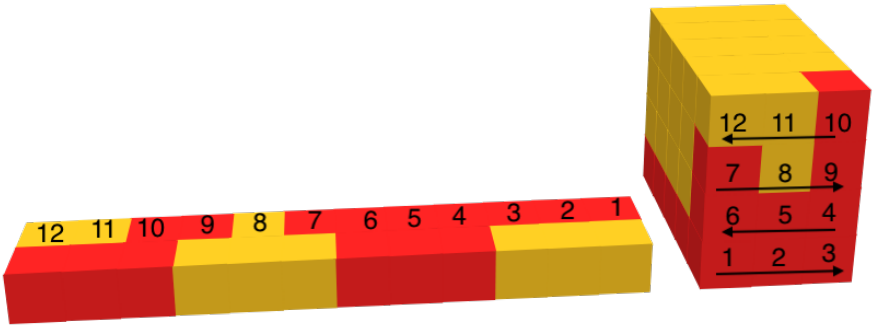


Figure 7 (Right) An example 3 × 4 × 5 shape, (Left) The first two rows of its encoding assembly. The first (closest) row encodes the direction followed for each row of a slice, and the second row encodes the presence of a shape tile or filler tile in each location. Yellow tiles represent '0', red tiles represent "1". Shape tiles and '+' direction growth are encoded as 0, fill tiles and '-' are encoded as 1. The encodings of additional slices only need a single row each, since the growth direction is shared across rows of consecutive slices.

The very first row of the encoding subassembly contains the information regarding the direction of the growth in our zig-zag pattern, and as a byproduct we also are able to easily retrieve the width of the rows of tiles. We compare the x values in the coordinates (x, y, z) between the first tile of a row and the last tile of a row by subtracting the x value between the two such that $\Delta x = x_{\text{last}} - x_{\text{first}}$. If a tile is contained in a row where $\Delta x > 0$ we denote this growth in the positive ('+') direction. Alternatively, if $\Delta x < 0$ we denote this growth in the negative ('-') direction.

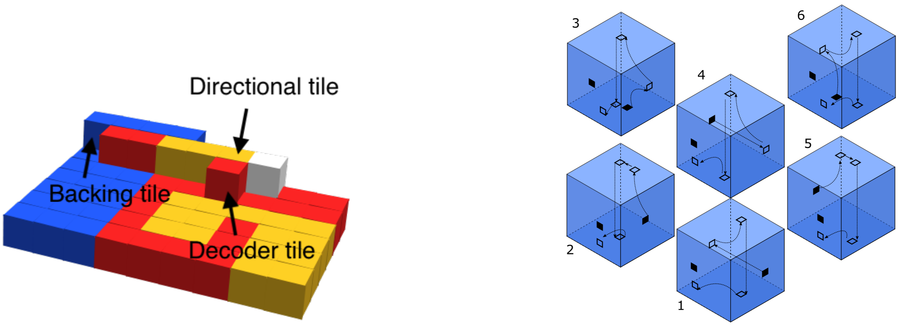
The STAM^R system $\mathcal{E}_S = (E, \Sigma_S, \tau = 2)$ generates the set of encoding assemblies Φ with a junk size of 4. $\mathcal{E}_S$ finitely completes, as each of the sub-constructions to carry out the encoding f_e require a finite number of steps (and thus, finite tile count) to complete. The final property which must hold is that regardless of the number of distinct shapes of input assemblies, the shapes of all will be correctly replicated. By our construction, there are never exposed glues on the surfaces of any pair of assemblies that each contain an input assembly that would allow them to bind to each other. Since junk assemblies produced by any assembly sequence are also unable to negatively interact with other assemblies, a system whose input assemblies have multiple shapes will behave simply as the union of individual systems which each have one input assembly shape, creating terminal assemblies of all of (and only) the correct shapes. Thus we prove Lemma 14 (with full details in [4]).

3.3 Shape Decoding

We now describe the tileset D which functions as a universal shape decoder. The STAM^R system for shape decoding is defined as $\mathcal{D}_\Phi = \{D, \Sigma_\Phi, \tau = 2\}$. Σ_Φ includes infinite copies of the tiles of D and the set Φ of encoding structures generated from $\mathcal{E}_S$. The shape decoding process and tile types required can be broken into 3 main sets of tiles. We describe the process for a single $\phi \in \Phi$ and note that the process proceeds identically for each encoding simultaneously. First, base tiles initiate the decoding process by binding to ϕ at a unique starting location. Base tiles also provide the remaining tile types to build an assembly which is guaranteed to be connected by at least strength 2 to the encoding structure. Second, we construct the shape and filler tiles and describe how encoded information allows for an assembly sequence of shape tiles guaranteed to be connected to their neighbors in the encoded shape. Third, we have a set of tiles which read the encoding of filler and shape tiles (decoder tiles), and allow for the sequential placement of shape and filler tiles based on their location in the encoding of a shape. In the tileset D, we use a decoding process which places tiles in the exact same order as the encoding process built the encoding assembly ϕ as described in Figure 7. For a full description of the tiles and their functionality, see [4].

3.3.1 Retrieving Encoded Information

Two pieces of information are explicitly encoded in ϕ. The bulk of the tiles in the encoding correspond to identifying if a location in a shape corresponds to empty space, or a tile of the shape. The second piece of information provided in the first row of the encoding is the the direction of growth; this can be utilized in two manners. First, the direction of growth provides to the system the types of tiles to be utilized to reach the point encoded, as growth processes vary significantly between “+” direction growth (encoded as a 0) and “-” direction growth (encoded as a “1”). Secondly, when the direction of growth encoded changes from 0 to 1 or 1 to 0, this indicates to the system when a tile is to be placed into a new row. This information is required to ensure that we can grow a slice such that each tile is guaranteed to be connected to its neighbor, and so tile faces are assigned the appropriate glues. Figure 8a demonstrates the tile bindings allowing information to be retrieved from an encoding.



(a) An example of the information which is gathered from the encoding structure. The directional tile gathers information regarding the growth type of tile location encoded. The direction change detector gadget (white) which detects that growth type ‘0’ shifts to growth type ‘1’, indicating a change of row and necessitating a direction change tile. The decoder tile, once glues are available to cooperatively bind to the encoding structure and the directional tile, determines that the tile in the current location is a shape tile. Backing tiles allow for the sensing of the first row.

(b) An example of normal row growth and direction change tiles used by the decoding process to build a slice. These tile types map to both shape and fill tiles. (1) and (4) are standard row growth tiles for ‘+’ and ‘-’ direction growth, respectively. (2) and (5) are row end tiles for ‘+’ and ‘-’ direction growth; they open cooperative binding sites which allow for tiles (3) and (6) to bind and change the direction of growth. Signal activation arrows demonstrate the order in which faces of shape tiles are determined to be either bound to a neighboring shape tile or have a fill tile adjacent to the face.

Figure 8 Tile type calculation and description of tile types used in shape decoding.

3.3.2 Proof of Universal Shape Decoding Correctness

We summarize the decoding process and show that during this process, the shapes which were encoded in the set of input encoding assemblies Φ are correctly assembled. We first consider the decoding process of a single encoding assembly $\phi \in \Phi$ and note that a similar process happens for all encoding assemblies simultaneously without interfering one another.

Our decoding process begins by building a base of tiles connected to ϕ. This base holds the shape as it is being constructed and is used to help ensure the connectivity of the shape as it is being constructed. The decoding process is performed in iterations, where during each iteration a row of ϕ is scanned tile-by-tile and a corresponding 2D slice of the shape is constructed. Each slice is constructed starting from the bottom (smallest y coordinate) to the top (largest y coordinate), with tiles attaching in a zig-zag manner, as illustrated in Figure 7. Each slice of the assembled shape corresponds to a unique z coordinate so for convenience we call the slice whose z coordinate is i, σ_i. As each slice is assembled, tiles are placed in each location of the slice, even those locations that will not be part of the final shape, though these will be removed during the assembly of the next slice.

The first slice σ_1 can be assembled naively, but during the assembly of each following slice, tiles which will not be part of the final shape on the previous slice must be removed. This is done as follows. Suppose that slice σ_i $(i > 1)$ is currently being assembled. Before a tile t_i is placed in a location (x, y, i), a gadget is used to determine the type of the tile (see Figure 8b for examples of the 6 main tile types) t_{i-1} at location $(x, y, i-1)$ (i.e. the tile with the same x and y coordinates on the previous slice). If this t_{i-1} is part of the final shape, then t_i is placed and signals are used to activate strength 2 glues between t_i and t_{i-1}; otherwise, if t_{i-1} is not part of the final shape, it is removed before t_i is placed. Regardless of the type of tile t_{i-1}, when t_i is placed, glues are activated which connect t_i to all adjacent tiles on the same slice. Once the final slice is assembled, a final zig-zag pass is made in the next z coordinate to remove all tiles from the last slice which are not part of the final shape.

It is also important to note that the base, on which the shape is being assembled, also forms a ceiling above the slices being assembled. This ceiling helps ensure that tiles on the top row of each slice are able to remain attached to the assembly during construction. It should be clear that during this decoding process (1) each tile that belongs to the final shape is placed in its correct location, and (2) that those tiles of a slice which are not part of the final shape will be removed from the assembly during the assembly of the next slice; However, because tiles are removed during the process, we must show that none of these removals can cause parts of the assembly to unintentionally detach. We state this as Lemma 16. The proof of this can be found in [4]

▶ **Lemma 16.** *Let ϕ be an encoding assembly which encodes the shape s. During the decoding process above, as slice σ_i $(i > 1)$ is being assembled, no tile in slices $\sigma_1, \ldots, \sigma_{i-1}$ which are part of the final shape assembly can detach.*

Given the set of input encoding structures Φ, the STAM^R system $\mathcal{D}_\Phi = \{D, \Sigma_\Phi, \tau = 2\}$ produces a set of terminal supertiles $S = \{s_1, \ldots, s_n\}$ in parallel with a maximum junk size of 3. $\mathcal{D}_\Phi$ finitely completes, as for the production of the set of shapes $s \in S$ from input encoding structures Φ a finite number of tiles are required for each encoding structure to produce a terminal assembly. We can guarantee this as each encoding produces a single terminal shape, as the encoding of the shape dissolves into size 1 junk after the terminal shape has decoded. By our construction, there are never exposed glues on the surfaces of any pair of assemblies that each contain an input encoding that would allow them to bind to each other. Since junk assemblies produced by any assembly sequence are also unable to interact with other assemblies, a system whose input assemblies have multiple shapes will behave simply as the union of individual systems which each have one input assembly shape, creating terminal assemblies of all of (and only) the correct shapes. Thus we prove Lemma 15.

Since we have shown the existence of universal encoding and universal decoding tilesets, we have the basis to demonstrate a universal shape replicator and prove Theorem 13. Due to the modularity of the constructions of Lemmas 14 and 15, at a high level we simply generate a new tileset $R = E \cup D$. A few minor changes must occur to satisfy Definition 12; the full proof can be found in [4].

4 Beyond Shape Replication

The constructions used to prove Theorem 13 were intentionally broken into separate, modular constructions proving Lemmas 14 and 15 and thus providing a universal shape encoder and a universal shape decoder. This is not only useful for proving their correctness, but also for allowing for computational transformations to be performed on the encodings of input shapes in order to instead produce output shapes based on those transformations. Like even the much simpler aTAM, the STAM (and STAM^R) are Turing universal, meaning any arbitrary computer program can be executed by systems in these models. Thus, given any program that can perform a computational transformation of the points of a shape and output points of another shape, tiles that execute that program (for instance, by simulating an arbitrary Turing machine in standard ways, e.g. [30, 24]) can receive as input the binary encodings of arbitrary shapes (after their creation by the universal shape encoder), transform them in any algorithmic manner, and then assemblies of the shapes output by those transformations can be produced (using the universal shape decoder).

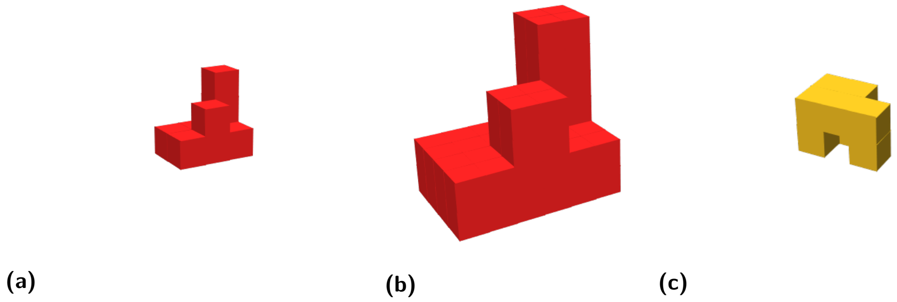


Figure 9 (a) An example shape, (b) The same shape at scale factor 2, (c) A shape which is complementary to the top surface of the shape in (a).

Due to space constraints, we do not go into great detail about the opportunities that such constructions provide. Instead, we mention just a few of the possibilities (and depict some in Figure 9) while noting that the possibilities are technically infinite:

1. Scaled shapes: a system could be designed to produce assemblies that have the shapes of input assemblies scaled by either a built-in constant factor (including negative, to shrink the shapes), or instead with another type of input assembly that specifies the scaling factor, allowing for a "universal shape scaler".
2. Inverse shapes: a system could be designed to produce assemblies that have the inverse, i.e. complementary, shapes of input assemblies (assuming the complements are connected, and restricting to a bounding box size since the complement of any finite shape is infinite).
3. Pattern matching: a system could be designed to inspect input assembly shapes for specific patterns and to either produce assemblies that signal the presence of a target pattern, or instead assemblies that are complementary to, and can bind to, the surfaces of assemblies containing those patterns.

Although such constructions are highly theoretical and quite complex, and thus unlikely in their current forms to be practically implementable, they provide a mathematical foundation for the construction of complex, dynamic systems that mimic biological systems. One possible example is an "artificial immune system" capable of inspecting surfaces, detecting those which match (or fail to match) specific patterns, and creating assemblies capable of binding to those deemed to be foreign, harmful, or otherwise targeted. As mentioned, there are infinite possibilities.

5 Impossibility of Shape Replication Without Deconstruction

In this section, we prove that in order for a system in the STAM^R to encode and/or replicate shapes which have enclosed or bent cavities (see Definitions 4 and 5), the input assemblies must have the potential for tiles to be removed. To do so, we first utilize a theorem from [2].

▶ **Theorem 4** (from [2]). *Let U be an STAM* tileset such that for an arbitrary 3D shape S, the STAM* system $\mathcal{T} = (U, \sigma_S, \tau)$ with $\mathrm{dom}\, \sigma_S = S$, $\mathcal{T}$ is a shape self-replicator for S and σ_S is non-porous. Then, for any $r \in \mathbb{N}$, there exists a shape S such that $\mathcal{T}$ must remove at least r tiles from the seed assembly σ_S.*

Theorem 4 from [2] applies to the STAM* (see Section A.3). However, the STAM^R is simply a restricted version of the STAM* which only allows tiles to be a single shape, that of a unit cube, and does not allow flexible glues. Since all assemblies in the STAM^R are non-porous (i.e. free tiles cannot pass through the tiles of an assembly or the gaps between bound tiles) and the STAM^R has more restrictive dynamics than the STAM*, the proof of this result, which shows the impossibility of self-replicating assemblies with enclosed cavities without removing tiles, suffices to prove the following corollary (stated using the terminology of this paper) as well.[2] Note that this proof holds even if an input assembly is not uniformly covered, but its glues are instead allowed to encode information about the shape.

▶ **Corollary 17.** *There exist neither a universal shape encoder nor a universal shape replicator in the STAM^R for the class of shapes with enclosed cavities whose assemblies are not deconstructable.*

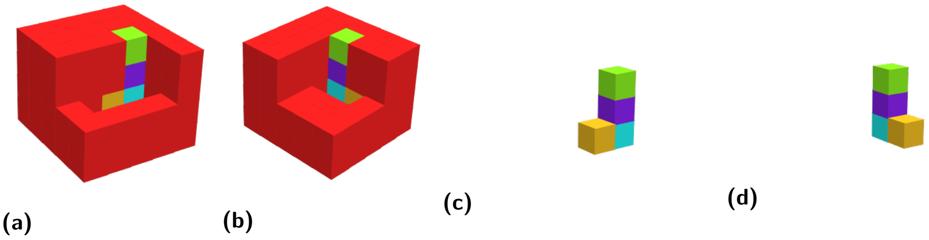


Figure 10 (a) and (b) Partial depictions of a pair of shapes which cannot be correctly encoded/replicated without a deconstructable input assembly. Each consists of a $5 \times 5 \times 4$ cube with a 4-cube-long bent cavity. For each, the green, purple, blue, and yellow locations indicate the empty locations that make the bent cavity. The rest of the $5 \times 5 \times 4$ cube locations would be filled in with red cubes (some have been omitted to make the cavity locations visible). (c) and (d) The shapes of assemblies that could grow into the bent cavities.

Our next theorem deals with shapes having bent cavities.

[2] The proof can be found in [2], and we omit duplicating it here due to space constraints.

▶ **Theorem 18.** *There exist neither a universal shape encoder nor a universal shape replicator in the STAM^R for the class of shapes with bent cavities whose input assemblies are uniformly covered but are not deconstructable.*

We prove Theorem 18 by contradiction. Therefore, let f_e be a shape encoding function and assume E is a universal shape encoder with respect to f_e, and let c be the constant value which bounds the size of the junk assemblies. (Nearly identical arguments will hold for a universal shape replicator.) Define the shapes s_1 and s_2 as shown in Figures 10a and 10b, i.e. each is a $5 \times 5 \times 4$ cube with a bent cavity that goes into the cube to a depth of 3, then turns one of two directions for each. Note importantly that the well is offset from the center of the cube such that s_1 and s_2 are not rotationally equivalent. Since E is assumed to be a universal shape encoder, there must exist two STAM^R systems $\mathcal{E}_1 = (E, \sigma_1, \tau)$ and $\mathcal{E}_2 = (E, \sigma_2, \tau)$, where σ_1 consists of infinite copies of tiles from E and infinite copies of uniformly covered assemblies in the shape of s_1, and σ_2 consists of infinite copies of tiles from E and infinite copies of uniformly covered assemblies in the shape of s_2 as follows.

$\mathcal{E}_1$ must produce terminal assemblies which encode shape s_1 but must not produce terminal assemblies which encode shape s_2, since no assembly of shape s_2 is included in its input assemblies. Similarly, $\mathcal{E}_2$ must produce terminal assemblies which encode shape s_2 but not s_1. Let $\vec{\alpha}$ be an assembly sequence in $\mathcal{E}_1$ which results in a terminal assembly encoding shape s_1. We now show that every action of $\vec{\alpha}$ must be valid, in the same ordering, in $\mathcal{E}_2$ but using an input assembly of shape s_2. This is because the exact same glues will be exposed by the input assemblies of shapes s_1 and s_2 in the same relative locations with the slight difference of relative rotations of the innermost locations of the bent cavities of each from the adjacent cavity locations. Assuming that, in $\vec{\alpha}$, tiles attach into all locations of the bent cavity (if only the location shown in yellow remains empty the same argument will hold, and if both the locations shown in yellow and blue remain empty then there is absolutely no difference in any aspect of the assembly sequence in $\mathcal{E}_2$ and the argument immediately holds), this results only in the relative orientations of at most the bottom two tiles being turned 90 degrees relative to the tile immediately above them (i.e. the tile in the purple location in Figure 10). Since tiles in the STAM^R are rotatable, with no distinction for directions, there is no mechanism for tiles in the purple locations of assemblies shown in Figures 10c and 10d from distinguishing from each other (via tile types, glues, or signals). Tiles of the same types which bind into those locations in $\vec{\alpha}$ must also be able to do so in the assembly sequence of $\mathcal{E}_2$ using the exact same glues and firing the exact same signals (if any). Thus $\vec{\alpha}$ must be a valid assembly sequence in $\mathcal{E}_2$ as well. This means that an assembly encoding the shape of s_1 is also created as a terminal assembly in $\mathcal{E}_2$. Note that if the constant c is greater than the size of the shapes s_1 and s_2 (i.e. $5 * 5 * 4 - 4 = 96$), then we can simply increase their dimensions until they are larger than c (but still contain the same bent cavities) and the argument still holds and the incorrectly produced assemblies cannot be considered "junk" assemblies. This is a contradiction that E is a universal shape encoder with respect to f_e and constant c. Since no assumptions were made about E other than it being a universal shape encoder, no such E can exist. By slightly altering the argument for a universal shape replicator R, and generating terminal assemblies of shapes s_1 and s_2 (rather than their encodings), the same argument holds to show no universal shape replicator exists. Thus Theorem 18 is proven.

References

1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Martin L. Demaine, Robin Flatland, Scott D. Kominers, and Robert T. Schweller. Shape replication through self-assembly and RNAse enzymes. In *SODA 2010: Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1045–1064, Austin, Texas, 2010. Society for Industrial and Applied Mathematics.

2 Andrew Alseth, Daniel Hader, and Matthew J. Patitz. Self-replication via tile self-assembly. Technical Report 2105.02914, Computing Research Repository, 2021. arXiv:2105.02914.

3 Andrew Alseth, Daniel Hader, and Matthew J. Patitz. Self-Replication via Tile Self-Assembly (Extended Abstract). In Matthew R. Lakin and Petr Šulc, editors, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, volume 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:22, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DNA.27.3.

4 Andrew Alseth, Daniel Hader, and Matthew J. Patitz. Universal shape replication via self-assembly with signal-passing tiles, 2022. doi:10.48550/ARXIV.2206.03908.

5 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors): Self-assembly in the 2HAM vs. aTAM. In Natacha Portier and Thomas Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 172–184. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

6 Cameron Chalk, Erik D. Demaine, Martin L. Demaine, Eric Martinez, Robert Schweller, Luis Vega, and Tim Wylie. Universal shape replicators via Self-Assembly with Attractive and Repulsive Forces. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 225–238. Society for Industrial and Applied Mathematics, January 2017. doi:10.1137/1.9781611974782.15.

7 Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.

8 E. D. Demaine, M. L. Demaine, S. P. Fekete, M. J. Patitz, R. T. Schweller, A. Winslow, and D. Woods. One tile to rule them all: Simulating any tile assembly system with a single universal tile. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, IT University of Copenhagen, Denmark, July 8-11, 2014, volume 8572 of *LNCS*, pages 368–379, 2014.

9 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008. doi:10.1007/s11047-008-9073-0.

10 Erik D. Demaine, Matthew J. Patitz, Trent A. Rogers, Robert T. Schweller, Scott M. Summers, and Damien Woods. The two-handed assembly model is not intrinsically universal. In *40th International Colloquium on Automata, Languages and Programming, ICALP 2013, Riga, Latvia, July 8-12, 2013*, Lecture Notes in Computer Science. Springer, 2013.

11 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Self-Assembly of Arbitrary Shapes Using RNAse Enzymes: Meeting the Kolmogorov Bound with Small Scale Factor (extended abstract). In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 201–212, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2011.201.

12 David Doty, Jack H. Lutz, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, FOCS 2012, pages 302–310, 2012.

13 Jérôme Durand-Lose, Jacob Hendricks, Matthew J. Patitz, Ian Perkins, and Michael Sharp. Self-assembly of 3-D structures using 2-D folding tiles. In David Doty and Hendrik Dietz, editors, *DNA Computing and Molecular Programming - 24th International Conference, DNA 24, Jinan, China, October 8-12, 2018, Proceedings*, volume 11145 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 2018.

14 Constantine Glen Evans. *Crystals that count! Physical principles and experimental investigations of DNA tile self-assembly.* PhD thesis, California Institute of Technology, 2014.

15 Tyler Fochtman, Jacob Hendricks, Jennifer E. Padilla, Matthew J. Patitz, and Trent A. Rogers. Signal transmission across tile assemblies: 3D static tiles simulate active self-assembly by 2D signal-passing tiles. *Natural Computing*, 14(2):251–264, 2015.

16 Jacob Hendricks, Meagan Olsen, Matthew J. Patitz, Trent A. Rogers, and Hadley Thomas. Hierarchical self-assembly of fractals with signal-passing tiles. Submit to Natrual Computing.

17 Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Replication of arbitrary hole-free shapes via self-assembly with signal-passing tiles. In Cristian S. Calude and Michael J. Dinneen, editors, *Unconventional Computation and Natural Computation - 14th International Conference, UCNC 2015, Auckland, New Zealand, August 30 - September 3, 2015, Proceedings*, volume 9252 of *Lecture Notes in Computer Science*, pages 202–214. Springer, 2015. doi:10.1007/978-3-319-21819-9_15.

18 Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Universal simulation of directed systems in the abstract tile assembly model requires undirectedness. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016), New Brunswick, New Jersey, USA* October 9-11, 2016, pages 800–809, 2016.

19 Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Reflections on tiles (in self-assembly). *Natural Computing*, 16(2):295–316, 2017. doi:10.1007/s11047-017-9617-2.

20 Nataša Jonoska and Daria Karpenko. Active tile self-assembly, Part 1: Universality at temperature 1. *International Journal of Foundations of Computer Science*, 25(02):141–163, 2014. doi:10.1142/S0129054114500087.

21 Nataša Jonoska and Daria Karpenko. Active tile self-assembly, Part 2: Self-similar structures and structural recursion. *International Journal of Foundations of Computer Science*, 25(02):165–194, 2014. doi:10.1142/S0129054114500099.

22 Yonggang Ke, Luvena L Ong, William M Shih, and Peng Yin. Three-dimensional structures self-assembled from DNA bricks. *Science*, 338(6111):1177–1183, 2012.

23 Alexandra Keenan, Robert T. Schweller, and Xingsi Zhong. Exponential replication of patterns in the signal tile assembly model. In David Soloveichik and Bernard Yurke, editors, *DNA*, volume 8141 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2013. doi:10.1007/978-3-319-01928-4_9.

24 James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers. Computability and complexity in self-assembly. *Theory Comput. Syst.*, 48(3):617–647, 2011. doi:10.1007/s00224-010-9252-0.

25 James I. Lathrop, Jack H. Lutz, and Scott M. Summers. Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science*, 410:384–405, 2009.

26 Austin Luchsinger, Robert Schweller, and Tim Wylie. Self-assembly of shapes at constant scale using repulsive forces. *Natural Computing*, August 2018. doi:10.1007/s11047-018-9707-9.

27 Pierre-Étienne Meunier, Damien Regnault, and Damien Woods. The program-size complexity of self-assembled paths. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 727–737. ACM, 2020. doi:10.1145/3357713.3384263.

28 Jennifer E. Padilla, Matthew J. Patitz, Robert T. Schweller, Nadrian C. Seeman, Scott M. Summers, and Xingsi Zhong. Asynchronous signal passing for tile self-assembly: Fuel efficient computation and efficient assembly of shapes. *International Journal of Foundations of Computer Science*, 25(4):459–488, 2014.

29 Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Exact shapes and Turing universality at temperature 1 with a single negative glue. In Luca Cardelli and William M. Shih, editors, *DNA Computing and Molecular Programming - 17th International Conference, DNA 17, Pasadena, CA, USA, September 19-23, 2011. Proceedings*, volume 6937 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2011.

30 Matthew J. Patitz and Scott M. Summers. Self-assembly of decidable sets. *Natural Computing*, 10(2):853–877, 2011. doi:10.1007/s11047-010-9218-9.

31 Matthew J. Patitz and Scott M. Summers. Identifying shapes using self-assembly. *Algorithmica*, 64(3):481–510, 2012.

32 Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstract). In *STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing*, pages 459–468, Portland, Oregon, United States, 2000. ACM.

33 Rebecca Schulman, Bernard Yurke, and Erik Winfree. Robust self-replication of combinatorial information via crystal growth and scission. *Proceedings of the National Academy of Sciences*, 109(17):6405–10, 2012. URL: `http://www.biomedsearch.com/nih/Robust-self-replication-combinatorial-information/22493232.html`.

34 David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. `doi:10.1137/S0097539704446712`.

35 Scott M. Summers. Reducing tile complexity for the self-assembly of scaled shapes through temperature programming. *Algorithmica*, 63(1-2):117–136, June 2012. `doi:10.1007/s00453-011-9522-5`.

36 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

37 Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. *Nature*, 567:366–372, 2019.

A Technical Appendix

A.1 Definition of the STAM^R model

Here we provide a definition of the model used in this paper, called the STAM^R (i.e. the "STAM with rotation"), which is based upon the 3D Signal-passing Tile Assembly Model (STAM) [15] (and similar to the model in [20, 21]). The STAM is itself based upon the 2-Handed Assembly Model (2HAM) [7, 9], also referred to as the "Hierarchical Assembly Model", which is a mathematical model of tile-based self-assembling systems in which arbitrarily large pairs of assemblies can combine to form new assemblies.

A *glue* is an ordered pair (l, s), where $l \in \Sigma^+ \cup \{s^* : s \in \Sigma^+\}$ is a non-empty string, called the *label*, over some alphabet Σ, possibly concatenated with the symbol "$*$", and $s \in \mathbb{Z}^+$ is a positive integer, called the *strength*. A glue label l is said to be *complementary* to the glue label l^*.

A *tile type* is a mapping of zero or more glues, along with *glue states* and possibly *signals*, which will be defined shortly, to the 6 faces of a unit cube. A *tile* is an instance of a tile type, and is the base component of the STAM^R. Each tile type is defined in a canonical orientation, but tiles can be in that orientation or any rotation which is orthogonal to it (i.e. they are embedded in $\mathbb{Z}^3$).

Every glue can be in one of three *glue states*: $\{\texttt{on}, \texttt{latent}, \texttt{off}\}$. If two tiles are placed next to each other, and their adjacent faces have glues $g_1 = (l, s)$ and $g_2 = (l^*, s)$, then those glues can form a *bond* whose *strength* is s. We require any copies of glues with the label l, or its complement l^*, in any given system have the same strength (e.g. it is not allowed to have one glue labeled l with strength 1 and another labeled l or l^* with strength 2).

A *signal* is a mapping from a glue g_s (the *source glue*) to an ordered pair, (g_t, s), where g_t (the *target glue*) is a glue on the same tile as g_s (possibly g_s itself) and $s \in \{\texttt{on}, \texttt{off}\}$. If and when g_s forms a bond with its complementary glue on an adjacent tile, the signal is *fired* to change the state of g_t to state s. Each glue of a tile type can be defined to have zero or more signals assigned to it. Each signal on a tile can fire at most a single time. When a glue is fired, the state of the target glue is not immediately changed, but the pair (g_t, s) is added to a queue of *pending signals* for the tile containing its glues. When a pending glue is selected for completion (in a process described below), then the state of g_t is changed to s if and only if its current state is s_0 and $(s_0, s) \in \{(\texttt{on}, \texttt{off}), (\texttt{latent}, \texttt{on}), (\texttt{latent}, \texttt{off})\}$. That is, the only valid glue state transitions are `on` to `off`, or `latent` to `on` or `off`.

A *supertile* is (the set of all translations and rotations of) a positioning of one or more connected tiles on the integer lattice $\mathbb{Z}^3$. Two adjacent tiles in a supertile can form a bond if the glues on their abutting sides are complementary and both are in the `on` state. Each supertile induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between every pair of bound tiles whose weight is the strength of the bound glues. A supertile is *τ-stable* if every cut of its binding graph cuts edges whose weights sum to at least τ. That is, the supertile is τ-stable if at least energy τ is required to separate the supertile into two parts. *Assembly* is another term for a supertile, and we use the terms interchangeably, to mean the same thing.

Each tile has a *tile state* that contains the current state of every glue as well as a (possibly empty) set of pending signals and a (possibly empty) set of completed signals. Every supertile consists of not only its set of constituent tiles, but also their tile states, and a set bonds that have formed between pairs of glues on adjacent tiles.

A system in the STAM^R is an ordered triple (T, S, τ) where T is a finite set of tiles called the *tileset*, S is a *system state* which consists of a multiset of supertiles that each have a count (possibly infinite), and $\tau \in \mathbb{Z}^+$ is the *binding threshold* (a.k.a. *temperature*) parameter of the system which specifies the minimum strength of bonds needs to hold a supertile together. In the initial state of a system, no tiles have pending signals, all pairs of adjacent glues which are both complementary and in the `on` state in all supertiles have formed bonds and any signals which would have been fired by those bonds are completed, and all distinct supertiles are assumed to start arbitrarily far from each other (i.e. none is enclosed within another). By default (and unless otherwise specified), the initial state contains an infinite count of all singleton tiles in T.

A system evolves as a (possibly infinite) series of discrete steps, called an *assembly sequence*, beginning from its initial state. Each step occurs by the random selection and execution of one of the following actions:

1. Two supertiles currently in the system, α and β, are translated and/or rotated without ever overlapping so that they can form bonds whose strengths sum to at least τ. The count of the newly formed supertile is increased by 1 in the system state and the counts of each of α and β are decreased by 1 (if they aren't ∞). In the newly created supertile, from the entire set of pairs of glues which can form bonds, a random subset whose strengths sum to $\geq \tau$ is selected and bonds formed by those glues are added to the set of bonds that have formed for that supertile. Additionally, for each glue which forms a bond, all signals for which it is a source glue, but which aren't already pending or completed, are added to the set of pending signals for its tile.
2. For any supertile currently in the system, from the set of pairs of glues which can form bonds but haven't, a glue pair is selected and a bond formed by those glues is added to the set of bonds that have formed for that supertile. Additionally, for each glue which forms that bond, all signals for which it is a source glue, but which aren't already pending or completed, are added to the set of pending signals for its tile.
3. For any supertile currently in the system, a pending signal is selected from the set of pending signals of one of its tiles. If the action specified by that signal is valid, the state of the target glue is changed to the state specified by the signal. The signal is removed

from the set of pending signals and added to the set of completed signals. If the action is not valid (i.e. the pair specifying the current state of the target glue and the desired end state is not in $\{(\texttt{on}, \texttt{off}), (\texttt{latent}, \texttt{on}), (\texttt{latent}, \texttt{off})\}$), then the signal is just removed from the pending set and added to the completed set, and there is no change to the target glue.

4. For a supertile γ currently in the system for which there exists one or more cuts of $< \tau$ (which could be the case due to one or more glues changing to the `off` state), one of those cuts is randomly selected and γ is split into two supertiles, α and β, along that cut. The count of γ in the system state is decreased by one (if it isn't ∞) and the counts of α and β are increased by one (if they aren't ∞).

Given a system $\mathcal{T} = (T, S, \tau)$, a supertile is *producible*, written as $\alpha \in \mathcal{A}[\mathcal{T}]$, if it either is contained in the initial state S or it can be formed, starting from S, by any series of the above steps. A supertile is *terminal*, written as $\alpha \in \mathcal{A}_{\Box}[\mathcal{T}]$, if it is producible and none of the above actions are possible to perform with it (and any other producible assembly, for list item 1).

Note that tiles are not allowed to diffuse through each other, and therefore a pair of combining supertiles must be able to translate and/or rotate without ever overlapping into positions for binding. It is allowed, though, for two supertiles, α and β, to translate and/or rotate into locations which are partially enclosed by another supertile γ before binding, potentially creating a new supertile, δ, which would not have been able to translate and/or rotate into that location inside γ, without overlapping γ, after forming. However, although the model allows for supertiles to assemble "inside" of others, in order to strengthen our results we do not utilize it for the constructions of our positive results, but its possibility does not impact our negative result.

A.2 STAM^R Gadgets and Tools

Throughout our results we repeatedly make use of several small assemblies of tiles, referred to as *gadgets*, and patterns of signal activations to accomplish tasks such as keeping track of state, removing specific tiles, and passing information across an assembly. In this section we describe several of these gadgets and signal patterns so that they can later be referenced during our construction. We intend that this section also serve as a basic introduction by example to the dynamics of signal tile assembly.

Detector Gadgets

Detector gadgets are used to detect when a specific set of tiles exist in a particular configuration relative to one another in an assembly. For a detector gadget to work, the tiles to be detected need to each be presenting a glue unique to the configuration to be detected. The strength of these glues should add to at least the binding threshold τ, but the total strength of any proper subset of the glues should not. If two or more tiles then exist in the configuration expected by the detector gadget, the gadget can cooperatively bind with the relevant glues. Upon binding, any signals with the newly bonded glues as a source will fire. These signals can be in the "detected tiles" or in the detector itself and can be used to initiate some other process based on the condition that the tiles exist in the specified configuration. More often than not, it's also desirable for signals within the detector gadget to deactivate its own glues so that it does not remain attached to the assembly after the detection has occurred.

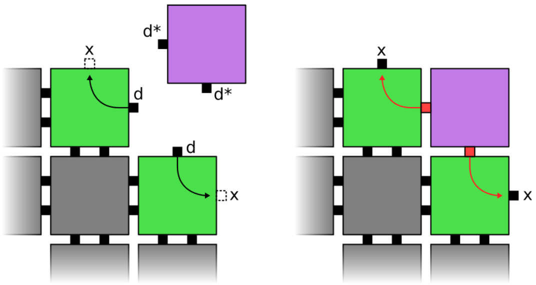


Figure 11 A simple detector gadget example.

Detector gadgets can exist in many forms depending on the configuration to detect, but the most simple is a single tile. Illustrated in Figure 11 is a simple detector gadget designed to detect 2 diagonally adjacent tiles, each presenting a strength-1 glue of type d towards a shared adjacent empty tile location. In this case, $\tau = 2$ and the detected tiles are designed to activate their x glues upon a successful detection. In general, detector gadgets can be made up of more than 1 tile. Duples of tiles can be used for instance to detect immediately adjacent tiles each presenting some specific glue on the same side. For detector gadgets consisting of more than 1 tile, the component tiles must be designed to have unique τ-strength glues between them so that the components can bind together piece-wise to form the whole gadget. Because all of the glues presented for the detection are needed to reach a cumulative strength of τ, only after it is fully formed will it be able to detect tiles and thus partially assembled detector gadgets will not erroneously perform partial detections. It is assumed in our results that signals within a detector gadget itself will cause the gadget to dissolve after a detection.

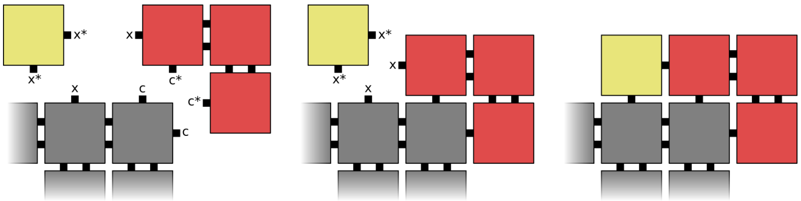


Figure 12 A corner gadget example.

Corner Gadgets

Corner gadgets are a specific type of detector gadget which are used primarily for facilitating the attachment of other tiles on the surface of some assembly. Corner gadgets can either be 2D, consisting of 3 tiles arranged in a 2×2 square with one corner missing, or 3D, consisting of 7 tiles arranged in a $2 \times 2 \times 2$ cube with one of the corners missing. Because of this shape, a corner gadget is able to cooperatively bind to any single tile of an assembly with 2 accessible, adjacent faces. These faces must be presenting specified glues whose cumulative strength is at least τ, but neither individually is. Illustrated in Figure 12 is the side view of a 2D corner gadget attaching to an assembly. After the attachment, it is then possible for additional tiles to cooperatively bind along the surface of the assembly. This behavior is useful for initiating the growth of shells of tiles around an assembly as will be seen in our later construction.

Like with detector gadgets, signals fired from the binding of a corner gadget can also be used to initiate other tasks, though special care needs to be taken for 3D corner gadgets when $\tau = 2$. Because a 3D corner gadget has 3 interior faces which can have glues to bind with a tile on the corner of an assembly, it is often desirable to fire signals from all 3 of these glues; however, because only 2 glues are necessary to meet the binding threshold when $\tau = 2$, the third may not form a bond immediately. If it is planned for the corner gadget to eventually detach, then it is crucial that any signals causing the corner gadget to detach cannot fire until all 3 of the interior glues have first bound. This can often be accomplished using *sequential signaling* as described below.

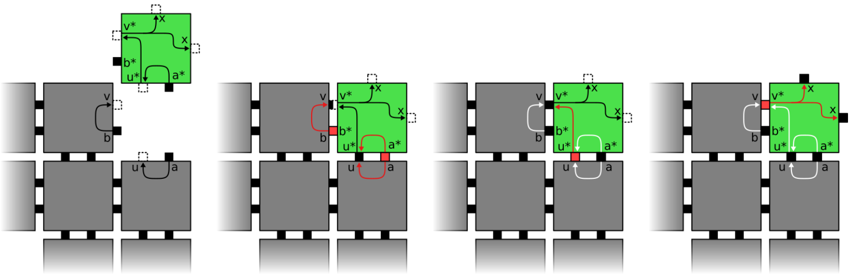


Figure 13 Sequential signaling example.

Sequential Signaling

By carefully adding additional helper glues and signals to a tile or tiles, we can ensure that signals in our tiles are fired in a specific order or ensure that a certain set of glues has successfully bound before certain signals are fired. The way in which this is done depends on the exact situation, but as an example consider the situation illustrated in Figure 13. In this situation we want the green tile to cooperatively bind to the assembly via glues of type a and b. Once this happens, we want to first activate additional glues of type u and v between the green tile and assembly so that each side of the green tile is attached to the assembly with strength 2, then we want glues of type x on the other sides of the green tile to activate. The arrangement of signals illustrated in Figure 13 guarantees that the x glues cannot activate before both the u and v glues do, since the signals which activate the x glues are dependent on the glues u and v. A similar arrangement of signals and glues is used to implement gadgets called *filler tiles* in our construction.

Tile Conversion

It is often useful for tiles to change behavior after receiving a specific signal. This can be done by having signals activate a new set of glues on the tile and deactivate old ones. This can be thought of as converting the tile into a different type of tile, but it's important to note that this process cannot happen indefinitely nor arbitrarily. Every tile conversion has to be prepared in the signals and latent glues of the tile and once those signals fire, they cannot fire again. It is possible for a tile to convert to another several times, but such a tile must have the necessary glues and signals for each conversion separately. It is also often possible achieve this behavior by detachment of one tile and attachment of another in the same location, though special care needs to be taken so that no other tiles can attach in the location during the conversion.

Tile Dissolving

For any arbitrary set of glues on a tile, we use the term *dissolving* to refer to the process of initiating signals which turn all possible glues to the `off` state We note that due to the asynchronous nature of the model that no guarantee can be made with regards to the order of the processing of the signals. A tile breaks apart from its supertile once a strength τ bond no longer exists between itself and its neighbors. However, other glues may be active when the tile does so, leading to the possibility of undesired binding due to exposed glues in the `on` state with pending `off` signals, unless special care is made to avoid this.

A.3 STAM* overview

The 3D Signal-passing Tile Assembly Model* (3D-STAM*, or simply STAM*) was introduced in [3] as a generalization of the STAM [28, 15, 16, 23] in which (1) tiles are 3D cubes rather than 2D squares, (2) multiple tile shapes are allowed in the same system, (3) tiles are allowed to flip and rotate (similar to [8, 19]), and (4) glues are allowed to be defined as either rigid (as in the aTAM, STAM, etc.) or *flexible* (as in [13]) so that even after being bound, tiles and subassemblies are free rotate with respect to tiles and subassemblies to which they are bound by bending or twisting around a "joint" in the glue.

A Coupled Reconfiguration Mechanism for Single-Stranded DNA Strand Displacement Systems

Hope Amber Johnson
The University of British Columbia, Vancouver, Canada

Anne Condon
The University of British Columbia, Vancouver, Canada

Abstract

DNA Strand Displacement (DSD) systems model basic reaction rules, such as toehold-mediated strand displacement and 4-way branch migration, that modify complexes of bound DNA strands. DSD systems have been widely used to design and reason about the correctness of molecular programs, including implementations of logic circuits, neural networks, and Chemical Reaction Networks. Such implementations employ a valuable toolkit of mechanisms – sequences of basic reaction rules – that achieve catalysis, reduce errors (e.g., due to leak), or simulate simple computational units such as logic gates, both in solution and on surfaces. Expanding the DSD toolkit of DSD mechanisms can lead to new and better ways of programming with DNA.

Here we introduce a new mechanism, which we call *controlled reconfiguration*. We describe one example where two single-stranded DSD complexes interact, changing the bonds in both complexes in a way that would not be possible for each independently on its own via the basic reaction rules allowed by the model. We use *coupled reconfiguration* to refer to instances of controlled reconfiguration in which two reactants change each other in this way. We note that our DSD model disallows pseudoknots and that properties of our coupled reconfiguration construction rely on this restriction of the model.

A key feature of our coupled reconfiguration example, which distinguishes it from mechanisms (such as 3-way strand displacement or 4-way branch migration) that are typically used to implement molecular programs, is that the reactants are single-stranded. Leveraging this feature, we show how to use coupled reconfiguration to implement Chemical Reaction Networks (CRNs), with a DSD system that has both single-stranded signals (which represent the species of the CRN) and single-stranded fuels (which drive the CRN reactions). Our implementation also has other desirable properties; for example it is capable of implementing reversible CRNs and uses just two distinct toeholds. We discuss drawbacks of our implementation, particularly the reliance on pseudoknot-freeness for correctness, and suggest directions for future research that can provide further insight on the capabilities and limitations of controlled reconfiguration.

2012 ACM Subject Classification Theory of computation; Applied computing → Chemistry

Keywords and phrases Molecular programming, DNA Strand Displacement, Chemical Reaction Networks

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.3

Funding *Hope Amber Johnson*: Supported by an NSERC Discovery Grant.
Anne Condon: Supported by an NSERC Discovery Grant.

1 Introduction

1.1 Summary

An important task in the field of molecular programming is to develop effective molecular programming languages, and to develop an effective abstraction hierarchy. Ideally, human-readable and human-writable programs at the top level of the hierarchy can be transformed

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham; Article No. 3; pp. 3:1–3:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

without further human input to a molecular implementation at the bottom level. A strong candidate pair of layers is the Chemical Reaction Network (CRN) model and its implementation with DNA Strand Displacement (DSD) systems. CRNs are a particularly well-studied abstract model of well-mixed chemical systems, in which many interesting programs can be and have been written [2, 3, 28, 36, 41]. Moreover, arbitrary CRNs can be transformed into DSD systems that implement the original CRN [7, 10, 30, 37, 39].

However, while we now have a good understanding of the capabilities and limitations of CRNs, we cannot say the same for DSD systems. Most DSD systems [35] use one of two simple mechanisms, three-way toehold exchange or four-way strand exchange, and the range of alternative mechanisms that could be used when designing DSD systems that simulate CRNs or other molecular programs is not well understood. We believe a more formal understanding of DSD would aid in the design process, exploring more – and possibly more efficient – ways of getting things done with DSD systems. Some work has been done toward this goal [4, 21, 23, 26, 29]. In the process of trying to improve formal understanding, we discovered a new mechanism in an unexplored area of the DSD design space. In this paper we present this new mechanism, show how it can be used to implement arbitrary CRNs, and discuss its further implications for understanding DSD.

We describe further background information, then we discuss our contribution, a new DSD mechanism we call *controlled reconfiguration* and its subclass *coupled reconfiguration*. This new mechanism has a number of properties fundamentally different from typical existing DSD mechanisms, including the use of single-stranded signals and fuels, and a stronger than usual dependence on the assumption of pseudoknot-freeness. We discuss the implications and challenges arising from those, including the ability of coupled reconfiguration to implement arbitrary CRNs.

1.2 Background

Our work revolves around DNA Strand Displacement (DSD): certain behaviors of DNA strands that have been used to build useful molecular devices, including logic circuits and neural networks [11, 31], DNA walkers on surfaces [43, 45], and many other things [35]. In one particularly powerful use, DSD can implement arbitrary Chemical Reaction Networks [7, 10, 30, 37, 39]. Figure 1 gives an example of a DSD implementation of a single reaction; details of the implementation are described further in Section 2. As in this example, DSD systems often rely on *fuels*, complexes of multiple DNA strands in a specific configuration which have to be made by an external process.

The Chemical Reaction Network (CRN) model is an abstract description of chemical dynamics that is widely used as a language for programming molecules. We focus on stochastic CRNs, which describe how counts of molecular species evolve when molecules react in a well-mixed solution. Stochastic CRNs are closely related to population protocols, a programming model in distributed systems [1, 2]. Simple CRNs (or population protocols) can compute approximate majority [3], simulate a 3-peak oscillator [39], or simulate boolean circuits [28]. In fact, CRNs can simulate Turing machines, albeit with a small probability of error [36]. However, the class of functions computable by CRNs without error is quite limited, being just the semilinear functions [1, 2, 9, 17, 27, 33]. Consequently, polymer systems [8, 24, 25, 30, 40] and surface CRNs [12, 32] have been proposed as more powerful molecular programming languages, capable of simulating Turing machines without error.

Importantly, arbitrary CRNs can, in theory, be “compiled” into physically realizable DSD systems [37], and in fact all of the CRN models listed above have DSD implementations [12, 30, 32]. These compilation schemes provide an abstraction hierarchy for molecular

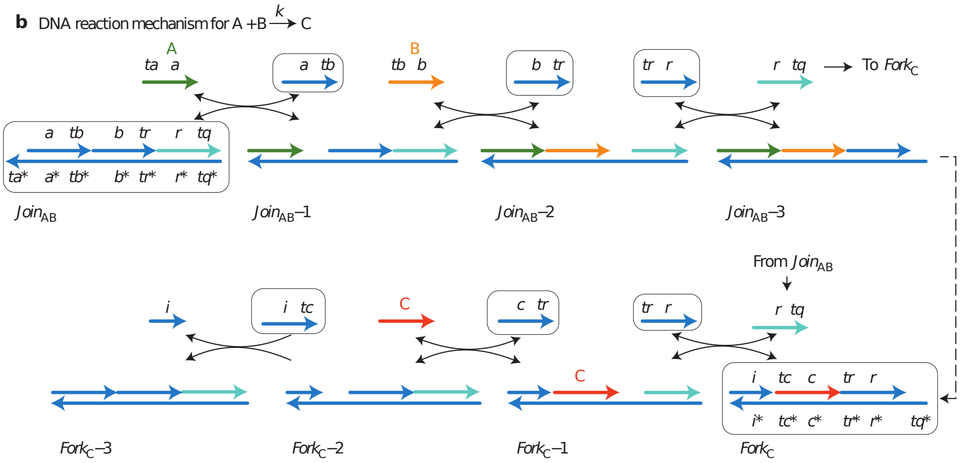


Figure 1 DSD implementation of a CRN reaction $A + B \rightarrow C$, adapted from Chen et al. [10]. The implementation consumes various *fuel* strands and complexes (in rounded boxes) to convert *signal* strands (labeled) A and B into the signal strand C. The implementation is a sequence of *primitive reactions*, as listed in Figure 2. For example, at the top left, the fuel $Join_{AB}$ reacts with the signal A, first by binding of toehold domains ta and ta^* (primitive reaction b), followed by 3-way strand displacement (primitive reaction m_3), producing complex $Join_{AB}$-1 and the strand with long domain a followed by toehold tb. In a more general system with multiple reactions, the fuels (and initial signals) must all be produced by an external source.

programming, analogous to compiling high-level languages into assembly and machine languages in traditional programming. CRNs are at the top level of the hierarchy, experimental DNA systems at the bottom, and formal DNA systems in the middle.

While the powerful models above have DSD implementations, we don't yet fully understand the capabilities and limitations of DSD systems. For example, consider the difficulties in building large DSD systems. So far, experimentally demonstrated DSD systems are usually small. CRN implementations, for example, have been demonstrated for CRNs of 3-4 species and reactions [10, 39]. Logic circuits and winner-take-all circuits specifically designed to scale up to larger systems have reached 6-gate circuits [31] and 100-input winner-take-all circuits [11]. Srivinas et al. have examined the effect of "leak" on DSD systems, which would get larger as systems try to scale up [39]. In particular, more complex fuels are less reliable. The previous examples of scaled-up circuits minimize error by using fuels of at most 2 strands [11, 31, 44]. "Leakless" gates have been proposed that would minimize this problem [42, 48], but this has not yet led to reliable large-scale CRN implementations. For this reason we are interested in understanding the capabilities of DSD better, to determine whether even simpler fuels can be used to do more complex tasks such as CRN implementations. We have some previous work exploring ways to formally implement CRNs with 2-stranded fuels [21, 23], but this also has not yet led to experimentally demonstrated large-scale CRN implementations. Existing DSD systems based on single-stranded fuels, such as the hybridization chain reaction [16] or assembly of multi-armed junctions [50], aggregate strands into large, multi-stranded complexes that mostly don't come apart, and so are not suitable for CRN simulation. To explore new DSD mechanisms that could be used for simulation of CRNs and other molecular programming models, we turn to formal DSDs.

Two examples of formal DSD models are Visual DSD [26, 29] and Peppercorn [4, 5]. These models ignore some aspects of experimental DNA systems, such as the actual sequence of bases in a DNA strand, and instead model strands as sequences of *domains*. The models

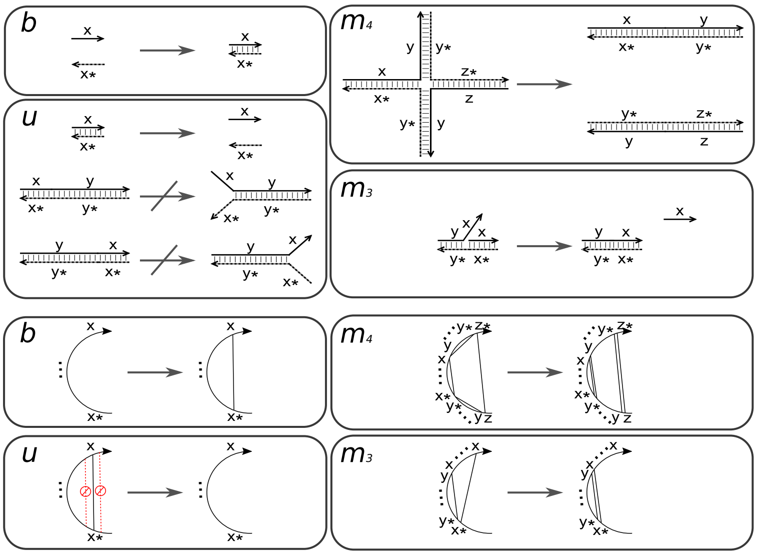


Figure 2 The four primitive reactions, or basic steps, of our DSD model, similar to those used by Petersen et al. [29] and our previous work [21]. Top half: the typical diagram format used in DSD, image adapted from our previous work [21]. The four primitive reactions shown are: domain binding (b), toehold (i.e., short domain) unbinding (u), 4-way branch migration of long domains (m_4) and three-way branch migration of long domains (m_3). Bottom: the same reactions in the circle diagram format we will use throughout this work. Circles indicate strands, with domains labeled; arrows indicate 3' ends; lines or arcs between domains indicate bonds. The crossed-out dashed bonds in the u reaction indicate that the reaction only happens if no such bonds exist. Regions represented by . . . may or may not include strand breaks.

also specify how DNA complexes are formed and modified via primitive reactions, or basic steps that involve binding and/or unbinding of domains. The different models broadly agree on the four primitive reactions, shown in Figure 2, but differ on some details. Given an initial set of complexes, the set of DNA complexes that can be enumerated from the initial set using the four primitive reactions, plus the associated reactions, comprise the species and reactions of a CRN which is referred to as the enumerated CRN.

This also allows formal verification that a DSD implementation of an original CRN is correct by comparing the enumerated and original CRNs. Definitions of "correct" implementation include pathway decomposition [34] and CRN bisimulation [22] as well as some others, and the whole process can be automated by the Nuskell compiler [5]. But formalization also allows abstract reasoning about the model, for example, our previous work proved that no DSD system can implement CRNs with a particular, very restrictive, set of properties [21].

Formal models define the four basic steps, but most work with DSD is done on the level of what Peppercorn calls *condensed reactions* [4, 21], short sequences of basic steps that "usually" occur as a unit. In fact, most DSD systems [35] use a variant of the same, simple condensed reaction, toehold-mediated 3-way branch migration, with a few exceptions. But while the set of basic steps is defined by the model, the space of possible condensed reactions is much less limited and not well explored. New condensed reactions might allow DSD systems with simpler fuels that are less prone to leak, or we might find a proof that the existing mechanisms are the simplest possible methods for doing complex tasks.

1.3 Our contributions

In this work, we describe a fundamentally new class of condensed reactions, which we call *controlled reconfiguration*, and a subclass we call *coupled reconfiguration.* This controlled reconfiguration mechanism relies on assumptions made in the formal model that haven't been tested experimentally, so whether it will work is uncertain. However, if the mechanism works as intended, it would enable a fully general CRN implementation using only single-stranded complexes as fuels, as opposed to the multi-stranded complexes of existing methods.

In Section 2, we define the CRN and DSD models that we use throughout the paper. In Section 3 we show the *controlled reconfiguration* and *coupled reconfiguration* mechanisms that are the subject of this paper. In Section 4 we show how coupled reconfiguration can be used to implement arbitrary CRNs, and give the sketch of a proof that the implementation is formally correct according to CRN bisimulation [22]. In Section 5 we discuss the ways in which controlled reconfiguration relies on the no-pseudoknots assumption of the model, and what can be learned from it even if that assumption turns out not to hold for physical DNA. Finally, Section 6 contains further discussion and conclusions.

2 Chemical Reaction Networks and DNA Strand Displacement Systems

We first describe Chemical Reaction Networks (CRNs), and then describe our formalization of DNA Strand Displacement (DSD) systems, which is a particular type of Chemical Reaction Network. We conclude by comparing our DSD system formalization with other variants in the literature.

Chemical Reaction Networks describe how states of a system of molecules change as a result of reactions. Let $\mathcal{S}$ be a set of molecular *species.* A *state* $\mathbf{s}$ is a multiset of species; we'll represent such multisets as functions $\mathbf{s} : \mathcal{S} \to \mathbb{N}$ mapping species to nonnegative integers, so $\mathbf{s}(\sigma)$ is the number of copies of σ in multiset $\mathbf{s}$. We also represent a multiset, such as $\mathbf{s}$, using summation notation: $\sum_{\sigma:\mathbf{s}(\sigma)>0} \mathbf{s}(\sigma)\sigma$. We use $\emptyset$ to denote the empty multiset. A *reaction* is a tuple $(\mathbf{r}, \mathbf{p}, k)$, where the reactants $\mathbf{r}$ and products $\mathbf{p}$ are multisets of species and k is a positive real number, called the reaction rate constant. We write reaction $(\mathbf{r}, \mathbf{p}, k)$ as

$$\sum_{\sigma:\mathbf{r}(\sigma)>0} \mathbf{r}(\sigma)\sigma \xrightarrow{k} \sum_{\sigma:\mathbf{p}(\sigma)>0} \mathbf{p}(\sigma)\sigma.$$

The reaction is *unimolecular* if $\mathbf{r}$ has size 1, and *bimolecular* if $\mathbf{r}$ has size 2 (where n copies of a single species σ counts as size n). Sometimes we omit the rate constant, when it is not pertinent to the constructions presented. For example, if $\mathcal{S} = \{A, B, C\}$, $\mathbf{r}$ is the multiset containing two A's and one C, and $\mathbf{p}$ is the multiset containing one B, then we can write the reaction $(\mathbf{r}, \mathbf{p})$ (ignoring the rate constant) as

$$2A + C \to B.$$

Formally, a *Chemical Reaction Network* (CRN) is a pair $(\mathcal{S}, \mathcal{R})$, where $\mathcal{S}$ is a finite set of species and $\mathcal{R}$ is a finite set of reactions involving the species in $\mathcal{S}$. Reaction $(\mathbf{r}, \mathbf{p})$ is *applicable* to state $\mathbf{s}$ if $\mathbf{s}(\sigma) \geq \mathbf{r}(\sigma)$ for all $\sigma \in \mathcal{S}$, and the result of the reaction is the state $\mathbf{s} - \mathbf{r} + \mathbf{p}$. If both $(\mathbf{r}, \mathbf{p})$ and $(\mathbf{p}, \mathbf{r})$ are in $\mathcal{R}$, we say that the reaction $(\mathbf{r}, \mathbf{p})$ is *reversible* and we write

$$\sum_{\sigma:\mathbf{r}(\sigma)>0} \mathbf{r}(\sigma)\sigma \rightleftharpoons \sum_{\sigma:\mathbf{p}(\sigma)>0} \mathbf{p}(\sigma)\sigma.$$

Our formalization of DNA Strand Displacement (DSD) systems is similar to several variants in the literature [5, 21, 26, 29, 38] and we note below where our model diverges from others. A *DSD system* is specified as a finite multiset of *complexes*, and a set of four *DSD primitive reactions*, or basic steps.

- A complex is a sequence of *strands* that are connected by *bonds*. Each strand is a sequence of *domains*, where each domain has an identity, or label, taken from some finite set. For each domain identity x in the set, there is also a complement, x^* with $(x^*)^* = x$, and bonds of a complex are always between domains with complementary identities. Domains can be *long* or *short*, and we refer to short domains as *toeholds*. Two consecutive domains on the same strand are *adjacent*. See Figure 2 for a visual representation of a complex: strands are arranged along a half-circle, with the sequence of strand domains in clockwise order, and an arc connects each pair of bound (complementary) domains. Complexes must be *pseudoknot free*: there is some ordering of strands along the circumference of a circle in which no bonds cross. (This ordering is in fact unique, see [14]).
- Primitive reactions, or basic steps, transform complexes while maintaining pseudoknot-freeness, and are of four types (also shown in Figure 2):
  - *Binding* (b): A bond is added between a pair of complementary domains. The domains may be in the same complex or in two different complexes; in the latter case the bond forms one new complex from the two participating complexes.
  - *Toehold unbinding* (u): A bond between a toehold t and its complement t^* is removed. This rule can only be applied if there is no bond between a pair of domains d and d^* that are adjacent to t and t^* respectively. Toehold unbinding applied to a complex produces either one or two complexes.
  - *3-way branch migration* (m_3): A bound long domain switches partners. That is, a complex in which long domains x_1 and x_2 are bound, and domain x_3 with same label as x_1 is unbound, is transformed so that x_2 and x_3 are bound and x_1 is unbound. This rule can be applied only when there is a bond between (complementary) domains respectively adjacent to x_2 and x_3.
  - *4-way branch migration* (m_4). Two pairs of bound long domains swap partners. That is, a complex in which domain x_1 is bound to x_2 and domain x_3 is bound to x_4, where x_1 and x_3 have the same domain label (and so x_2 and x_4 have the complementary domain label) is transformed so that x_1 is bound to x_4, and x_2 is bound to x_3. This rule can be applied only when there is a bond between domains adjacent to x_1 and x_4, plus a bond between domains adjacent to x_2 and x_3.
- The following sequences of steps (sometimes called *motifs* [23]) appear often, acting as a unit, in our mechanisms and many others:
  - *Toehold-mediated 3-way strand exchange* (tx_3). One pair of toeholds binds (b). The two toeholds were adjacent to a pair of bound domains x, x^* and an unbound domain x, respectively, which are now able to branch migrate (m_3). The previous bond was adjacent to a pair of toeholds on the other side and preventing them from unbinding, but now they are able to do so (u).
  - *Two-toehold-mediated 4-way strand exchange* (tx_4). Two pairs of toeholds bind in either order, in such a way that a four-way junction is formed (b, b). The four-way junction then branch migrates (m_4). The previous bonds in the four-way junction were each adjacent to a pair of toeholds on the opposite side, which are now capable of unbinding in either order (u, u).

(Both of those motifs are reversible as long as the initial toehold binding(s) is/are reversible, and in fact the reverse of each one is an instance of the same motif. Concrete examples of the motifs occur in Figures 3 and 4.)

The four basic steps are often grouped into *condensed reactions*, sequences of one bimolecular step followed by as many unimolecular steps as necessary until no further meaningful unimolecular steps are possible [4, 21]. The tx_3 and tx_4 motifs, for example, are often used as entire condensed reactions [23, 35], but can also be used as parts of larger condensed reactions. We don't use the details of this definition much, but the mechanisms we present in Section 3 are in fact condensed reactions.

Let C be a set of complexes. The *DSD system enumerated from* C, using the set of reaction rules $\{b, u, m_3, m_4\}$, is the smallest CRN $(\mathcal{S}, \mathcal{R})$ such that any complex in C is a species in $\mathcal{S}$, any application of DSD reaction rules b, u, m_3 or m_4 to reactants in $\mathcal{S}$ is a reaction in $\mathcal{R}$, and the products of each such reaction are species in $\mathcal{S}$.

We end this section by summarizing differences between our DSD system formalization and other formalizations in the literature. A key point is that our definition enforces pseudoknot-freeness: no reactions may introduce pseudoknots. The DSD systems of Badelt et al. [5], Johnson [21], and Petersen et al. [29] also enforce pseudoknot-freeness, but those of Lakin et al. [26] and Spaccasassi et al. [38] allow pseudoknotted complexes. We emphasize that our constructions in the following sections would *not* be correct in a model that includes pseudoknots. We discuss the implications of our pseudoknot-freeness constraint further in Section 5.

Another difference is that in some models [21, 26], toeholds can mediate 3-way strand displacement even when not adjacent to the long domains that bind; rather the toeholds can be "adjacent" to the reaction in a weaker sense than we use it here [29], or in fact be "remote" [20, 21]. We also do not include yet other DNA strand displacement reactions that have been used in experimental work, such as cooperative strand displacement [51]. None of these additions to the model are needed for our purposes. In contrast with pseudoknots, we believe that our correctness arguments can be adapted even if the model is generalized in these ways.

3 Controlled reconfiguration and coupled reconfiguration

We present a class of mechanisms that we call *controlled reconfiguration*, and a subclass we call *coupled reconfiguration*. The mechanisms work in the formal DSD model as defined in Section 2, but have not yet been tested experimentally. In particular, these methods depend on the assumption that domains cannot bind if it would create a pseudoknot, which is one of the more questionable assumptions. Whether this mechanism describes the behavior of real DNA strands, or whether it is better interpreted as an exploration of the formal model, is discussed properly in Section 5. All further discussion until then is assuming everything functions as described in the model.

Figure 3 shows an example of a *controlled reconfiguration* mechanism. The long strand, as a single copy by itself, has two stable configurations, and cannot switch between them; all the free X^* and Y^* domains that could start the reconfiguration are themselves blocked off by existing bonds, and the interaction necessary to do so would be a pseudoknot. The short strand displaces the bond between Z domains, triggering a sequence of steps each involving a domain "freed" by the previous displacement. Eventually, the same Z bond is re-formed and the short strand is released, with the long strand having reconfigured. Intuitively, the bonds function as locked doors, preventing what's behind them from interacting with what's outside, until unlocked with a key hidden behind the previous door.

The controlled reconfiguration mechanism is an example, and proof-of-concept, of how one strand can control the reconfiguration of another. However, in the mechanism in Figure 3, the "control" strand only allows the reconfiguring strand to freely switch between its two

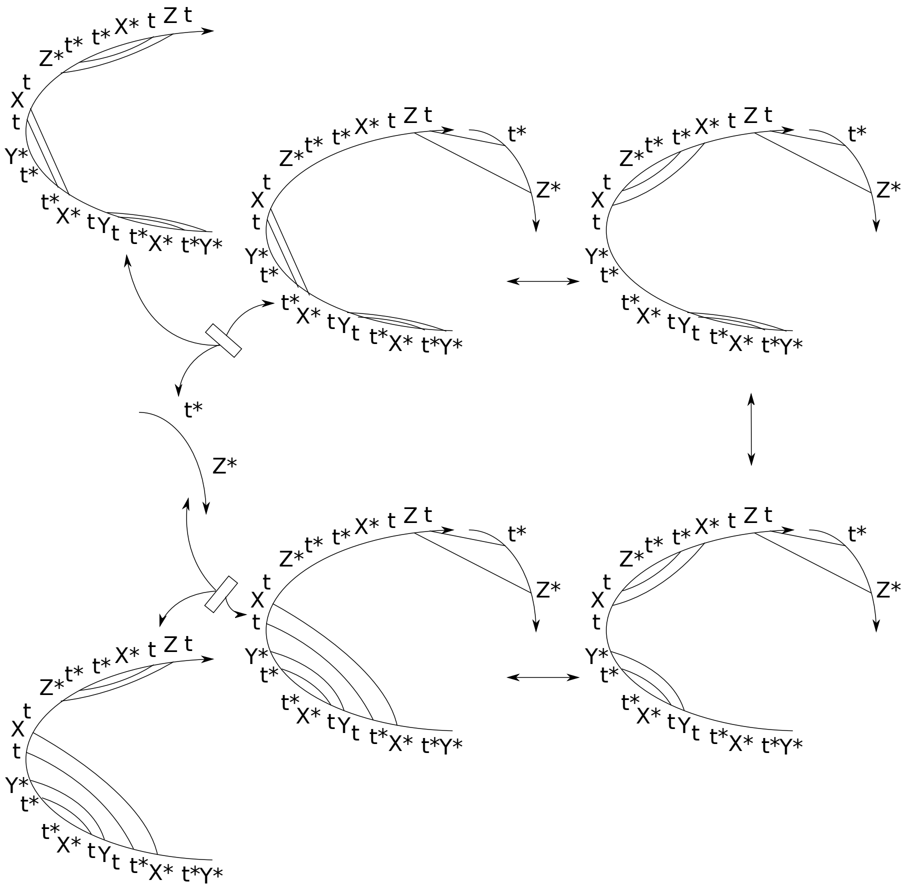


Figure 3 A simple controlled reconfiguration mechanism. t is a short domain, X, Y, and Z are long. Each step in the figure is a tx_3 motif. The net effect is that the long strand changes from the configuration in the top left to the configuration in the bottom left, using the short strand as a catalyst, while the long strand could not change configuration on its own.

configurations. Neither the control strand nor any other part of the system knows which configuration the reconfiguring strand is in, so it can't, for example, force the strand into a desired configuration and then proceed to the next task. This limits its usability as a module in larger systems. For that, we would want a mechanism where two different strands simultaneously serve as the control for each other's reconfiguration, in such a way that either both can change or neither can change, but not just one or the other. Figure 4 shows the *coupled reconfiguration* mechanism, which we will use for our formal CRN implementation.

In the coupled reconfiguration mechanism, each strand has a pair of A_1, A_1^* domains and a pair of A_2, A_2^* domains, but crossed over such that (excluding pseudoknots) only one pair can be bound at once, and it cannot reconfigure on its own. Further, there is a pair of G ("guard") domains that, by their bond, prevent random other strands from interacting with the domains inside the strand. The "left" and "right" strands are asymmetric in the toehold identities, and one left and one right strand can come together and exchange G bonds by a tx_4 motif. This opens each strand up to interacting with the other. (Two left strands or two right strands do not have complementary external toeholds, so such pairs cannot interact.) Then, if one of the strands was in the A_1 configuration and the other in the A_2 configuration, they can go through a process of exchanging bonds that eventually leads to both strands

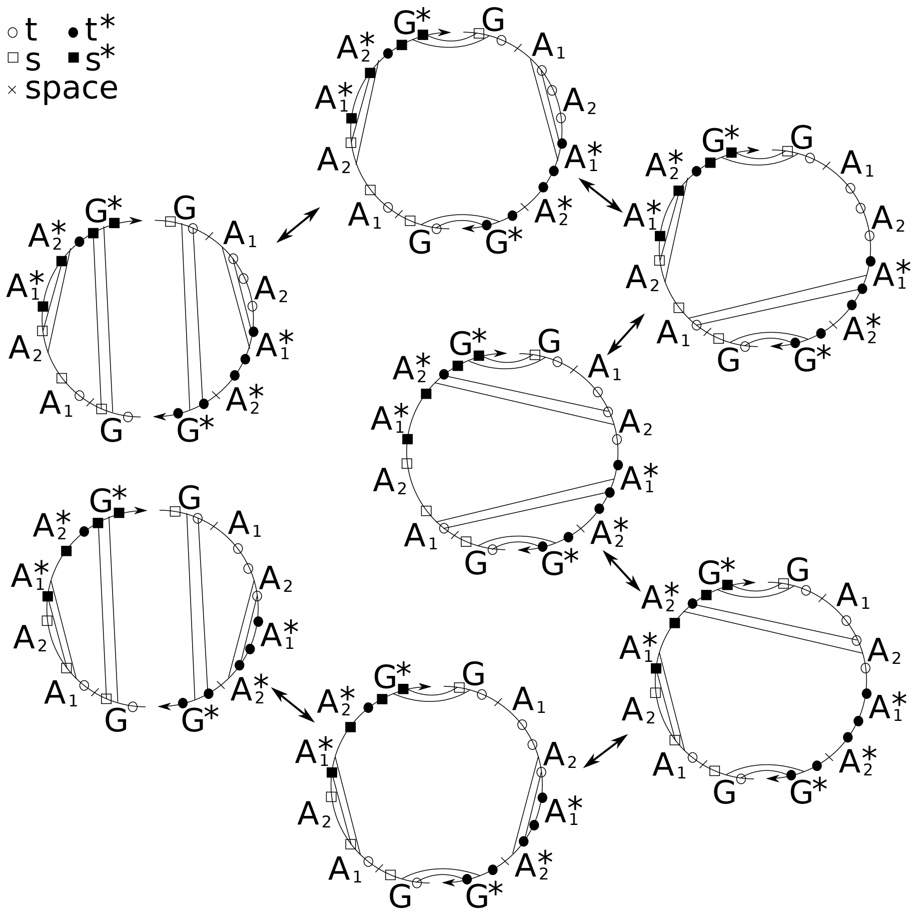


Figure 4 The coupled reconfiguration mechanism. Long domains are given by name while toeholds are represented by symbols. Space domains don't interact with anything, but prevent some spurious interactions. A left and a right strand can join together to reconfigure via a tx_4 motif on G domains, and separate when the reconfiguration is done by reversing the tx_4 motif. Then a sequence of tx_3 motifs lets each strand enable the reconfiguration of the other, one strand moving from the A_1 configuration to the A_2 configuration and the other doing the opposite. The mechanism is fully reversible, but the strands can only separate if either both strands have changed configuration or neither has.

having swapped configurations. The G bonds can then reverse the tx_4 motif to separate again, but this can only happen when either both have reconfigured or neither has. Any left and right strand in the same configuration can join together, but no further reaction can happen except for separating unchanged.

The reaction as shown in Figure 4 is fully reversible, but can be made irreversible by adding any of the following four toeholds: a t^* after the A_1^* on the left strand or after the A_1^* on the right strand makes the reaction only go forward (clockwise around the diagram shown), while a t before the A_2 on the left strand or before the A_1 on the right strand makes the reaction only go backward. In either case, the strand the toehold is added to becomes unreactive after the reaction, while the other strand is still capable of further coupled reconfigurations (with counterpart strands that don't have the added toehold).

The coupled reconfiguration reaction can be abstracted as $A_{2,l} + A_{1,r} \rightleftharpoons A_{1,l} + A_{2,r}$. Since each configuration has a different bond, there are regions of each strand that are covered in one configuration and uncovered in the other, and different (left and/or right) strands can share the A_1, A_2 domain identities while having different content in the conditionally covered regions. This mechanism turns out to be powerful enough to implement arbitrary CRNs.

We think of "controlled reconfiguration" more generally as any DSD reaction where two or more complexes interact and separate, where (at least) one of the complexes at the end is made up of the same strands in a different configuration, such that the reconfiguration could not have happened in that complex on its own. The example in Figure 3 is in particular "catalytic controlled reconfiguration", where the complex that controls the reconfiguration is itself unchanged. Then "coupled reconfiguration" more generally is where two or more complexes interact and separate, where (at least) two of the complexes at the end have reconfigured in a way that each one could not on its own, and further such that the reaction could have reconfigured both complexes or neither, but not only one of them. Then the mechanisms we presented were examples of controlled and coupled reconfiguration, but other examples exist. In particular, Zhang and Winfree have presented a mechanism that meets this definition of controlled reconfiguration, but is neither catalytic nor coupled [52].

4 Implementing CRNs with coupled reconfiguration

The coupled reconfiguration mechanism enables different strands to, in a certain intuitive sense, pass information to each other by reconfiguring. This is sufficient to implement, among other things, arbitrary CRNs. In Section 4.1 we discuss how to combine coupled reconfiguration reactions to implement CRNs. In Section 4.2 we give an argument for formal correctness of this method according to CRN bisimulation [22].

4.1 The mechanism of a reaction

We first consider the reaction $A + B \rightleftharpoons C$, which is sufficient to implement arbitrary reversible CRNs (and arbitrary CRNs, if both directions are available as irreversible reactions). We use what in Section 3 we referred to as "right" strands as *species* strands that represent the abstract species, with species A getting a unique pair of long domains A_1 and A_2, and corresponding to right strands with the (toeholds omitted) sequence $G\ A_1\ A_2\ A_1^*\ A_2^*\ G^*$. Such a strand in the configuration with A_1 domains bound to each other represents the species A, and we call the strand in that configuration the *signal* strand A_{on}. The same strand in the A_2 configuration is called A_{off}, and represents nothing. Species B and C get unique domains B_1, B_2, C_1, and C_2, according to the same pattern.

To implement the reaction we use what in Section 3 we called "left" strands as *gate* strands. Where r refers to the reaction $A + B \rightleftharpoons C$, the gate strand will interleave the domains for A, B, and C in the order shown in Figure 5. To implement the forward reaction, the gate strand starts in configuration r_{for}, with bonds A_2, B_2, and C_1. A coupled reconfiguration reaction with A_{on} produces r_{Afor}, where the B_2 bond is exposed. This then reacts with B_{on} to produce r_{Cback}, covering up the A_1 bond but exposing the C_1 bond. That allows a reaction with C_{off}, producing C_{on} and r_{back}, where the B_2^* and A_2^* domains are covered by the C_2 bond. Thus, the gate consumes a signal A and B and produces a signal C, where each step covering up the previous step and uncovering the next step ensures that they can only be done in that order. The reverse reaction is implemented by the reverse of this process, starting with r_{back} and eventually producing r_{for}.

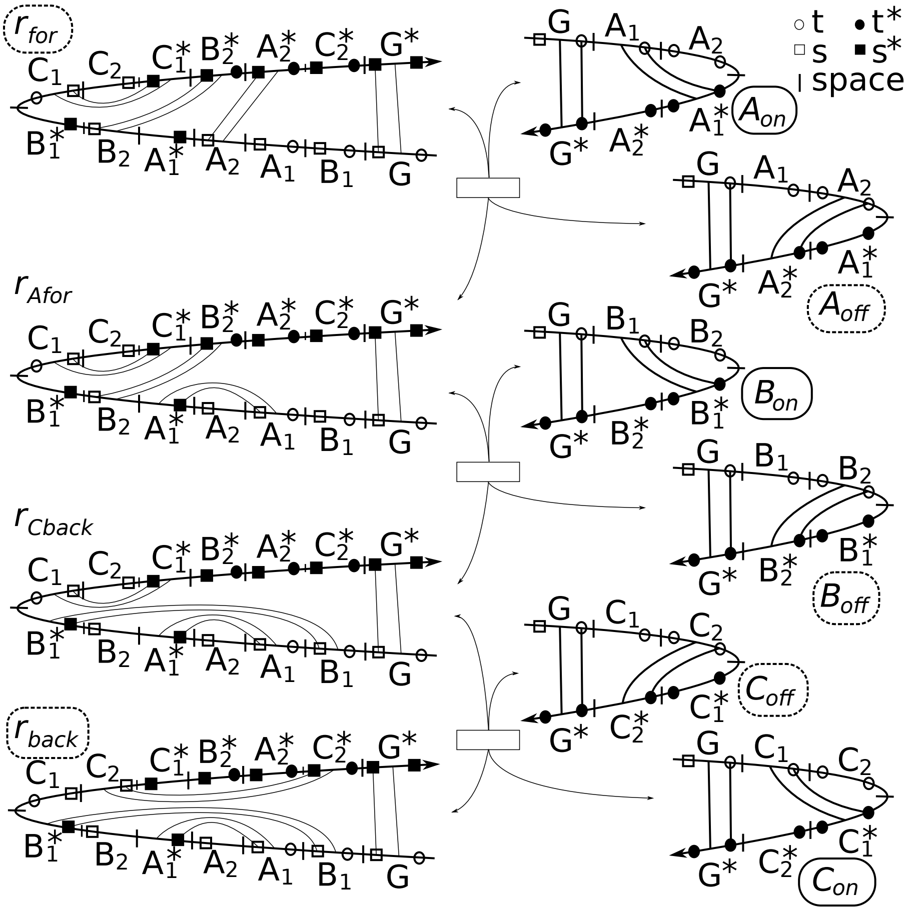


Figure 5 Simulation of the reaction $r = A + B \rightleftharpoons C$. Each reaction represents an instance of the coupled reconfiguration sequence shown in Figure 4. Spacer domains separate each long domain and its flanking toeholds from those of the adjacent domain, to prevent certain unintended interactions. Species are given names with solid boxes for signal strands, dashed boxes for fuel strands, and no boxes for intermediate (other) strands.

4.2 Correctness of the mechanism

To prove our mechanism correct, we use CRN bisimulation [22]. Here, however, we only give a sketch of the process. First, we can describe the DSD system as a CRN, the *implementation CRN*, compared to the *formal CRN* that it allegedly implements. To do that we enumerate the possible interactions between strands in our DSD system, according to the rules in Section 2.

The complexes in the system include some designated as *fuels*, namely r_{for} and r_{back} for each reaction r, and A_{off} for each species A, and some designated as *signals*, namely A_{on} for each species A. Fuels are assumed to be always present in "enough" quantity (and how to achieve that is left as an exercise for the experimenter), while signals represent the corresponding formal species. The interactions between strands fall into the following categories:

- Spurious toehold bindings that don't lead to an m_4 step between G domains.
- "Nuisance" interactions between a gate strand and a species strand that have no intended, and no possible, productive reaction.
- Productive reactions between strands intended to interact.

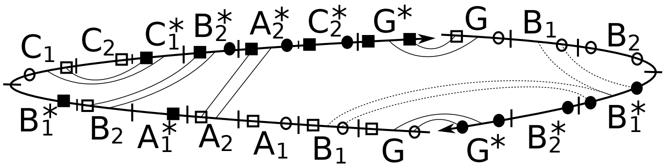


Figure 6 A nuisance interaction between r_{for} and B_{on}. Any gate strand and any signal strand can bind and open up the G domains, but only pairs intended to react can meaningfully reconfigure. In this example, r_{for} and B_{on} can bind, open G domains, and even do toehold exchange (represented by dashed lines) on the B_1 domains, but further interaction with B domains is blocked by the A_2 bond. The only possible continuation is for the reaction to eventually reverse itself and separate back into r_{for} and B_{on} unmodified.

In the first category, arbitrary sequences of b and u steps between strands on external toeholds can happen, but can only make a meaningful change if they eventually form a binding of two toeholds that enables an m_4 step on G domains. Otherwise, the strands involved will all eventually unbind unchanged. This can form arbitrarily large (and increasingly unlikely) chains, which to formally treat requires the techniques of Polymer Reaction Network bisimulation [24]; this can be done, but we don't include it here.

In the second category, any gate strand and any species strand can bind on two toeholds and open the G domains in a tx_4 motif. But they can only do a coupled reconfiguration reaction if the gate strand has the domains corresponding to the species strand in the opposite configuration (e.g. a gate strand with a B_2 bond interacting with a species strand with a B_1 bond) and none of those domains on the gate strand are blocked by any other bond. Any other combination may start, but cannot finish, a coupled reconfiguration reaction, and can only eventually separate into the original unmodified strands. Figure 6 gives an example of such an interaction.

The third category is the reactions shown in Figure 5. Effectively, this means the set of reachable complexes is the set of things shown in that figure; intermediate states between them; and spurious bindings from the first two categories. Modeling the system with the *condensed reaction* model [4, 21] allows us to treat the first and second categories as trivial, abstract away from the intermediates, and describe the implementation CRN for the formal reaction $A + B \rightleftharpoons C$ as follows:

$$\begin{aligned} r_{for} + A_{on} &\rightleftharpoons r_{Afor} + A_{off} \\ r_{Afor} + B_{on} &\rightleftharpoons r_{Cback} + B_{off} \\ r_{Cback} + C_{off} &\rightleftharpoons r_{back} + C_{on} \end{aligned}$$

Since fuels are always present (or always producible), bisimulation and other verification methods often work with an implementation CRN where fuels are removed:

$$\begin{aligned} A_{on} &\rightleftharpoons r_{Afor} \\ r_{Afor} + B_{on} &\rightleftharpoons r_{Cback} \\ r_{Cback} &\rightleftharpoons C_{on} \end{aligned}$$

CRN bisimulation involves an *interpretation* of implementation species as (multisets of) formal species, usually denoted m; here $m(A_{on}) = A$, $m(B_{on}) = B$, $m(C_{on}) = C$, $m(r_{Afor}) = A$, and $m(r_{Cback}) = C$. We see that: (a) any implementation reaction, when interpreted, is either trivial or $A + B \rightleftharpoons C$; (b) any non-signal implementation species (r_{Afor} and r_{Cback})

can turn into the signal species (A_{on}, B_{on}, or C_{on}) for its interpretation; and (c) the signal species can implement any reaction that their corresponding formal species should be able to do. These are the conditions of modular CRN bisimulation [22].

Because of the modularity condition [22], and because our comments on (lack of) crosstalk reactions apply between modules as well as within a module, an arbitrary number of reactions of the form $A + B \rightleftharpoons C$ can be implemented by combining the implementations for each one. Because arbitrary reversible CRNs can be implemented up to CRN bisimulation by reactions of the form $A+B \rightleftharpoons C$ and CRN bisimulation is transitive [22], this mechanism can implement arbitrary reversible reactions. To implement an irreversible CRN, use the method mentioned in Section 3 to make coupled reconfiguration irreversible by adding a toehold to the gate strand: alter $r_{Afor} + B_{on} \rightarrow r_{Cback}$ for $A + B \rightarrow C$ and $C_{on} \rightarrow r'_{Afor} + B_{on}$ for $C \rightarrow A + B$ (where r'_{Afor} is the altered version of r_{Afor} that can interact meaningfully with A_{off} but not B_{on}).

5 Pseudoknots and questioning the model

Whether any mechanism, written on paper, will actually work in the lab, is always a question. But controlled reconfiguration, in particular, depends on an assumption of the DSD model that hasn't been particularly well tested: the assumption that if a step would form a pseudoknot, that step can't happen. So while it's possible that controlled reconfiguration will work experimentally as we presented it, it's reasonably likely that it won't. However, there are reasons why, even if it doesn't work as presented in unmodified DNA, the fact that it exists in the model of DSD is still important for the study and design of DSD systems.

Traditionally the "no-pseudoknots" assumption was not a belief that pseudoknots don't happen, but a belief that we don't want to use them. RNA pseudoknots appear in many biological examples [46], and DNA nanotechnology applications such as tile assembly [18], DNA origami [13], and RNA origami [19] use fundamentally pseudoknotted structures. However, pseudoknotted structures are much less well understood: pseudoknots introduce steric contributions to energy that aren't modeled by the standard, nearest-neighbor models [49], and even with an energy model, predicting pseudoknotted structures is computationally harder [15].

For this reason, most engineered DSD systems [35] are designed such that the intended pathway does not involve pseudoknots, but also there are no non-trivial unintended pathways even when allowing pseudoknots, so that the mechanism works as intended whether or not pseudoknots are possible. In contrast, the coupled reconfiguration pathway has plenty of unintended pathways that would be possible if pseudoknots were possible, and works only under the assumption that they are not.

The first thing this result does is illustrate a difference between the model and how actual DNA behaves, suggesting ways to refine the model. For example, we might want to formally add the condition that the intended pathway has to exclude pseudoknots, but unintended pathways with pseudoknots break the system. CRN bisimulation [22] suggests one way to formalize this: the permissive condition is checked with pseudoknots banned and the delimiting condition is checked with pseudoknots allowed. So this result suggests that it would be worth investigating whether coupled reconfiguration, or single-stranded fuel-based CRNs in general, are possible or provably impossible in such a model.

Another interpretation of our result is that *if* some molecule behaves like it does in this model, *then* controlled reconfiguration can be used with that molecule. Even if unmodified DNA at standard experimental conditions forms these pseudoknots, there may be an artificial nucleic acid, or unrelated biomolecule with strand displacement-like behavior, that doesn't. If a single-stranded-fuel-based CRN implementation proves to be a particularly useful molecular device, it may be worth trying to find or design such a molecule.

Possibly relevant to this question are the experimental results of Zhang and Winfree's switchable DNA catalyst [52]. There, the catalyst switches between two states based on an input, where a hypothetical pseudoknotted spontaneous switching pathway exists. However, in that example one state is favored over the other with a one-directional non-pseudoknotted spontaneous switching pathway, and can only be held in the less favorable state when an additional strand is bound. So the experimental results may be a useful starting point for investigation, but are not conclusive in any direction on their own.

Finally, treating DSD as a model of computation, we'd like to know both how computationally powerful DSD is, and which features of the model give it which powers. This result shows us that the model with pseudoknots completely disallowed can do coupled reconfiguration, which is powerful enough to implement CRNs with single-stranded fuels. It further suggests that the ability to treat bonds as logical blockers or locked doors, preventing the two sides from interacting, is what gives the model that power.

6 Discussion and conclusions

We have introduced the mechanisms of controlled reconfiguration and coupled reconfiguration, and shown how they can be used to construct single-stranded CRN implementations. We discussed how the questionable assumption of pseudoknot-freeness means that the mechanism might not work as designed, but its existence in the model would likely have interesting implications. If the mechanism does work, either as designed or something similar, it would have a number of challenges as well as a number of benefits for DSD system design in general. Here we discuss some of the larger challenges, some of the larger benefits, and some possible aspects for further investigation.

Internal toeholds

In the model, we've assumed that m_3 and m_4 steps only happen if there are bound domains adjacent to the future reaction, usually toeholds, and that u steps can happen whenever there are no *adjacent* domains holding the toeholds together, even if the unbinding doesn't separate complexes. Both of those are not well explored, and different models have made different assumptions [21, 29]. We note that, first of all, our mechanism doesn't depend on toehold identities to prevent reactions from happening, so it should work in a model where internal m_3 steps don't need immediately adjacent toeholds, such as the one from [21]. Second, the concept of "toehold" in general is a domain of the right length to bind and unbind reversibly, and by doing so, hold things together long enough for m_3 and m_4 steps to happen. When the binding is between otherwise separate complexes, that length is 4-6 bases [53]. Internally, the length may be the same 4-6 bases, or may be more like 2 bases, but whatever length it happens to be, that length is the one we would use for the internal toeholds.

Simulating reaction rate constants

Our constructions in Sections 3 and 4 do not address how to specify rate constants for the reactions of the DSD system that simulates a CRN, so as to ensure consistency with the rate constants of the CRN. Soloveichik et al. [37] explain how this can be done within reasonable timescales, for their DSD simulation of a CRN, through toehold design and accounting for buffering effects. While the details of their simulation are quite different from ours, the key principles of their approach should also apply to our construction. We note also that many useful CRNs have the nice property of being *rate independent* in the sense that their correctness does not depend on specific reaction rate constants [6, 9, 47].

Fuel synthesis

Our construction uses single-stranded fuels, compared to the double-stranded or larger fuels of typical DSD systems. Synthesis of single strands is typically easier than synthesis of multi-stranded complexes [31]. However, a challenge is to ensure that each fuel strand is in its specified secondary structure, which is one of two or more stable (pseudoknot-free) structures for the strand. It might be possible to take advantage of co-transcriptional folding to ensure that fuel complexes have the correct structures, or it might be possible to filter out strands with incorrect structures by hybridization to a surface via a free long domain.

Single-stranded DSD

Existing single-stranded DSD systems involve single strands interacting with each other and aggregating [16, 50]. In contrast, the typical CRN implementation [37] involves reactions where (usually) one strand interacts with a larger complex, eventually staying bound and releasing (usually) one other strand, where either the new complex or the new strand (or both) is a meaningful signal that goes on to interact with the rest of the system. In particular, in those systems, most complexes are the only "stable" configuration of the strands that make them up. With single-stranded fuels that are meant to stay single-stranded, this clearly isn't possible, otherwise there would be no meaningful reactions. But if the single strands had multiple configurations that they could switch between on their own, that wouldn't implement bimolecular $A + B \rightleftharpoons C$ logic. So the novelty of coupled reconfiguration, and its ability to implement that logic, is why it makes single-stranded non-aggregative DSD systems possible.

Locality and impossibility

In our previous work on impossibility in DSD systems [21], one of the important steps was a locality theorem: that, with the restrictions in that result (notably including forbidding m_3 steps), if a sequence of unimolecular steps breaks a four-way junction, then eventually reforms the same four-way junction with the same domains, then the same result can be gotten by a sequence of steps that never breaks the junction. The coupled reconfiguration mechanism shows by counterexample that without those restrictions, and assuming pseudoknots can't be formed, the same statement would not hold. In our previous work, this theorem showed that with those restrictions, CRNs with 2-stranded fuels were impossible; with the assumptions in this work, coupled reconfiguration makes CRNs with single-stranded fuels possible. Further, to our knowledge controlled reconfiguration is the first DSD mechanism that involves a bond being initially present, broken, then reformed, with an effect that was not possible without breaking that bond. So we suspect that this concept of locality is interesting, and worth investigating whether other such mechanisms exist, especially ones that don't depend on pseudoknots being unable to be formed.

Expanding the DSD toolbox

We've expanded the DSD toolbox by introducing controlled reconfiguration and coupled reconfiguration, and shown its application to reversible DSD simulation of reversible CRNs, in a way that is easily adapted also to non-reversible CRNs. Our implementation scheme also has the following desirable properties defined in our previous work [21]: systematic with $O(1)$ toeholds, physically reversible, and using only bimolecular condensed reactions, and we have already shown that it is systematically correct under modular CRN bisimulation and uses single-stranded fuels. There are a number of CRN implementations that each fail a different one of the desirable conditions [23], and the coupled reconfiguration-based implementation fails a condition which was not listed but is also desirable, the condition of not relying on the assumption that pseudoknots don't happen. So it would be interesting to investigate whether there is a variant of coupled reconfiguration, or another implementation scheme, that fulfills the same conditions without relying on that assumption.

Further, the coupled reconfiguration mechanism reveals a capability of the current model of DSD that physical DNA strands may or may not have, either in the mechanism as we presented it or in a different mechanism with similar properties. If it does, then we would like to see if the advantages of the scheme enable physical DSD implementations with less error than existing schemes. Whether it does or not, it would be interesting to investigate the importance of the assumption that pseudoknots don't happen, and how it affects the power of DSD systems.

References

1 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, March 2006. doi:10.1007/s00446-005-0138-3.

2 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299. ACM, 2006. doi:10.1145/1146381.1146425.

3 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21:87–102, 2008. doi:10.1007/s00446-008-0059-z.

4 Stefan Badelt, Casey Grun, Karthik V Sarma, Brian Wolfe, Seung Woo Shin, and Erik Winfree. A domain-level DNA strand displacement reaction enumerator allowing arbitrary non-pseudoknotted secondary structures. *Journal of the Royal Society Interface*, 17(167):20190866, 2020. doi:10.1098/rsif.2019.0866.

5 Stefan Badelt, Seung Woo Shin, Robert F Johnson, Qing Dong, Chris Thachuk, and Erik Winfree. A general-purpose CRN-to-DSD compiler with formal verification, optimization, and simulation capabilities. In Damien Woods and Yannick Rondelez, editors, *DNA Computing and Molecular Programming*, volume 9818 of Lecture Notes in Computer Science, pages 232–248. Springer, 2017. doi:10.1007/978-3-319-66799-7_15.

6 Daniele Cappelletti, Andrés Ortiz-Muñoz, David F. Anderson, and Erik Winfree. Stochastic chemical reaction networks for robustly approximating arbitrary probability distributions. *Theoretical Computer Science*, 801:64–95, 2020. doi:10.1016/j.tcs.2019.08.013.

7 Luca Cardelli. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science*, 23(02):247–271, 2013. doi:10.1017/S0960129512000102.

8 Luca Cardelli and Gianluigi Zavattaro. On the computational power of biochemistry. In *International Conference on Algebraic Biology*, pages 65–80. Springer, 2008. doi:10.1007/978-3-540-85101-1_6.

9 Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural Computing*, 13(4):517–534, 2014. doi:10.1007/s11047-013-9393-6.

10 Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8(10):755–762, 2013. doi:10.1038/nnano.2013.189.

11 Kevin M Cherry and Lulu Qian. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, 559(7714):370, 2018. doi:10.1038/s41586-018-0289-6.

12 Samuel Clamons, Lulu Qian, and Erik Winfree. Programming and simulating chemical reaction networks on a surface. *Journal of the Royal Society Interface*, 17(166):20190790, 2020. doi:10.1098/rsif.2019.0790.

13 Swarup Dey, Chunhai Fan, Kurt V Gothelf, Jiang Li, Chenxiang Lin, Longfei Liu, Na Liu, Minke AD Nijenhuis, Barbara Saccà, Friedrich C Simmel, Hao Yan, and Pengfei Zhan. DNA origami. *Nature Reviews Methods Primers*, 1(1):1–24, 2021. doi:10.1038/s43586-020-00009-8.

14 Robert M Dirks, Justin S Bois, Joseph M Schaeffer, Erik Winfree, and Niles A Pierce. Thermodynamic analysis of interacting nucleic acid strands. *SIAM review*, 49(1):65–88, 2007. doi:10.1137/060651100.

15 Robert M Dirks and Niles A Pierce. A partition function algorithm for nucleic acid secondary structure including pseudoknots. *Journal of computational chemistry*, 24(13):1664–1677, 2003. doi:10.1002/jcc.10296.

16 Robert M Dirks and Niles A Pierce. Triggered amplification by hybridization chain reaction. *Proceedings of the National Academy of Sciences*, 101(43):15275–15278, 2004. doi:10.1073/pnas.0407024101.

17 David Doty and Monir Hajiaghayi. Leaderless deterministic chemical reaction networks. In David Soloveichik and Bernard Yurke, editors, *DNA 2013: Proceedings of The 19th International Meeting on DNA Computing and Molecular Programming*, volume 8141 of *Lecture Notes in Computer Science*, pages 46–60. Springer International Publishing, 2013. doi:10.1007/978-3-319-01928-4_4.

18 Constantine G Evans, Rizal F Hariadi, and Erik Winfree. Direct atomic force microscopy observation of DNA tile crystal growth at the single-molecule level. *Journal of the American Chemical Society*, 134(25):10485–10492, 2012. doi:10.1021/ja301026z.

19 Cody Geary, Guido Grossi, Ewan KS McRae, Paul WK Rothemund, and Ebbe S Andersen. RNA origami design tools enable cotranscriptional folding of kilobase-sized nanoscaffolds. *Nature chemistry*, 13(6):549–558, 2021. doi:10.1038/s41557-021-00679-1.

20 Anthony J Genot, David Yu Zhang, Jonathan Bath, and Andrew J Turberfield. Remote toehold: a mechanism for flexible control of DNA hybridization kinetics. *Journal of the American Chemical Society*, 133(7):2177–2182, 2011. doi:10.1021/ja1073239.

21 Robert F. Johnson. Impossibility of sufficiently simple chemical reaction network implementations in DNA strand displacement. In Ian McQuillan and Shinnosuke Seki, editors, *Unconventional Computation and Natural Computation*, pages 136–149. Springer International Publishing, 2019. doi:10.1007/978-3-030-19311-9_12.

22 Robert F Johnson, Qing Dong, and Erik Winfree. Verifying chemical reaction network implementations: A bisimulation approach. *Theoretical Computer Science*, 2018. doi:10.1016/j.tcs.2018.01.002.

23 Robert F. Johnson and Lulu Qian. Simplifying Chemical Reaction Network Implementations with Two-Stranded DNA Building Blocks. In Cody Geary and Matthew J. Patitz, editors, *26th International Conference on DNA Computing and Molecular Programming (DNA 26)*, volume 174 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DNA.2020.2.

24 Robert F Johnson and Erik Winfree. Verifying polymer reaction networks using bisimulation. *Theoretical Computer Science*, 843:84–114, 2020. doi:10.1016/j.tcs.2020.08.007.

25 Matthew R. Lakin and Andrew Phillips. Modelling, simulating and verifying Turing-powerful strand displacement systems. In Luca Cardelli and William Shih, editors, *DNA Computing and Molecular Programming*, pages 130–144, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-23638-9_12.

26 Matthew R. Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011. doi:10.1093/bioinformatics/btr543.

27 Jérôme Leroux. Vector addition systems reachability problem (a simpler solution). In *EPiC*, volume 10, pages 214–228. Andrei Voronkov, 2012.

28 Marcelo O Magnasco. Chemical kinetics is Turing universal. *Physical Review Letters*, 78(6):1190–1193, 1997. doi:10.1103/PhysRevLett.78.1190.

29 Rasmus L Petersen, Matthew R Lakin, and Andrew Phillips. A strand graph semantics for DNA-based computation. *Theoretical computer science*, 632:43–73, 2016. doi:10.1016/j.tcs.2015.07.041.

30 Lulu Qian, David Soloveichik, and Erik Winfree. Efficient Turing-universal computation with DNA polymers. In Yasubumi Sakakibara and Yongli Mi, editors, *DNA Computing and Molecular Programming*, volume 6518 of Lecture Notes in Computer Science, pages 123–140. Springer, 2011. doi:10.1007/978-3-642-18305-8_12.

31 Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011. doi:10.1126/science.1200520.

32 Lulu Qian and Erik Winfree. Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface. In Satoshi Murata and Satoshi Kobayashi, editors, *DNA Computing and Molecular Programming*, volume 8727 of Lecture Notes in Computer Science, pages 114–131. Springer, 2014. doi:10.1007/978-3-319-11295-4_8.

33 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

34 Seung Woo Shin, Chris Thachuk, and Erik Winfree. Verifying chemical reaction network implementations: A pathway decomposition approach. *Theoretical Computer Science*, 2017. doi:doi:10.1016/j.tcs.2017.10.011.

35 Friedrich C. Simmel, Bernard Yurke, and Hari R. Singh. Principles and applications of nucleic acid strand displacement reactions. *Chemical Reviews*, 119(10):6326–6369, 2019. PMID: 30714375. doi:10.1021/acs.chemrev.8b00580.

36 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008. doi:10.1007/s11047-008-9067-y.

37 David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010. doi:10.1073/pnas.0909380107.

38 Carlo Spaccasassi, Matthew R Lakin, and Andrew Phillips. A logic programming language for computational nucleic acid devices. *ACS Synth. Biol.*, 8(7):1530–1547, July 2019. doi:10.1021/acssynbio.8b00229.

39 Niranjan Srinivas, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik. Enzyme-free nucleic acid dynamical systems. *Science*, 358, 2017. doi:10.1126/science.aal2052.

40 Allison Tai and Anne Condon. Error-free stable computation with polymer-supplemented chemical reaction networks. In Chris Thachuk and Yan Liu, editors, *DNA Computing and Molecular Programming*, pages 197–218, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-26807-7_11.

41 Chris Thachuk and Anne Condon. Space and energy efficient computation with DNA strand displacement systems. In Darko Stefanovic and Andrew Turberfield, editors, *DNA Computing and Molecular Programming*, volume 7433 of Lecture Notes in Computer Science, pages 135–149. Springer, 2012. doi:10.1007/978-3-642-32208-2_11.

42 Chris Thachuk, Erik Winfree, and David Soloveichik. Leakless DNA strand displacement systems. In Andrew Phillips and Peng Yin, editors, *DNA Computing and Molecular Programming*, volume 9211 of Lecture Notes in Computer Science, pages 133–153. Springer, 2015. doi:10.1007/978-3-319-21999-8_9.

43 Anupama J. Thubagere, Wei Li, Robert F. Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, Erik Winfree, and Lulu Qian. A cargo-sorting DNA robot. *Science*, 357(6356), 2017. doi:10.1126/science.aan6558.

44 Anupama J Thubagere, Chris Thachuk, Joseph Berleant, Robert F Johnson, Diana A Ardelean, Kevin M Cherry, and Lulu Qian. Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. *Nature Communications*, 8:14373, 2017. doi:10.1038/ncomms14373.

45 Toma E Tomov, Roman Tsukanov, Yair Glick, Yaron Berger, Miran Liber, Dorit Avrahami, Doron Gerber, and Eyal Nir. DNA bipedal motor achieves a large number of steps due to operation using microfluidics-based interface. *Acs Nano*, 11(4):4002–4008, 2017. doi:10.1021/acsnano.7b00547.

46 F. H. van Batenburg, A. P. Gultyaev, C. W. Pleij, J. Ng, and J. Oliehoek. Pseudobase: a database with RNA pseudoknots. *Nucleic acids research*, 28(1):201–204, January 2000. doi:10.1093/nar/28.1.201.

47 Marko Vasic, Cameron Chalk, Austin Luchsinger, Sarfraz Khurshid, and David Soloveichik. Programming and training rate-independent chemical reaction networks, 2021. arXiv:2109.11422.

48 Boya Wang, Chris Thachuk, Andrew D Ellington, Erik Winfree, and David Soloveichik. Effective design principles for leakless strand displacement systems. *Proceedings of the National Academy of Sciences*, 115(52):E12182–E12191, 2018. doi:10.1073/pnas.1806859115.

49 Tianbing Xia, John SantaLucia, Mark E. Burkard, Ryszard Kierzek, Susan J. Schroeder, Xiaoqi Jiao, Christopher Cox, and Douglas H. Turner. Thermodynamic parameters for an expanded nearest-neighbor model for formation of RNA duplexes with Watson-Crick base pairs. *Biochemistry*, 37(42):14719–14735, 1998. PMID: 9778347. doi:10.1021/bi9809425.

50 Peng Yin, Harry MT Choi, Colby R Calvert, and Niles A Pierce. Programming biomolecular self-assembly pathways. *Nature*, 451(7176):318–322, 2008. doi:10.1038/nature06451.

51 David Yu Zhang. Cooperative hybridization of oligonucleotides. *Journal of the American Chemical Society*, 133(4):1077–1086, 2010. doi:10.1021/ja109089q.

52 David Yu Zhang and Erik Winfree. Dynamic allosteric control of noncovalent DNA catalysis reactions. *Journal of the American Chemical Society*, 130(42):13921–13926, 2008. doi:10.1021/ja803318t.

53 David Yu Zhang and Erik Winfree. Control of DNA strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society*, 131(47):17303–17314, 2009. doi:10.1021/ja906987s.

Exploring Material Design Space with a Deep-Learning Guided Genetic Algorithm

Kuan-Lin Chen ✉
Department of Chemical and Biomolecular Engineering,
Johns Hopkins University, Baltimore, MD, USA

Rebecca Schulman ✉
Department of Chemical and Biomolecular Engineering,
Johns Hopkins University, Baltimore, MD, USA
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA
Department of Chemistry, Johns Hopkins University, Baltimore, MD, USA

Abstract

Designing complex, dynamic yet multi-functional materials and devices is challenging because the design spaces for these materials have numerous interdependent and often conflicting constraints. Taking inspiration from advances in artificial intelligence and their applications in material discovery, we propose a computational method for designing metamorphic DNA-co-polymerized hydrogel structures. The method consists of a coarse-grained simulation and a deep learning-guided optimization system for exploring the immense design space of these structures. Here, we develop a simple numeric simulation of DNA-co-polymerized hydrogel shape change and seek to find designs for structured hydrogels that can fold into the shapes of different Arabic numerals in different actuation states. We train a convolutional neural network to classify and score the geometric outputs of the coarse-grained simulation to provide autonomous feedback for design optimization. We then construct a genetic algorithm that generates and selects large batches of material designs that compete with one another to evolve and converge on optimal objective-matching designs. We show that we are able to explore the large design space and learn important parameters and traits. We identify vital relationships between the material scale size and the range of shape change that can be achieved by individual domains and we elucidate trade-offs between different design parameters. Finally, we discover material designs capable of transforming into multiple different digits in different actuation states.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Computing methodologies → Modeling and simulation

Keywords and phrases Machine Learning, Deep Learning, Computational Material Design, Multi-Objective Optimization, DNA Nanotechnology

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.4

Supplementary Material *Software (Source Code)*: `https://github.com/charliecharlie29/Deep-Learning-Guided-Genetic-Algorithm`
archived at `swh:1:dir:b857ed74a5167e203a399609e9eb71bbad7cb091`

Funding This research is supported by Department of Energy under Grant No.DE-SC0010426 and by National Science Foundation under Grant No.EFRI-1830893 and by Army Research Office under Grant No.W911NF2010057.

Acknowledgements We would like to thank the Maryland Advanced Research Computing Center (MARCC) for providing cloud computing GPU services.

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham; Article No. 4; pp. 4:1–4:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Combinatorial Metamorphic Materials

The structure of a device determines its function. It is interesting to ask how we might manufacture *combinatorial metamorphic* devices that can take on many different forms and functions in response to a wide range of triggers. As metamorphic devices are capable of achieving a wide range of potential functions, they allow users to choose a function fit for a particular task and to alter the shape of the device to access that function. Because one such structure could be transformed into a large set of target devices, a single metamorphic structure can be stored or carried in place of a large set of traditional devices. These reasons make metamorphic devices more flexible and versatile than their traditional counterparts. These advantages have stimulated researchers from a range of communities to explore the design and construction of metamorphic devices[1, 3, 17, 21] for applications such as soft robotics[10, 21] and even quantum computing[5].

DNA-co-polymerized Hydrogels as Metamorphic Materials

Here we consider a means to construct metamorphic devices in which specific biochemical cues, DNA sequences, trigger the shape change of specific hydrogel domains[2]. Each hydrogel domain contains DNA crosslinks that allow the material to expand into swollen state upon activating the domain with a specific DNA sequence (growing actuators) and contract into shrunken state upon de-activation with another DNA trigger (shrinking actuators). To understand how we might use these types of materials to create metamorphic structures, we ask how to design material structures from 4 sets of the following active materials: hydrogels with a specific set of DNA crosslinks, growing actuators and shrinking actuators (Figure 1). Using multistage photolithography[2, 8, 18] to assemble these materials should allow us fabricate metamorphic devices that react to biomolecular signals with high specificity.

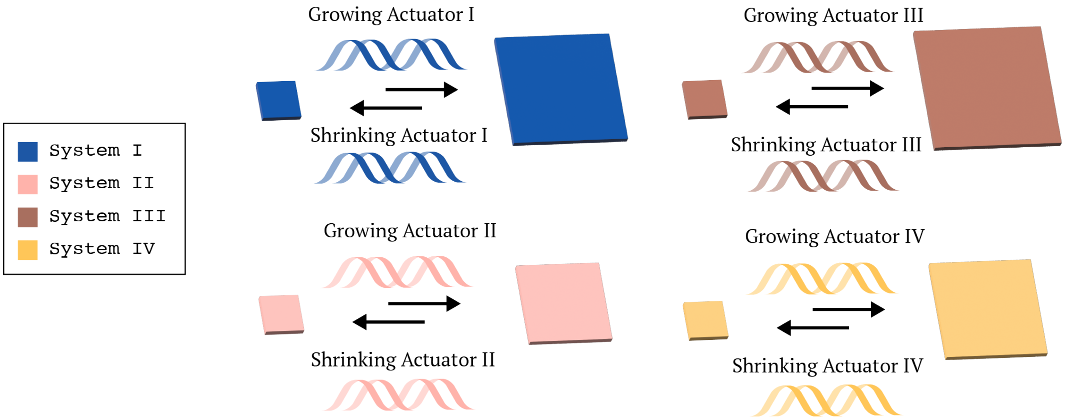


Figure 1 DNA-co-polymerized hydrogel. A set of 4 different materials that can be enlarged and contracted by DNA signal. Exposure to its shrinking signals prompts a material to enter its shrunken state, while exposure to its growing signal prompts a material to enter its swollen state. The signals are orthogonal (each affects only its target material and not others). Each material's expanded and shrunken states are of somewhat different sizes.

Metamorphic Multi-state Digit Transformer

DNA-co-polymerized hydrogels whose size can be bidirectionally controlled by DNA signals bring forth the possibility of building complex metamorphic materials. To explore this concept, we seek to investigate whether it would be possible to use the materials in Figure 1 to design a *metamorphic digit transformer*. We ask how one might build metamorphic devices using the 4 active materials with orthogonal DNA actuation systems and a passive material without a DNA actuation system. Each actuator system drives a specific domain into swollen state upon activation, and shrunken state when inactive. As a result, the device is multi-stable and has 16 possible states ($2^4 = 16$) when all DNA actuators are used. Our goal is to find a design where as many of its final outputs as possible resemble the shape of different digits from 0 to 9. We consider concatenated segments of bilayer hydrogel segments as a design space. The overall outline of this process is shown in Figure 2. Because these devices must be fabricated using multistage photolithography, the resulting designs must be consistent with this mode of fabrication, which imposes physical limits on the design space. Each bilayer segment will typically be on order 500μm in height and 250μm in width. In order for the curvature of the resulting structures to be governed by curvature along the lengthwise, segmented axis, the overall length of the structure must be significantly longer than either the structure's width or height. Lithography also limits the sizes of the segments to be between 50μm and 5000μm, below which size resolution becomes difficult to control, and above which would require a larger light source and mask. Moreover, we want to limit the number of distinct fabrication steps required, as difficulties such as alignment, device lift-off, and transfer increase rapidly with the number of fabrication steps.

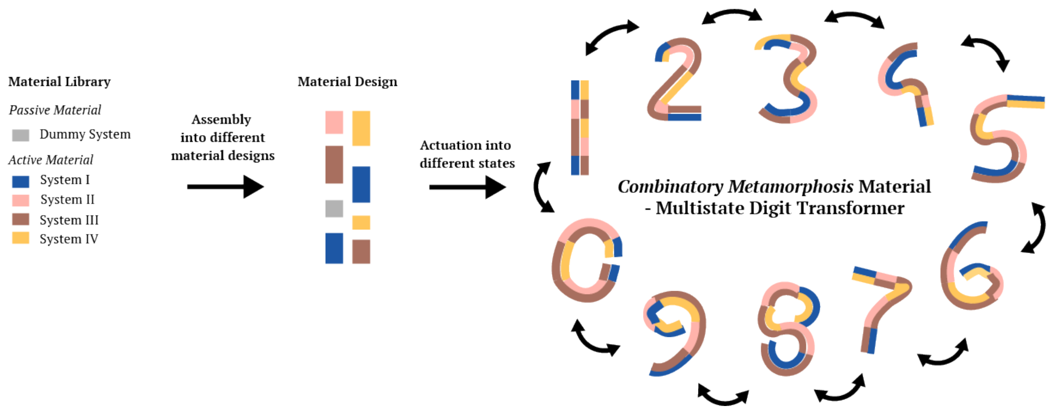

Material Library
Passive Material
Dummy System
Active Material
System I
System II
System III
System IV
Assembly into different material designs
Material Design
Actuation into different states
Combinatory Metamorphosis Material - Multistate Digit Transformer

Figure 2 Building a metamorphic digit transformer from DNA-co-polymerized hydrogels. A metamorphic digit transformer is a stack of bilayers of DNA-co-polymerized hydrogels. The digit transformer would be able to change its shape into the shape of different digits upon the activation of different biochemical actuation programs, which switch the conformation of the structure between one of multiple stable states.

Machine Learning and Genetic Algorithm for Material Discovery

Finding a design for a structure with a large number of different forms is a challenge. This challenge increases as the number of design objectives (*i.e.* stable states) increases, and tighter physical and manufacturing limits are imposed. The design space, consisting of the types, arrangements, and lengths of hydrogel segments is too large to search through *via* trial and error, and the conflicting nature of the constraints also makes structured design methods challenging to consider. Inspired by how machine learning and artificial intelligence techniques have been transformative in different areas of material design and discovery[9, 11, 14, 20], we seek to design digit transformer devices by developing a computational material discovery method. The method we describe mimics Darwinian evolution[4, 19] (through the use of a genetic algorithm) and combines numeric simulation[15] with state-of-the-art machine-learning models[16]. The resulting system simulates and autonomously evolves generations of material design variants *in-silico* to find designs for devices that satisfy the multiple design objectives[7] we created (Figure 2).

2 Methods

2.1 Material Simulation

To develop a geometric simulation platform for our DNA-co-polymerized hydrogel, we considered predicted values of changes in contour length (ΔL) and radii of curvature (RoC) during the expansion and contraction of bilayers consisting of different combinations of actuator types. The simulation of the material starts with defining a straight bilayer structured hydrogel with specified length and actuator pattern. We then look up the different values for RoC and ΔL given the target actuated state. The final folded shape is then calculated using the segment length, derived RoC and ΔL for each segment (Figure 3, top). We assume there is little stress along the horizontal axis and that the shape of the resulting structure is achieved by a linear "stack" of different segments then concatenate with one another to form a smooth curve (Figure 3, bottom).

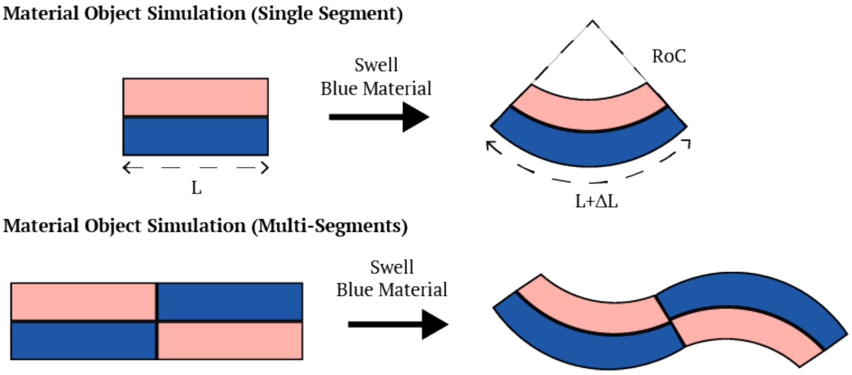

Material Object Simulation (Single Segment)
Swell
Blue Material
RoC
L
L+ΔL
Material Object Simulation (Multi-Segments)
Swell
Blue Material

Figure 3 Simulation of DNA-co-polymerized bilayer hydrogel segment(s). The shape of a folded single segment is determined by the segment's length and the two types of actuators that make up the segment and their actuation states (expanded or contracted), as the states determine the overall values for the change in radius of curvature and contour length. Here, the bilayer studied has system I (blue) in its contracted state in the bottom segment and system II (pink) in its expanded state in the top segment. In a device with multiple segments, simulation is done with the assumption that there exists little stress between the segments so that the final conformation is the integrated sum of each single segment simulated independently.

2.2 Simulation of Device Curving

To construct a simulated "device," we create a list of segment lengths and a matrix of actuator types and their states. This design is encoded as a list S of segment lengths and a list P that encodes the actuator pattern (top then bottom) within each segment.

$$S = \{L_1, L_2, ..., L_n\} \tag{1}$$

$$P = \{\{X_{11}, X_{12}, ..., X_{1n}\}, \{X_{21}, X_{22}, ..., X_{2n}\}\} \tag{2}$$

where

$$X \in \{0, 1, 2, 3, 4\}$$

with 0 representing the passive system and 1 to 4 representing the 4 active materials.

A device's design is determined by simulating the curving of the device in all 16 possible states (where each of the four actuator types is in each of the two distinct states). An example of the possible states of a typical design is shown in Figure 4.

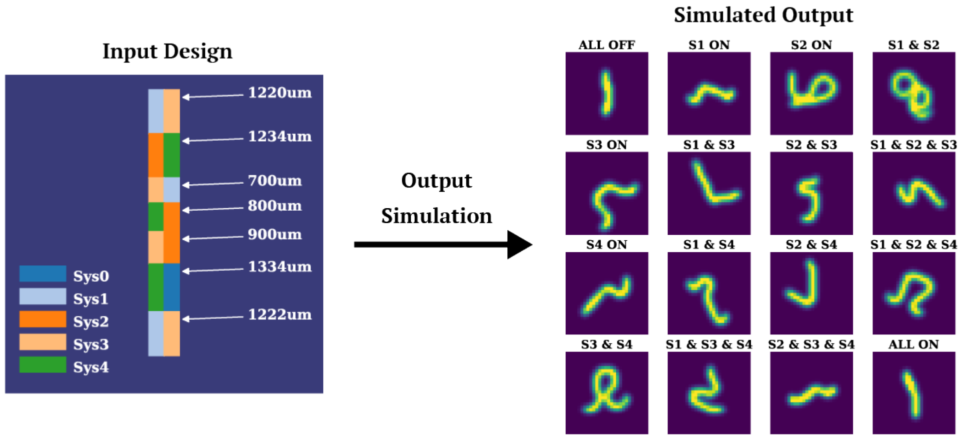


Figure 4 Example of a simulation of a device's curving in each of its 16 states. Shown are the input design and a map of the predicted output shapes. In the input section, different colors represent different actuators and the length of each bilayer segment is shown to the segment's right. The output diagrams show the predicted device shapes of all 16 possible actuation states. The label above each image specifies the actuator state (S2 designates system II, and so on, while ON designates the swollen state and OFF designates the shrunken state). For scoring purpose, the shapes are plotted in 28-by-28-pixels-images that are treated with a Gaussian blur filter ($\sigma = 1$).

2.3 Deep Learning Model for Design Selection

To efficiently automate the scoring and selection process for design optimization, we train a convolutional neural network (CNN) classifier[13] to distinguish the digit-similarity of the predicted geometric outputs. We use the *Tensorflow* library and train the model using a combinatory dataset consisting of the MNIST dataset[6] and an artificial dataset. We label twenty-four thousand images generated with the simulation platform manually to build an artificial dataset and add an additional class (class "10") for images that do not look digit-like and instead look like random squiggles. This additional class helps the model recognize bad shapes and images that the hydrogel device most often bends into. We train a sequential convolutional neural network consisting of two 2D convolutional layers (with 30 and 15 filters) with 2D max-pooling layers following each convolutional layer, and three fully connected layers (with 128, 50, and 11 nodes) and train on the combinatory data-set. The *relu* activation is used in all layers except for the final classification layer, where the *softmax* function is used. We used the *adam* optimizer[12] with categorical cross-entropy loss function and trained for 50 epochs. This model is then used to score and select ideal outputs that resemble digits. During the scoring and selection process, we rotate the images to explore the full potential of the designs.

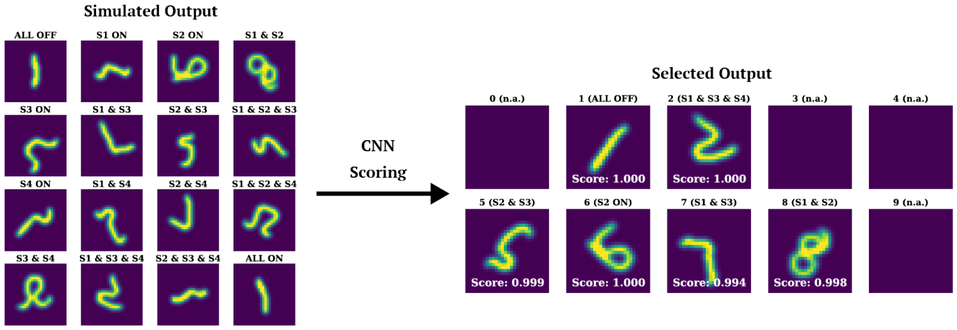


Figure 5 Example of design output scoring and selection. After the simulation, 16 images are generated and scored by the CNN to determine whether each represents a digit or not. Note that the final classification layer of the CNN model has 11 nodes where the first 10 represent the score (or probability) of resemblance to digit from 0 to 9. The 11-th value represents the value of resemblance to non-digit-like images. We use the max value of each class to represent the result of each image and discard the result if the 11-th value is the maximum - meaning the CNN model determines that this image is highly unlikely to represent any digits. Thus, only images with "max digit probability" are selected after the scoring and selection process of the CNN model.(Note that notations for simulated outputs are same as shown in Figure 4.)

2.4 Genetic Algorithm

We develop a genetic algorithm to evolve the material designs autonomously. The algorithm works by initiating a large batch of random designs with a fixed population size. At each iteration, the whole population is simulated and scored with the convolutional neural network(CNN) and a multi-objective loss function to determine the fitness score of each design.

2.4.1 Loss Function - Fitness Evaluation

We tried a variety of different loss function to track and optimize design:

Fitness Evaluation on Digit Quality Alone

We initially started the algorithm based on a simple loss function where only the digit quality is tracked and scored. This, however, leads to issues where designs that form a wide range of "mediocre digits" cannot compete with designs that form only a few number of "perfect digits", and we lose these designs throughout the evolution trajectory. While it is more likely for these "mediocre designs" to evolve and grow into designs that can fold into all digits, this loss function reduces the survival chance for them and thus are not ideal for the search.

$$fitness = \sum_{i=0}^{9} log(1 - V_i) \tag{3}$$

where

V_i: the score of each digits where $i \in \{0, 1, 2, ..., 10\}$

Fitness Evaluation on Digit Diversity and Quality

Moving forward, we improved the loss function so that we evaluate the performance based on the diversity of the digits formed as well as the quality of different digits. The diversity is evaluated with α, where we count how many digits a design formed. To calculate α for each design, we iterate through the classification results of the outputs and count how many of them are classified as digits (with *softmax*-ed classification value ranging from 0 to 9). α is the number of non-repeating digits formed. The quality of different digits is evaluated and stored in the list V, and $V_i = 0$ when the ith digit is not found. With this method we are more likely to find better designs and this loss function is used throughout the rest of the paper.

$$fitness = \sum_{i=0}^{9} log(1.0001 - V_i) \cdot \alpha \tag{4}$$

where:

V_i: the score of each digits where $i \in \{0, 1, 2, ..., 10\}$
α: diversity coefficient, the number of digits formed

2.4.2 Mutation Function

Once the whole population is scored, the population is then sorted according to the calculated fitness and 80% of the population is eliminated. The survivor designs are then delivered to a mutation function where we use the single-parent-mutation method to preserve the gene of each design and produce offspring. Each survivor design produces four offspring and sent with their offspring to the next generation to compete. This way the size of the population remains fixed throughout the evolution process. The iteration cycle continues until we reach the maximum generation limit.

Mutation Function - Vanilla Form

Mutation function is where we determine how the genetic information of the parent design is passed down to the offspring. The gene G in our designs consists of two matrices, the segment list S and the actuator pattern matrix P, such that $G = G(S, P)$. In the vanilla form of the mutation function, we randomly update either the S or P to mutate the genetic information, where each has a 50% chance of mutation. For the S-mutation offspring, we randomly assign new S while preserving the P. For the P-mutation offspring, we randomly swap out 50% of the genes within the pattern while preserving the S. Note that instead of swapping out all the genes, only 50% of them are mutated to ensure enough of the genetic information is maintained.

Mutation Function - With Fabrication Limit

Additionally, we use a more advanced mutation function that accounts for the fabrication complexity to ensure that the converging designs are within reasonable physical limits. The form of the function looks pretty similar to the vanilla form above except for additional fabrication step calculation. We define the fabrication steps as the steps needed to pattern these gels with the photolithography setup. The steps needed depends on the sum of different actuators on each side of the gel. For example, if we have a design that has actuator [1, 2, 1, 3, 1] on the one side and [4, 2, 2, 4, 2] on the other, it takes 3 steps to pattern the first side given 3 different actuators used and 2 steps on the other. The total fabrication steps would be five to pattern these devices. The new mutation function now calculates the steps needed when the P is updated and rejects the P if it exceeds the maximum step allowed. The function would then reassign P and check and iterate until it converges on a new P that satisfies the fabrication limit.

Listing 1 Pseudo-code of the deep-learning guided genetic algorithm.

```
initialize first generation of designs

for i in range(generation_limit):
    fitness scoring with CNN model and loss function

    eliminate 80% of the population

    mutate survivors and generate descendants
```

Finally, we select the top 5 survivor designs to be the optimum converged designs in each evolution tree.

2.5 Search Evaluation

At the end of every evolution tree, the 5 final converged designs are saved and manually scored for an objective scoring; this is to mediate the possibility of false-positives from the convolutional neural net and to assure that the final outputs are digit-like to both machines and human. Figure 6 shows an example of the evaluation process, where each image gets a score of 1 (labeled with a green box) if it looks like a perfect digit, 0.5 (labeled with a blue box) if it looks similar but not perfect and 0 if unrecognizable. This is used as the final subjective-matrix to ensure the convergence of the model is reasonable.

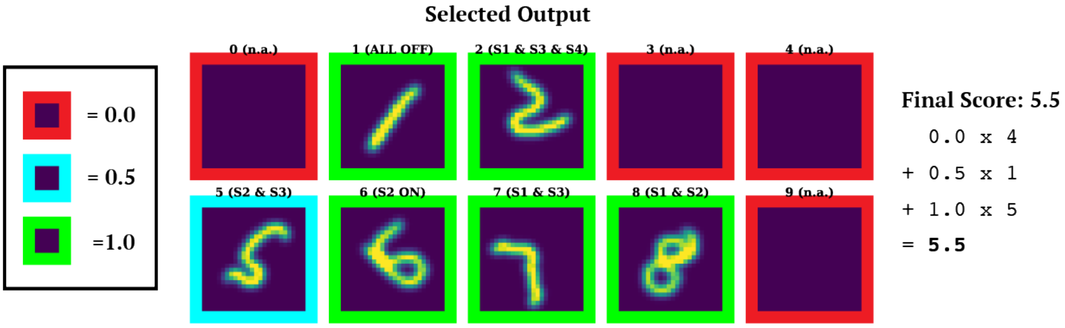


Figure 6 Example of manual(human-scored) evaluation. All converged outputs are manually scored to provide a final score for each design. In the example, the design forms 5 perfect digits (marked in green), 1 recognizable digit (marked in blue) and is unable to form other digits (marked in red). The final score is thus 1 x 5 + 0.5 x 1 = 5.5.

3 Results and Discussions

We then deployed the algorithm on the search for an optimal design. During the search, we seek to explore the parameter space and learn the effect of different hyper-parameters including - degree of freedom, design scale, search duration, and fabrication limit on the ability to find optimal design. As the searching process itself is stochastic and no independent event is repeatable, we evaluate the search of each condition with 3 separate evolution trees and aggregate the information to avoid an independent event being an outlier. After the search, all converged designs are manually scored to assure the accuracy of evaluation. The distribution of final scores is used as a final metric, along with the loss trajectory, to help us determine whether a condition is ideal or not.

3.1 Degree of Freedom: Number of Segments

We first seek to learn the effect of how the number of segments affects the material's ability in finding digit transformer designs. During the search, we fixed the parameter of the total length to be roughly 8000μm and the search duration to be 100 generations. We ran the search of different segment lengths from 2 to 100 with 3 separate evolution trees for each condition. The result of the loss trajectory is shown in Figure 7 as an objective matrix to evaluate the effect of different segment numbers from the machine's perspective. The manually scored results of the converged designs of each condition are shown in Figure 8 as a subjective matrix for evaluation.

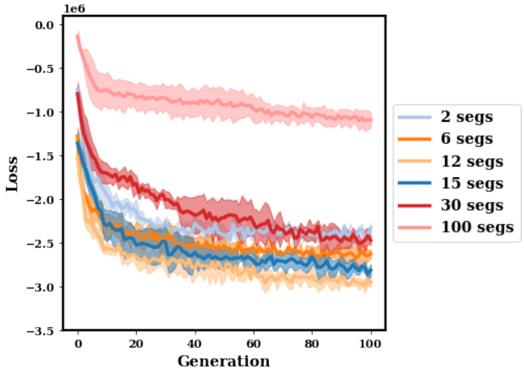


Figure 7 Loss trajectory of search on different segment number.

The number of segments is analogous to the degree of freedom of the system. More segments means there are more knobs to tune during the search and, theoretically speaking, better performance of the algorithm. The result, however, shows that this is only true within a certain extent. There exists a sweet-spot, and exceeding this region actually impales the ability to find good designs, as shown in Figure 8. We believe that this is because we are also increasing the complexity of the problem when we raise the number of segments used in the search, and after a certain amount of freedom is introduced, the benefit of having more knobs is out-weighted by the increase in complexity. Currently, our method and algorithm are unable to find good designs when the system is too complex. From this we learn the importance of maintaining the balance between degree of freedom and problem complexity when we are solving a problem with models.

3.2 Design Scale

Next we seek to learn the effect of the design scale on search performance. During the search, we fixed the parameter of number of segments used to 12 and the search duration to 100 generations. We ran the search of different total length scale from 800μm to 80,000μm with 3 separate evolution trees for each condition. The result of the loss trajectory is shown in Figure 9 as an objective matrix to evaluate the effect of different segment numbers from the machine's perspective. The manually scored results of the converged designs of each condition are shown in Figure 10 as a subjective matrix for evaluation.

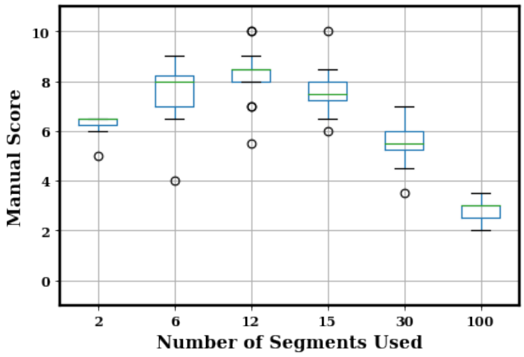


Figure 8 Score distribution of search on different segment number.

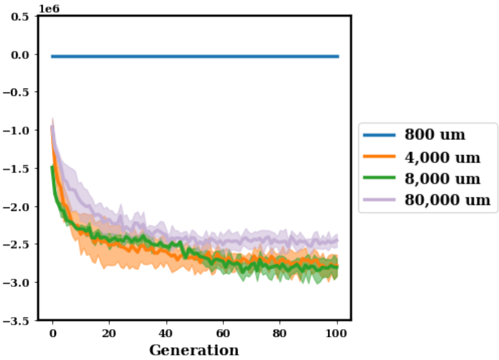


Figure 9 Loss trajectory of search on different length scale.

During the search, we learned that the length scale played an important role in the performance and that there also exists a sweet spot in the length scale when searching for an optimal design. We found out that when the length scale of the material is too small, the devices are unable to bend into large angles as the folding power the actuators provide is not enough. This limits the outputs of the devices to be simply straight or slightly bent 1s, and stops the algorithm from finding interesting designs. This is why the short length scale condition (800μm) only get score 1s in Figure 10, and we can see the algorithm is unable to optimize anything at all inspecting the loss trajectory in Figure 9. The search only starts becoming interesting and meaningful when the scale is large enough at 4,000μm, but the performance starts to decrease again when the scale becomes too large. This is because the devices are also more likely to misfold into undesired “random squiggles” when they are too long, which also corresponds to the folding power the actuators provide and the problem we are trying to solve. With this search we are able to locate the ideal length range for our devices given our actuator power and our target objectives.

3.3 Search Duration

Another parameter we changed during the search is the duration of the evolution. We fixed the parameter of number of segments used to be 12, the length scale to be 8,000μm, and search duration to be 150 generations with 3 separate evolution trees. We harvest and save the top 5 designs every 30 generations so we can track the manual scores at different evolution stages. The result of the loss trajectory is shown in Figure 11 as an objective matrix

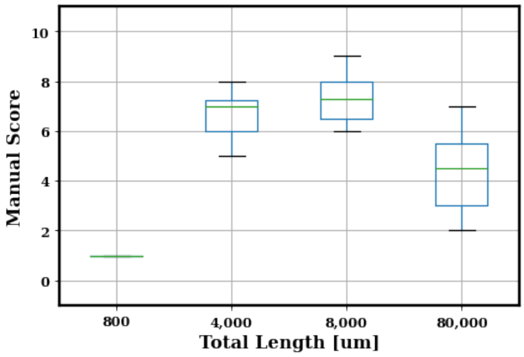


Figure 10 Score distribution of search on different length scale.

to evaluate the effect of different segment numbers from the machine's perspective. The manually scored results of the converged designs of each condition are shown in Figure 12 as a subjective matrix for evaluation.

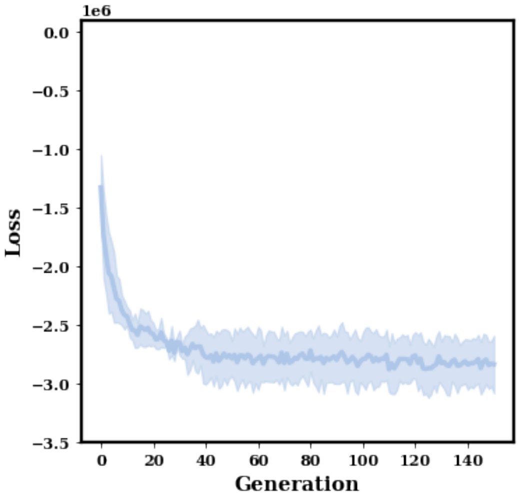


Figure 11 Loss trajectory of search on different search duration.

Here, we learn that designs start becoming meaningful quite early in the search, this makes sense as the algorithm is programmed to preserve all good designs found along the evolution path. The search also starts converging toward the minimum at around 90 to 120 generations, which is also why we set most search durations to 100 in other conditions. A more interesting point, however, is that we do not want the search to go on infinitely either. While the loss stopped decreasing after a certain amount of generations passed, we also observed a decrease in "gene diversity" when the search became too long. Here, we define "genes" $G(S, P)$ as the component that makes up the design input - the segment list S and pattern matrix P. We found out that as the search becomes too long, the genes especially in P start to become less diverse, with many survivors sharing similar P and the search process becoming a randomized S swapping search with little optimization going on. Moving forward, it may be helpful in future searches to add a gene diversity evaluation tracker within the search and program the mutation rate to change adaptively in response to the gene diversity.

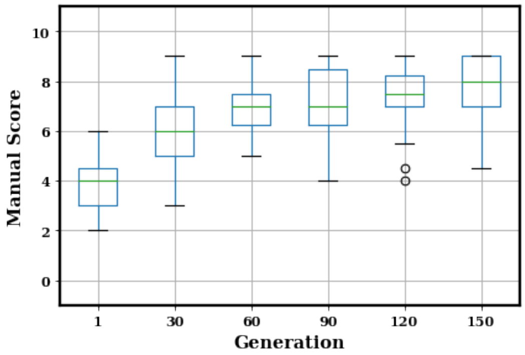


Figure 12 score distribution of search on different search duration.

3.4 Fabrication Limit

Finally, we investigated how fabrication limit affects the search performance, as it is also vital that the designs found should not exceed the physical patterning limit. Currently, we are imposing a 6 step limitation on the search, as exceeding this value increases the difficulty patterning the devices. During the search, we fixed the length scale to be 8,000μm and search duration to be 100 generations with 3 separate evolution trees for the two conditions - 6-segment designs and 12-segment designs. We compared the results with the same conditions shown above to learn how fabrication limit affects the search performance.

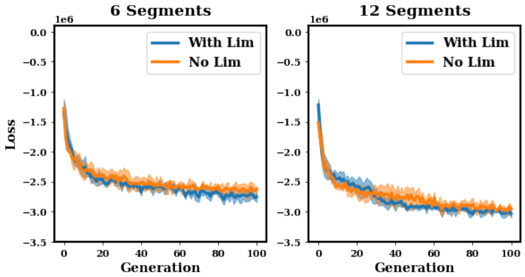


Figure 13 Loss trajectory of search on different fabrication limit.

We showed that the fabrication limit imposed does not drastically change the behavior or the performance of the search in our given conditions. While the limit does slightly decrease the performance of the optimal designs for the 12-segment search, we do think it is still possible to find a design that can fold into all digits given more evolution trees deployed for a wider and larger search. It is therefore encouraging that we now have a computational material discovery method that can autonomously learn and evolve the material designing process while considering real-world physical limitations.

4 Conclusion

Here, we demonstrate the development of a computational material discovery method for a multi-stable soft material with orthogonal actuators and automate the multi-objective optimization process with a genetic algorithm and integration of a deep-learning model. We show that we are able to efficiently explore the large parameter space and learn the effect of different variables. We also show that we can impose real-world physical constraints to discover reasonable designs.

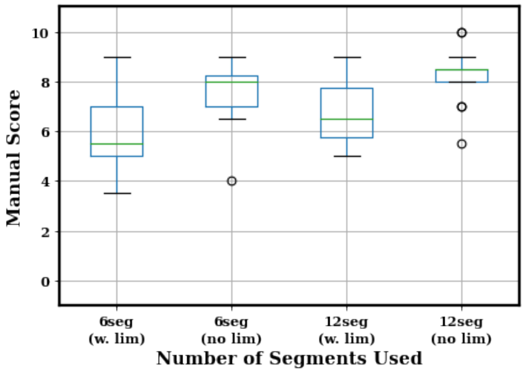


Figure 14 Score distribution of search on different fabrication limit.

In future work, one could expand the dimension of the simulation platform to handle more complex material actuation simulation. Building on this structure, it could also be possible to develop an advanced discovery platform that can optimize the functions of DNA-actuated hydrogel designs for tasks such as building locomotive robots.

References

1 Dhiraj Bhatia, Christian Wunder, and Ludger Johannes. Self-assembled, programmable dna nanodevices for biological and biomedical applications. *ChemBioChem*, 22(5):763–778, 2021.

2 Angelo Cangialosi, ChangKyu Yoon, Jiayu Liu, Qi Huang, Jingkai Guo, Thao D. Nguyen, David H. Gracias, and Rebecca Schulman. Dna sequence–directed shape change of photopatterned hydrogels via high-degree swelling. *Science*, 357(6356):1126–1130, 2017. doi:10.1126/science.aan3925.

3 VF Cardoso, C Ribeiro, and S Lanceros-Mendez. Metamorphic biomaterials. *Bioinspired Materials for Medical Applications*, pages 69–99, 2017.

4 N. Chakraborti. Genetic algorithms in materials design and processing. *International Materials Reviews*, 49(3-4):246–260, 2004. doi:10.1179/095066004225021909.

5 Peter W Deelman, Lisa F Edge, and Clayton A Jackson. Metamorphic materials for quantum computing. *MRS Bulletin*, 41(3):224–230, 2016.

6 Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

7 Abhijith M Gopakumar, Prasanna V Balachandran, Dezhen Xue, James E Gubernatis, and Turab Lookman. Multi-objective optimization for materials discovery via adaptive design. *Scientific reports*, 8(1):1–12, 2018.

8 Mustapha Jamal, Sachin S Kadam, Rui Xiao, Faraz Jivan, Tzia-Ming Onn, Rohan Fernandes, Thao D Nguyen, and David H Gracias. Bio-origami hydrogel scaffolds composed of photocrosslinked peg bilayers. *Advanced healthcare materials*, 2(8):1142–1150, 2013.

9 Paul C Jennings, Steen Lysgaard, Jens Strabo Hummelshøj, Tejs Vegge, and Thomas Bligaard. Genetic algorithms for computational materials discovery accelerated by machine learning. *NPJ Computational Materials*, 5(1):1–6, 2019.

10 Michał Joachimczak, Reiji Suzuki, and Takaya Arita. Artificial metamorphosis: Evolutionary design of transforming, soft-bodied robots. *Artificial life*, 22(3):271–298, 2016.

11 Yongfei Juan, Yongbing Dai, Yang Yang, and Jiao Zhang. Accelerating materials discovery using machine learning. *Journal of Materials Science & Technology*, 79:178–190, 2021. doi:10.1016/j.jmst.2020.12.010.

12 Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*, 2014. arXiv:1412.6980.

13 Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi:10.1109/5.726791.

14 Yue Liu, Tianlu Zhao, Wangwei Ju, and Siqi Shi. Materials discovery and design using machine learning. *Journal of Materiomics*, 3(3):159–177, 2017. doi:10.1016/j.jmat.2017.08.002.

15 Arun Mannodi-Kanakkithodi and Maria KY Chan. Computational data-driven materials discovery. *Trends in Chemistry*, 3(2):79–82, 2021.

16 Tarak K. Patra, Venkatesh Meenakshisundaram, Jui-Hsiang Hung, and David S. Simmons. Neural-network-biased genetic algorithms for materials design: Evolutionary algorithms that learn. *ACS Combinatorial Science*, 19(2):96–107, 2017. doi:10.1021/acscombsci.6b00136.

17 Anjali Rajwar, Sumit Kharbanda, Arun Richard Chandrasekaran, Sharad Gupta, and Dhiraj Bhatia. Designer, programmable 3d dna nanodevices to probe biological systems. *ACS Applied Bio Materials*, 3(11):7265–7277, 2020.

18 Ruohong Shi, Joshua Fern, Weinan Xu, Sisi Jia, Qi Huang, Gayatri Pahapale, Rebecca Schulman, and David H Gracias. Multicomponent dna polymerization motor gels. *Small*, 16(37):2002946, 2020.

19 Changwon Suh, Clyde Fare, James A Warren, and Edward O Pyzer-Knapp. Evolving the materials genome: How machine learning is fueling the next generation of materials discovery. *Annual Review of Materials Research*, 50:1–25, 2020.

20 Rama Vasudevan, Ghanshyam Pilania, and Prasanna V Balachandran. Machine learning for materials design and discovery, 2021.

21 Zhi Zhao, Chao Wang, Hao Yan, and Yan Liu. Soft robotics programmed with double crosslinking dna hydrogels. *Advanced Functional Materials*, 29(45):1905911, 2019.

Modelling and Optimisation of a DNA Stack Nano-Device Using Probabilistic Model Checking

Bowen Li
Interdisciplinary Computing and Complex bioSystems (ICOS) Research Group,
School of Computing, Newcastle University, Newcastle upon Tyne, UK

Neil Mackenzie
Interdisciplinary Computing and Complex bioSystems (ICOS) Research Group,
School of Computing, Newcastle University, Newcastle upon Tyne, UK

Ben Shirt-Ediss
Interdisciplinary Computing and Complex bioSystems (ICOS) Research Group,
School of Computing, Newcastle University, Newcastle upon Tyne, UK

Natalio Krasnogor
Interdisciplinary Computing and Complex bioSystems (ICOS) Research Group,
School of Computing, Newcastle University, Newcastle upon Tyne, UK

Paolo Zuliani
Interdisciplinary Computing and Complex bioSystems (ICOS) Research Group,
School of Computing, Newcastle University, Newcastle upon Tyne, UK

Abstract

A DNA stack nano-device is a bio-computing system that can read and write molecular signals based on DNA-DNA hybridisation and strand displacement. *In vitro* implementation of the DNA stack faces a number of challenges affecting the performance of the system. In this work, we apply probabilistic model checking to analyse and optimise the DNA stack system. We develop a model framework based on *continuous-time Markov chains* to quantitatively describe the system behaviour. We use the PRISM probabilistic model checker to answer two important questions: 1) What is the minimum required incubation time to store a signal? And 2) How can we maximise the yield of the system? The results suggest that the incubation time can be reduced from 30 minutes to 5-15 minutes depending on the stack operation stage. In addition, the optimised model shows a 40% increase in the target stack yield.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases probabilistic model checking, CTMC, DNA computing, DNA stack

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.5

Supplementary Material *Software (Source Code)*: `https://github.com/shelllbw/mcSTACK`
archived at `swh:1:dir:7b4a4cbb0e4f6ab425aedd6066c2a221f7dffebe`

Funding This work was partly funded by the EPSRC's project "Synthetic Portabolomics: Leading the way at the crossroads of the Digital and the Bio Economies" (EP/N031962/1). Krasnogor was supported by the Royal Academy of Engineering under the Chairs in Emerging Technologies scheme.

Acknowledgements The authors would like to thank Dr. Harold Fellermann for helpful advice.

1 Introduction

DNA computing is an emerging field that aims to use DNA, biochemistry, and molecular biology to construct information-processing devices. Since its first appearance where DNA was used for encoding the directed Hamiltonian path problem [1], the approach has been applied to design a wide range of biocomputing applications such as molecular computational circuits [7, 22], DNA storage technologies [9, 21], and synthetic controllers [8]. In [20], we presented the design and *in vitro* implementation of a DNA stack nano-device that can read and write molecular signals in a last-in first-out way. The stack data structure is implemented as a linear chain of partially complementary DNA strands which are used to represent both data and read/write operations. The operations are achieved via DNA-DNA hybridisation and strand displacement of complementary toehold domains. Although experimental results show a generally successful implementation, the system still faces a number of challenges affecting the performance of the resulting stack. That is in particular due to the presence of unwanted interactions among the biochemical species that implement the system.

To improve the performance of the DNA stack system, we propose the use of probabilistic model checking [4, 17] in the stack design process. Probabilistic model checking is a formal verification technique for modelling and analysing stochastic (probabilistic) systems. A benefit of the technique is the ability to exhaustively explore finite-state probabilistic models, typically Markov chains or variants. Since all possible behaviours of the system are analysed, the result is more reliable (and faster in some cases) than stochastic simulation where one simulation run follows a single possible trajectory of the system. Although probabilistic model checking was originally applied to software verification, the technique has been successfully used in biological settings as well [6, 14, 15, 19]. By quantitatively evaluating properties such as *"what is the probability for the system to reach steady-state within 10 minutes"*, we are able to predict the behaviour and optimise the design of a complex biochemical system.

In this work, we develop a probabilistic model framework based on *continuous-time Markov chains* (CTMCs) for describing the DNA stack system presented in [20]. We address two important challenges of the system: how to minimise the incubation time, and how to maximise the yield of the target stack. A set of properties based on the Continuous Stochastic Logic (CSL) are defined to quantitatively and rigorously analyse the dynamic behaviour of the DNA stack model. By model checking these properties we optimise the experimental protocol for constructing the DNA stack *in vitro*. The probabilistic model checker PRISM [18] is used for the model development and analysis.

The rest of paper is organised as follows: Section 2 briefly introduces the concept of the DNA stack system, its experimental implementation, and CTMC-based probabilistic model checking. Section 3 describes the computational model of the DNA stack system, while Section 4 presents several validation and model checking results. Section 5 discusses experimental validation of the model checking results and concludes the paper.

2 Background

2.1 DNA stack system

We give here a brief overview of the DNA stack system – more details can be found in [12, 16, 20]. A computational stack is an abstract data type that serves as a linear collection of data elements, with two main operations: *push* (adds an element to the *head* of collection) and *pop* (removes from the *head* of the collection and returns the most recently added element). Because of this LIFO (Last In, First Out) feature, stacks are much used for enforcing sequential access to data.

A DNA stack is a *molecular* implementation of the stack data structure where the data elements that are stored, as well as the operations needed to operate the stack, are engineered via a set of single stranded DNA strands, also called "DNA data operators"; DNA data operators use both sequence and (partial) secondary structures information to achieve the functionality by mimicking their computational counterpart. The construction of a DNA stack structure is achieved via DNA-DNA hybridization and strand displacement of complementary

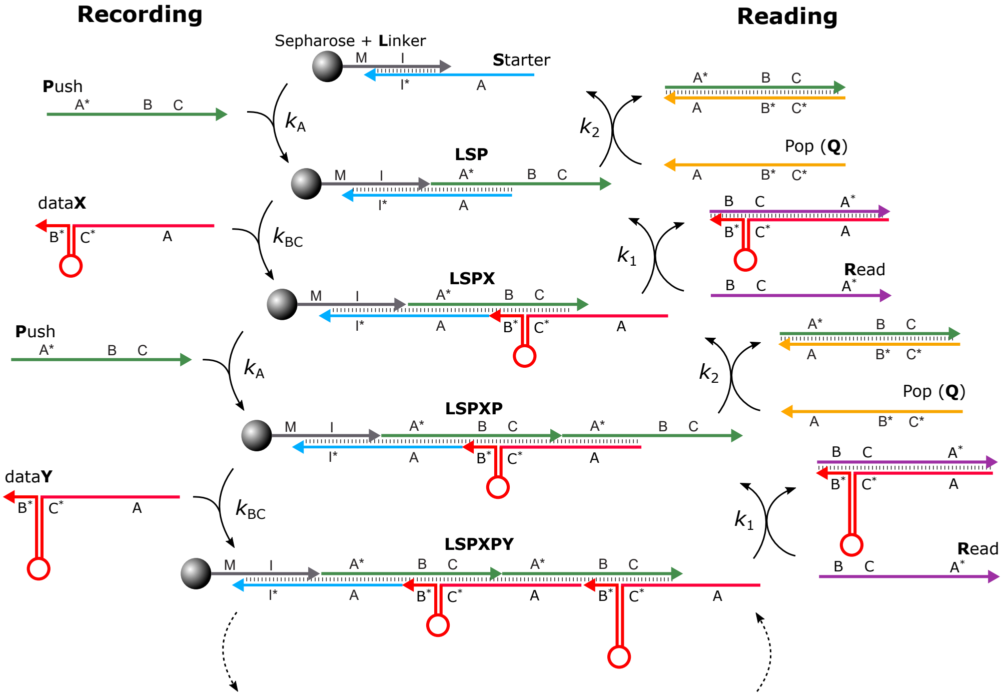


Figure 1 Schematic of the DNA stack nano-device.

toehold domains. In a DNA stack implementation, the push operation prepares the stack for receiving an element, while pop undoes that. To record an element in the DNA stack, one performs a push operation followed by a data recording operation. To reverse this, a specific read operation is applied that removes the last element added, followed by a pop operation which undoes push and returns the stack to its initial state.

Figure 1 gives a schematic representation of a DNA stack system that performs two push and two pop operations for the data elements *dataX* (X) and *dataY* (Y). Six distinct DNA data operators were developed with their unique toehold domains which can be fully complementary (A and A*, B and B*, *etc.*). In addition to the toehold domains, each type of DNA data operator has an unique hairpin motif for the data storage. These hairpins do not participate in hybridisation or branch migration; they simply represent 0s and 1s (although, potentially, they could each store or encode more complex data).

The process to record an element starts from an empty stack, which is represented by hybridised *linker* and *start* strands (LS) with the linker attaching to streptavidin beads. The first *push* DNA data operator (P) is added to irreversibly hybridise with the empty stack via the exposed toehold domain A. The stack is now in its data state (LSP), and the corresponding reaction for this operation can be written as $\mathrm{LS} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{LSP}$. The LSP species now has a single open toehold region BC that can accept *dataX* (or *dataY*) DNA data operator via the reaction: $\mathrm{LSP} + \mathrm{X} \xrightarrow{\mathrm{k_{BC}}} \mathrm{LSPX}$. The process can be repeated by adding any number of *push* and then *data* DNA data operator. The reading process proceeds in a reversed way. Starting from a constructed stack, e.g., LSPXPY, the *read* DNA data operator peels the last recorded *dataY* DNA data operator off the stack by hybridisation at the exposed domain A and then three-way branch migration with domains BC. This operation forms the LSPXP stack and a *read-dataX* (RX) double stranded helix which does not expose any single stranded domain and will not participate in further DNA interactions – the corresponding reaction is $\mathrm{LSPXPY} + \mathrm{R} \xrightarrow{\mathrm{k}_1} \mathrm{LSPXP} + \mathrm{RX}$. Similarly, the *pop* DNA data operator (Q) peels off the exposed *push* DNA data operator, thus making the stack ready for another read (or write). The reaction also produces a double stranded helix *push-pop* (PQ): $\mathrm{LSPXP} + \mathrm{Q} \xrightarrow{\mathrm{k}_2} \mathrm{LSPX} + \mathrm{PQ}$.

2.2 Experimental methods

The experimental protocols for the DNA stack are detailed in [20]. Briefly, 400 μL of Sepharose beads (Cytiva) are incubated and washed to remove the storage ethanol. Then $300nM$ of *linker* is added to the Sepharose beads and incubated at 25°C for 30 minutes. After the incubation is completed the species is washed (using a solvent of filtered 1X Tris EDTA(TE)) and the sample is centrifuged for 5s at max RPM. The supernatant is then carefully removed. The process is repeated for each sequential DNA data operator in the order *push*, *data* DNA data operator and so on, until the *releaser* is added to remove the constructed stack from the Sepharose beads. The stack is then analyzed via polyacrylamide gel electrophoresis.

The wash event performed prior to each DNA data operator addition aims to eliminate supernatants such as RX, PQ, P, X, *etc.* It is worth to notice that due to nonspecific binding to beads, the supernatant cannot be perfectly removed by the washing. Therefore, some of these residues can trigger side reactions to form undesired species in the next stack operation, which may harm the target stack production. For example, in the recording process of adding the second *push* DNA data operator , the stack LSPXP can also hybridise with residual *dataX* DNA data operator added in the previous step. Together with the new *push* DNA data operator P, different types of undesired polymers, such as LSPXPX, LSPXPXP, PXPX and others, can be formed in the chemistry.

2.3 Probabilistic model checking

Probabilistic model checking [4, 17] is a powerful technique for analysing systems that exhibit stochastic behaviour. Unlike classical model checking [10] which can only provide qualitative answers, i.e., the system either satisfies or violates a given property, probabilistic model checking can reason *quantitatively* about the probability that the property is true (or false). To probabilistically model check a system, two inputs are required: a probabilistic system model and the specification of a temporal property about the system behaviour. In this section, we introduce the DNA stack system model, and the language utilised for specifying the temporal properties.

2.3.1 Level-based CTMC

Continuous-time Markov chains (CTMCs) are a well-known stochastic model utilised for describing and analysing reaction networks [13]. Traditionally, the CTMC state is a population vector giving the number of molecules of each species. This representation, however, is infeasible for many real-life biological systems, which usually have large molecular numbers. In such cases, the size of the underlying CTMCs become huge: the so-called *state-space explosion* problem. Level-based CTMCs were introduced to model systems in a more abstract and efficient way [2, 6]. In a level-based CTMC, each state is characterised by molecular *concentrations*, discretised into a number of levels. In this paper we use a similar idea, but instead of concentrations levels we utilise *molecular count* levels for a more explicit state-space representation.

▶ **Definition 1** (Level-based CTMC). *A* level-based continuous time Markov chain *is a tuple* (N, S, S_0, R) *where* N *is the molecular count level number,* S *is a finite set of states;* $S_0 \subseteq S$ *is the set of initial states; and* $R : S \times S \to \mathbb{R}_{\geq 0}$ *is the transition rate matrix. Moreover, each state* $s \in S$ *is a tuple* $s = (n_1, n_2, ...n_k)$ *for a fixed* $k \in \mathbb{N}$ *(number of species), where* $n_i : 0 \leq n_i \leq N$ *is the molecular count level for the* i*-th species.*

Thus, given a maximum molecular count M for all species, each level n_i represents the molecular count intervals $[0, \frac{M}{N}), [\frac{M}{N}, 2 \cdot \frac{M}{N}), \ldots, [(N-1) \cdot \frac{M}{N}, N \cdot \frac{M}{N}]$ (level-based CTMCs are useful when $M \gg N$, hence by taking N to be a power of 10 prevents any rounding issues). Level-based CTMCs can be directly derived from stochastic reaction models – for deterministic models one can convert concentrations and rate constants as usual, or directly use concentration levels [2, 6]. Consider a simple reaction $\mathrm{A} + \mathrm{B} \xrightarrow{\mathrm{k}} \mathrm{C}$ with the stochastic rate constant k, and a possible state $s = (n_A, n_B, n_C)$ of species' molecular count levels. The transition rate is calculated as $\frac{n_A \cdot R \cdot n_B \cdot R \cdot k}{R}$ where $R = \frac{M}{N}$.

2.3.2 Continuous Stochastic Logic

Continuous Stochastic Logic (CSL) [3, 5] is a formal notation to express probabilistic temporal properties of CTMCs and other dynamical systems. Formulae in CSL can be classified into state and path formulae:

$$\begin{aligned} \textit{State formula } \Phi &::= \textit{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid P_{\bowtie p}[\varphi] \mid S_{\bowtie p}[\varphi] \\ \textit{Path formula } \varphi &::= X^I \Phi \mid \Phi \, U^I \Phi \end{aligned}$$

where $a \in AP$ is an atomic proposition over the states of a CTMC, $p \in [0, 1]$ is a probability threshold, $\bowtie \in \{\leq, <, \geq, >\}$, and I is an interval of $\mathbb{R}_{\geq 0}$. State formulae are evaluated over states of a CTMC, while path formulae are evaluated over paths (or executions/traces) of a CTMC. Formulae $P_{\bowtie p}[\varphi]$ and $S_{\bowtie p}[\varphi]$ are transient-state and steady-state CSL properties, respectively. In this work, we only consider transient-state properties since our DNA stack has a finite reaction time. A CTMC state s satisfies $P_{\bowtie p}[\varphi]$ if the probability of all the paths starting from s and satisfying φ, satisfies the bound $\bowtie p$. There are two types of operators in a path formula: the *next* operator where $X^I \Phi$ is true if Φ is satisfied in the next state of the path and at a time point $t \in I$, and the *until* operator where $\Phi_1 U^I \Phi_2$ is true if Φ_2 is satisfied at some time point within interval I, and that Φ_1 is true up until that point. Additional path operators can be derived as:

- the *eventually* operator F (future) where $F^I \Phi := \textit{true}\ U^I \Phi$, and
- the *always* operator G (globally) where $G^I \Phi := \neg F^I \neg \Phi$

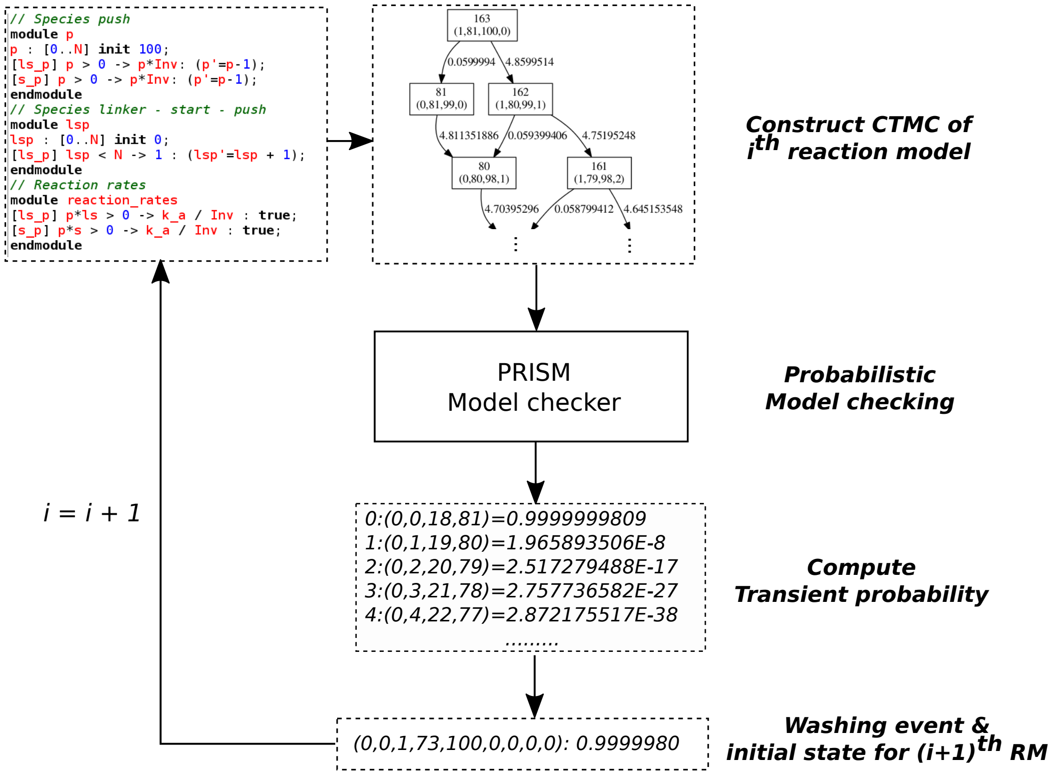
Intuitively, $F^I \Phi$ expresses that Φ is eventually satisfied at some time point within I, whereas a path satisfies $G^I \Phi$ if Φ is true at every time point in I. (See [5] for the formal CSL semantics.)

3 Modelling DNA stack system

3.1 Workflow overview

We have extended the PRISM model checker by implementing an iterative workflow[1] to model the process of the DNA stack system (Figure 2). PRISM offers a high-level modelling language for describing system behaviour. Following the styles defined in [11, 6], we developed

[1] Source code available at `https://github.com/shelllbw/mcSTACK`.



```
// Species push
module p
p : [0..N] init 100;
[ls_p] p > 0 -> p*Inv: (p'=p-1);
[s_p] p > 0 -> p*Inv: (p'=p-1);
endmodule
// Species linker - start - push
module lsp
lsp : [0..N] init 0;
[ls_p] lsp < N -> 1 : (lsp'=lsp + 1);
endmodule
// Reaction rates
module reaction_rates
[ls_p] p*ls > 0 -> k_a / Inv : true;
[s_p] p*s > 0 -> k_a / Inv : true;
endmodule
```

```
0:(0,0,18,81)=0.9999999809
1:(0,1,19,80)=1.965893506E-8
2:(0,2,20,79)=2.517279488E-17
3:(0,3,21,78)=2.757736582E-27
4:(0,4,22,77)=2.872175517E-38
.........
```

```
(0,0,1,73,100,0,0,0,0): 0.9999980
```

Figure 2 CTMC-based workflow for modelling the DNA stack system.

a set of level-based PRISM reaction models (RMs) covering the possible reactions in each DNA operation step. Starting from the first RM which describes the reactions after first adding *push* DNA data operator , the workflow is as follow:

- **Construct CTMC based on** $mathbf i^{th}$ **RM**: if $i = 0$, the initial state for generating the first CTMC is defined as an assumed state of the DNA chemistry such that some initial steps have been performed (see Appendix for details). Otherwise, the initial states are obtained from the result of the previous step. Here we use the *explicit* PRISM engine for building CTMCs as our model has a potentially large state space, but only a small fraction of which is actually reachable. The engine takes less than 10s to build a CTMC with 10^6 states on a 3.4 GHz Intel Core i5 processor with 12GB memory – much faster than the *MTBDD* engine which takes hours.
- **Compute transient probability**: the transient probability at the end of the incubation time is calculated for each CTMC state. Each state represents a possible molecular level that the system can reach at the end of incubation.
- **Perform wash event**: the transient states and their probabilities are modified in order to model the wash event. Washing is assumed to be an instantaneous event. All the state variables corresponding to the bead-bound species (e.g., *LSP, LSPX*) have their value N changed to $(1 - \mu)N$, where $\mu \in [0, 1]$ is the constant fraction of beads lost on each wash. All the state variables corresponding to the non bead-bound supernatant species (e.g., *X, RX, PQ*) have their value N' changed to $\phi N' B_m$, which models the survival from wash

event due to non-specifically bind to beads. Here $\phi \in [0,1]$ is the constant fraction of supernatant solution transferred through the wash cycle to the next reaction stage, and $B_m = (1-\mu)^n$ is the normalised mass of beads remaining where n is the index of the current wash. Moreover, states with the same value after the modification are merged and their probabilities accumulated.

- **Generate initial state of $(\mathbf{i+1})^{\mathbf{th}}$ RM**: in addition to the wash event, transient states are further manipulated for constructing the next RM. This includes extending the states with new variables (i.e., new species formed in the next reaction stage) with the corresponding state values (i.e., initial level of newly added DNA data operators). To reduce the state space size, a cut-off probability of 10^{-5} is introduced so that only states with relatively high probability are considered as initial states in the next step.

3.2 Reaction models

The major challenge of model checking DNA stack systems is the *state space explosion* problem. A stack system with several operations can potentially generate huge numbers of reachable states after only a few iterations. It is therefore important not only to consider the reaction models that we developed are able to describe system behaviour, but also to make sure a reasonable state space size can be maintained for the model analysis. To alleviate the problem, we have applied several methods.

As discussed in Section 3.1, states with low probability (smaller than 10^{-5}) will not be considered as initial states for constructing the next CTMC. Since all reactions in the system are considered as irreversible and the incubation time is sufficiently long, a small portion of the state space often takes a large probability. In most cases, with a 10^{-5} cut-off probability we are able to preserve at least 98% of the total probability with no more than 20 states after 6-8 iterations. Level-based RM and CTMC are applied to further reduce the state space size. To achieve a good precision, a maximum level of 100 ($N = 100$) is used in this work. Thus, given a maximum concentration M of $300nM$ (or equivalently ≈ 27,100 molecules in a volume of 1.5×10^{-13} L), each level represents $3nM$ (271 molecules) of a DNA species.

A coarse-grained RM framework is developed aiming to capture essential reactions in each operation step. Since in this work we are interested in the formation of the target stack, each model consists of two types of reactions: a main reaction which can form the target DNA stack species, and a set of side reactions that can directly or indirectly react with the species in the main reaction, thereby reducing the production of the target stack. Let us consider the following operation sequence as an example:

add *push* DNA data operator P ⨟ add *dataX* DNA data operators X ⨟ add *push* DNA data operator P

where we assume the initial species in the chemistry are bead-bound *linker-starter* LS and *starter* S. We can write a unique RM for each incubation stage after adding each new DNA data operator:
RM1 (add *push* DNA data operators P):

$$\boxed{\mathrm{LS} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{LSP} \quad (\text{target}), \qquad \mathrm{S} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{SP}}$$

RM2 (add *dataX* DNA data operators X):

$$
\begin{array}{ll}
(1) & \mathrm{LSP} + \mathrm{X} \xrightarrow{\mathrm{k_{BC}}} \mathrm{LSPX} \ \ \text{(target)}, \\
(2) & \mathrm{P} + \mathrm{X} \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{X} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma, \\
(3) & \mathrm{X} \ + \ \mathrm{P}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma, \quad \mathrm{P} + \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{P}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma, \\
(4) & \mathrm{LSPX} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{LS}\sigma, \quad \mathrm{LSP} + \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{LS}\sigma, \quad \mathrm{LSPX} + \mathrm{P}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{LS}\sigma
\end{array}
$$

and RM3 (add *push* DNA data operators P):

$$
\begin{array}{ll}
(1) & \mathrm{LSPX} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{LSPXP} \ \ \text{(target)}, \\
(2) & \mathrm{P} + \mathrm{X} \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{X} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma, \\
(3) & \mathrm{X} \ + \ \mathrm{P}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma, \quad \mathrm{P} + \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{P}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma, \\
(4) & \mathrm{LSPXP} + \mathrm{X} \xrightarrow{\mathrm{k_{BC}}} \mathrm{LS}\sigma, \quad \mathrm{LSPX} + \mathrm{P}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{LS}\sigma, \quad \mathrm{LSPXP} + \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{LS}\sigma
\end{array}
$$

where σ denotes all possible polymers constructed by P and X (PXP, PXPX, PXPXP,...). The first reaction in each RM is the main reaction which produces target stack species (i.e., LSP in RM1, LSPX in RM2, and LSPXP in RM3), while the rest are side polymerisation reactions. In each main reaction, the initial concentrations of new-added reactant (X or P) is always higher than those pre-existing reactant (LS, LSP or LSPX) due to the wash in previous step. Therefore, a considerable amount of the newly added reactants that cannot take part in the reactions and survive from wash events would remain in the chemistry. During the incubation stage of the next operation step, these residues or the σ polymer they formed negatively affect the production rate of target stacks (i.e., side reactions (2)-(4) in RM2 and RM3). In particular, reactions (2) and (3) are hybridisation of non bead-bound supernatants to σ polymers, and reactions (4) are hybridisation of bead-bound stack and supernatants. In the model, we do not explicitly represent the detailed structure of σ polymers as it generates RM with infinite reactions. Instead, we only show their left-most (start) region which determines the capability to hybridise with the stack species in the main reactions. In this way, the RM complexity is greatly reduced. There are two types of σ polymers that can be formed under such representation: the one starting with *push* P DNA data operator (Pσ) and the one starting with *data* W DNA data operator (Wσ). Both of them are able to hybridise with the stack species in the main reaction inhibiting the production of target stacks.

On the other hand, some reactions are not considered in our RM framework either because the molecular level of their reactants stays at zero, or they do not harm the production of target stacks. For example, two σ polymers with complementary start and end region may hybridise and form a longer polymer: $\mathrm{P}\sigma\mathrm{W} + \mathrm{P}\sigma\mathrm{P} \longrightarrow \mathrm{P}\sigma\mathrm{P}$. Such reaction in fact reduces the polymer concentration and thereby benefits the production of target stacks. By ignoring those reactions, we assume that our RM is always the “worst” case for the target stacks’ formation, and thus the lower bound of their concentration is guaranteed.

We can use the same idea to build RMs for the *read* and *pop* operation steps. Popping the last added *push* DNA data operator and then reading the *dataX* DNA data operator yield the following RMs:

RM4 (add *pop* DNA data operators Q):

$$\boxed{\begin{array}{ll} (1) & \mathrm{LSPXP} + \mathrm{Q} \xrightarrow{\mathrm{k_2}} \mathrm{LSPX} + \mathrm{PQ} \ \ \text{(target)}, \\ (2) & \mathrm{LSPX} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{LSPXP}, \quad \mathrm{P} + \mathrm{Q} \xrightarrow{\mathrm{k_{ABC}}} \mathrm{PQ} \end{array}}$$

and RM5 (add *read* DNA data operators R):

$$\boxed{\begin{array}{ll} (1) & \mathrm{LSPX} + \mathrm{R} \xrightarrow{\mathrm{k_1}} \mathrm{LSP} + \mathrm{RX} \ \ \text{(target)}, \quad \mathrm{LSP} + \mathrm{Q} \xrightarrow{\mathrm{k_2}} \mathrm{LS} + \mathrm{PQ}, \\ (2) & \mathrm{Q} + \mathrm{R} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}, \quad \mathrm{R} + \mathrm{Q} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q} \\ (3) & \mathrm{R} + \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}, \quad \mathrm{Q} + \tau\mathrm{R} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}, \quad \tau\mathrm{R} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}, \quad \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R} \\ (4) & \mathrm{LSPX} + \tau\mathrm{R} \xrightarrow{\mathrm{k_{chain}}} \mathrm{LS}\sigma + \mathrm{RX} + \mathrm{PQ} \end{array}}$$

Similar to the RMs for *push* and *write* steps, the production of target stack species (LSPX in RM4, and LSP in RM5) are affected by both main reactants and side reactions. The residual species surviving wash events (i.e., P in RM4, and Q in RM5) can either directly hybridise with target stack or indirectly react with them via τ polymers which trigger chain annihilation reactions [20], namely (4) in RM5. It is worth noting that the occurrence of chain annihilation reactions is determined by the right-most (end) region of τ polymer. Thus the right-most region of τ is considered explicitly during the polymer formation, which produces either τR or τQ (i.e., reactions (2) in RM5).

3.3 Model example

To illustrate how the CTMC-based DNA stack model works, consider the below operation sequence:

add *push* DNA data operator P $\fatsemi$ add *dataX* DNA data operator X $\fatsemi$ add *push* DNA data operator P .

The corresponding RMs are the RM1-RM3 illustrated above. To start, the initial state of RM1 is set as:

(LS=81, P=100, S=2, LSP=0, SP=0)

which corresponds to LS $= 243nM$, P $= 300nM$ and S $= 6nM$ (lower LS concentration is caused by the wash events in set-up process). The supernatant survival rate ϕ is set as 0.2, and the loss rate of bead-bound species μ is 0.1. The incubation time of each step is 30 minutes. Moreover, we only consider transient states with probability greater than 10^{-5} as initial states for the next RM. With the above conditions, a level-based CTMC is built from RM1 with 246 states and 408 transitions, and after incubation the transient probability ($> 10^{-5}$) for the following state is

(LS=0, P=17, S=0, LSP=81, SP=2) = 0.9999999436478932.

The state shows that after incubation all copies of LS contributed to forming LSP, but there are still 17 levels of P remained in the chemistry. Then the wash event is performed by removing 80% ($\phi = 0.2$) of the supernatant species P and SP, and 10% ($\mu = 0.1$) of the (bead-bound) target stack LSP.

To construct the next CTMC, the states are also extended with new variables representing the new species introduced by RM2. Moreover, variable X is set to 100, meaning that $300nM$ of new *dataX* DNA data operators is added to the chemistry. State variables corresponding to the species LS, S and SP will not be considered as their concentration levels will remain zero, giving the following initial state and probability:

$$(\mathrm{P{=}3,\ LSP{=}73,\ X{=}100,\ LSPX{=}0,\ LS}\sigma = 0, \mathrm{P}\sigma = 0, \mathrm{W}\sigma = 0) = 0.9999999436478932.$$

The CTMC built from RM2 and the above initial state consists of 5,782 states and 20,108 transitions. The transient probability shows that after incubation there are 25 states with probability greater than 10^{-5}, with the total probability greater than 99.99%. By performing a wash event and state extension, 7 initial states are generated for building the third CTMC that adds *push* DNA data operators (RM3):

$$\begin{aligned}
&(\mathrm{P{=}100,\ LSPX{=}66,\ X{=}3,\ LSPXP{=}0,\ LS}\sigma = 0, \mathrm{P}\sigma = 0, \mathrm{W}\sigma = 0) = 0.01657086916873019\\
&(\mathrm{P{=}100,\ LSPX{=}65,\ X{=}3,\ LSPPX{=}0,\ LS}\sigma = 0, \mathrm{P}\sigma = 0, \mathrm{W}\sigma = 0) = 0.12083719755305258\\
&(\mathrm{P{=}100,\ LSPX{=}64,\ X{=}3,\ LSPXP{=}0,\ LS}\sigma = 0, \mathrm{P}\sigma = 0, \mathrm{W}\sigma = 0) = 0.17053834871599782\\
&(\mathrm{P{=}100,\ LSPX{=}63,\ X{=}3,\ LSPXP{=}0,\ LS}\sigma = 0, \mathrm{P}\sigma = 0, \mathrm{W}\sigma = 0) = 0.03350530289916429\\
&(\mathrm{P{=}100,\ LSPX{=}63,\ X{=}4,\ LSPXP{=}0,\ LS}\sigma = 0, \mathrm{P}\sigma = 0, \mathrm{W}\sigma = 0) = 0.36343236468968576\\
&(\mathrm{P{=}100,\ LSPX{=}64,\ X{=}4,\ LSPXP{=}0,\ LS}\sigma = 0, \mathrm{P}\sigma = 0, \mathrm{W}\sigma = 0) = 0.26484767882197474\\
&(\mathrm{P{=}100,\ LSPX{=}65,\ X{=}4,\ LSPXP{=}0,\ LS}\sigma = 0, \mathrm{P}\sigma = 0, \mathrm{W}\sigma = 0) = 0.03026448265945934
\end{aligned}$$

The resulting CTMC has 46,500 states and 180,110 transitions, and completes this simple example.

4 Results

In this section, we apply our CTMC-based framework to model a 2-signal DNA stack system that manipulates two data elements, X and Y. The model will be first validated against stochastic simulation of a rule-based model implementing the same system. Then we apply probabilistic model checking technique to analyse and optimise the system.

4.1 Modelling and validating a 2-signal DNA stack system

The 2-signal DNA stack system pushes two data elements *dataX* (X) and *dataY* (Y) to the stack, and then performs two pop operations to read out the recorded data:

add *push* DNA data operators P ⨾ add *dataX* DNA data operators X ⨾ add *push* DNA data operators P ⨾ add *dataY* DNA data operators Y ⨾
add *read* DNA data operators R ⨾ add *pop* DNA data operators Q ⨾ add *read* DNA data operators R ⨾ add *pop* DNA data operators Q

The reaction models of the above eight operation steps are detailed in the Appendix. A maximum molecular level $N = 100$ is applied, and a fixed $300nM$ (100 levels) of each DNA data operators is added to the chemistry at the beginning of each operation step. For each step, we evaluate the mean level of the target stack after 30 minutes of incubation time.

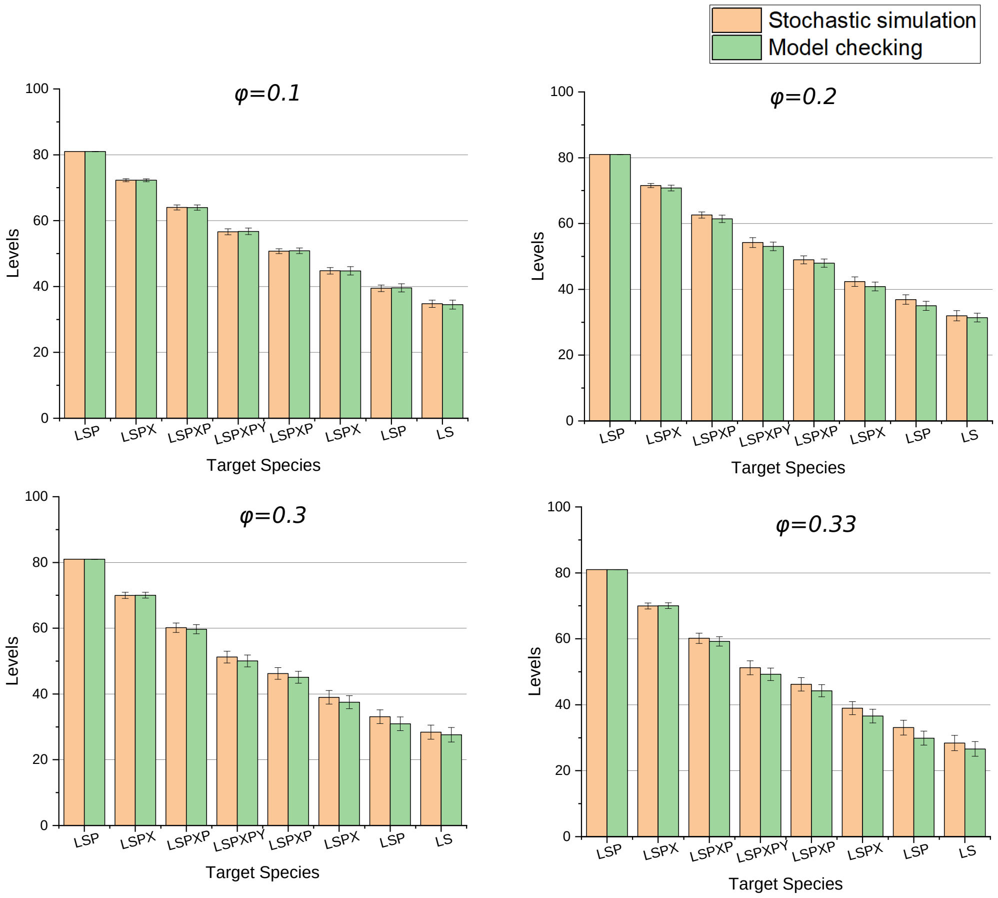


Figure 3 Comparison of the PRISM CTMC-based model and stochastic simulation of the rule-based DNA stack model: mean concentration of the target stacks after 30 minutes of incubation time for each reaction step for different values of ϕ; $\mu = 0.1$ in all cases.

Table 1 Performance of CTMC-based model and rule-based stochastic simulation: state space and transition size for all eight operation steps, total state space probability in the last step, and CPU times for model checking (MC) and stochastic simulation (SS).

	States	Transitions	Total probability	CPU time (MC)	CPU time (SS)
$\phi = 0.1$	3.3×10^4	1.1×10^5	99.87%	1.5 mins	1.8 mins
$\phi = 0.2$	2.8×10^5	1.2×10^6	99.59%	39 mins	2.2 mins
$\phi = 0.3$	2.8×10^6	1.5×10^7	98.77%	5.1 hours	2.7 mins
$\phi = 0.33$	4.0×10^6	2.2×10^7	98.43%	8.0 hours	2.9 mins

The green bars in Figure 3 show the results of the CTMC-based model with fixed $\mu = 0.1$ and various ϕ values. It can be observed that the mean level of the target stacks decrease gradually w.r.t both the operation step and the increase in ϕ. This is not only because 10% of the stack species are removed during each wash event, but also because residues (e.g., strands P, X, R) that survived the wash accelerate side reactions. As a result, the level of the target stack (LS) drops to 26 in the last step when $\phi = 0.33$ is applied. In contrast, since less residue survives when $\phi = 0.1$, there are 34 levels of the target stack (LS) produced in the last step. Some performance statistics are reported in Table 1. The size of the state space grows significantly with ϕ, and the computation takes 8 hours when $\phi = 0.33$. Despite the large state space, in all cases more than 98% of the total probability reaches the last step when a 10^{-5} cut-off probability is used in each step.

Due to the simplifications applied to the RMs, behaviours such as polymer-polymer hybridisation are not considered in the CTMC-based model. It is therefore interesting to compare the results of our model with the results of the more detailed model. We compare our CTMC-based model against stochastic simulation of a rule-based model implementing the same system [20]. The rule-based model covers 20 DNA strand polymerisation and displacement rules. The comparison is shown in Figure 3. The stochastic simulations are run with 100 replicates for each case. Overall there is a good agreement between the two models. In the CTMC-based model, since we simplify the RMs by guaranteeing the lower bound of target stack, the mean level is always lower than that of the stochastic simulation. With $\phi = 0.1$, however, such difference is negligible (less than 1% in the last step). This is due to the small amount of residues survived from wash, which reduces the occurrence of the side reactions. With $\phi = 0.33$ more residues are in the chemistry, thus the difference between the two models increases to 4%. The performance of the stochastic simulations is given in Table 1. Although a single simulation takes 1-3 mins to finish, a much longer time (normally a few hours) is needed to collect enough replicates for computing statistically reliable results.

4.2 Optimising incubation time

Due to reaction irreversibility, the level of the target stack can eventually reach a stable value if there is no other species with which it can react. Knowing the time to reach such a stable state is important as it helps to determine the minimum incubation time. This is particularly useful for large DNA stack systems, which may involve tens of operation steps. In this section, we will analyse our probability CTMC-based model to optimise the incubation time of each DNA operation step. The 2-signal system developed in the previous section is considered. The operations of push, write, pop, and read are achieved by adding a fix amount ($300nM$) of each of the respective DNA data operators. Based on *in vitro* experiment measurements [20], ϕ and μ are set as 0.33 and 0.1, respectively.

To optimise the incubation time, we use the probabilistic model checker PRISM to quantitatively verify the following CSL property:

$$\mathit{Prop}1 : P_{=?}[\ G_{\geq T}\ \text{`target\_spec\_stable'}\ ]\ .$$

Intuitively, the property asks: "What is the probability that the target stack reaches a stable level from time T onwards?" The condition "target_spec_stable" depends on the reactions of a given operation step. For example, for the RM2 of Section 3.2, the property becomes:

$$\mathit{Prop}1(\mathrm{RM2}) : P_{=?}[\ G_{\geq T}\ (\mathrm{LSP} = 0 \wedge \mathrm{P} = 0 \wedge \mathrm{P}\sigma = 0)\ ]$$

That is, the target stack LSPX reaches its stable level when all the main reactant LSP is transformed, and there is neither *push* DNA data operators P nor Pσ polymers in the chemistry (LSP, X, P, and Pσ are the species that can affect the production of LSPX). The main reactant X is not considered in the property as it has higher level than LSP. Thus, the property can always be satisfied at some time-unit as LSP, P, and Pσ can eventually be consumed by X-ended species. Using the same idea, we can re-write *Prop1* for any reaction network. For example, the yield of target species LS in RM5 in Section 3.2 is determined by LSPX, Q and τQ. The corresponding property is:

$$\mathit{Prop}1(\mathrm{RM5}) : P_{=?}[\ G_{\geq T}\ (\mathrm{LSPX} = 0 \wedge \mathrm{Q} = 0 \wedge \tau\mathrm{Q} = 0)\ ]\ .$$

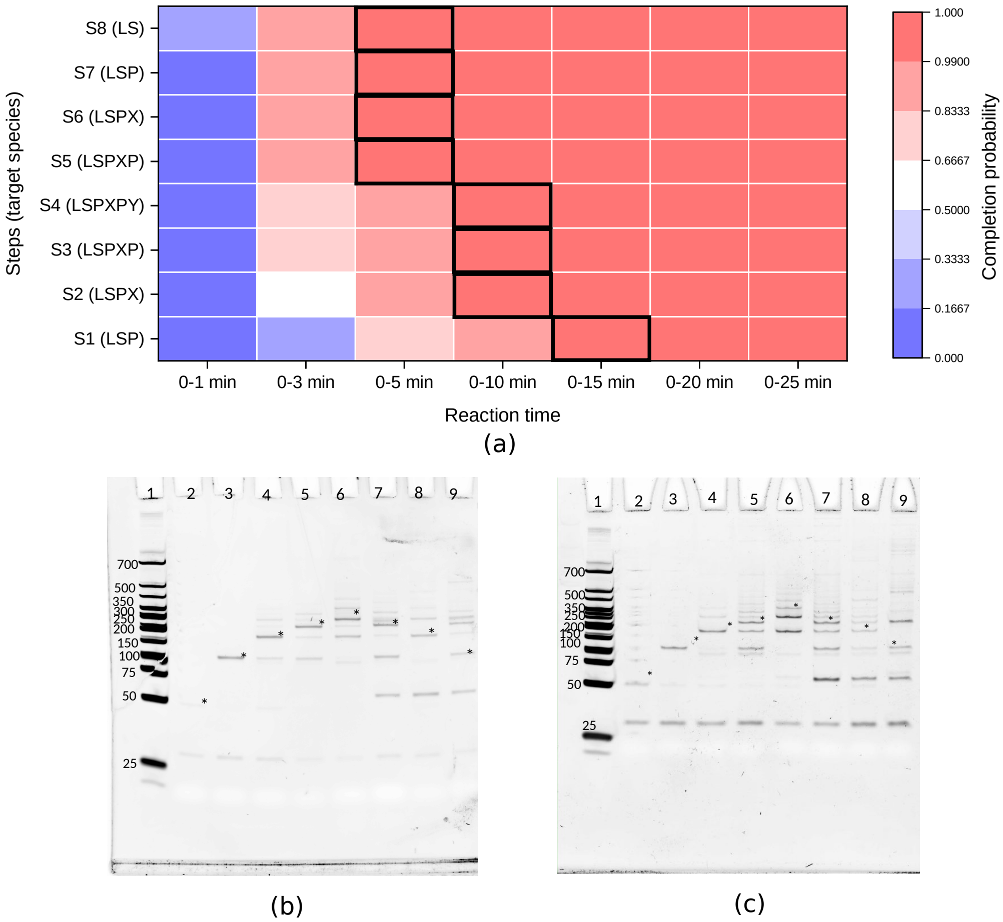


Figure 4 **(a)** Heatmap for the satisfaction probability of *Prop1* for target species of the 2-signal DNA stack over a range of time units T, from 60 (1 min) to 1,500 (25 min). Cells with black border indicate the suggested incubation time for each operation step. **(b)** Gel electrophoresis for a 2-signal DNA stack operation with a 30 minutes incubation. For each lane the target species has an asterisk next to it. Every other unlabeled species is a byproduct. Specifically: Lane 1: NEB low Molecular ladder (see Appendix for details); Lane 2: S; Lane 3: SP; Lane 4: SPX; lane 5: SPXP; Lane 6: SPXPX; Lane 7: SPXPX+R; Lane 8: SPXPX+RQ; and Lane 9: SPXPX+RQR. **(c)** Gel electrophoresis for a 2-signal DNA stack with the variable incubation timings taken from the schema optimised by probabilistic model checking.

Figure 4 (a) shows the probability distribution of *Prop1* with incubation time T ranging from 60 units (1 min) to 1500 units (25 min). The highlighted cells correspond to properties with the lowest reaction time T as well as having at least 99% satisfaction probability. From the result, it is suggested that an incubation time of 10-15 min is required for the first four operation steps. This is because at these stages the concentration levels of the reactants in the main reactions are relatively high and also close to each other. However, the incubation time can be reduced to 5 min in the following steps, as there is a significant difference in the levels of the reactants, which causes a rapid consumption of reactants with lower concentration.

Experimental validation of the model checking result is given in Figure 4 (b) and (c). A full stack operation for the 2-signal system is performed with 30 minutes incubation time (Figure 4 (a)), and the variable incubation time suggested from the model checking

((Figure 4 (b)). As can be seen, in addition to the target species at each operation stage (asterisks), multiple unwanted species also exist. This is in particular the case during the reading process (lanes 7-10) with the increase of side reaction rates. However, the two images show a qualitative agreement, indicating that a similar performance can be achieved with optimised incubation time.

4.3 Improving target stack yield

The results in Section 4.1 show that high ϕ values (fraction of supernatants surviving the washing) can significantly reduce the yield of the target stack. This is due to the large amount of undesired DNA species (e.g., *push* DNA data operators P in RM2) that survives from the wash event and increase the rate of side reactions, which compete with the main reactions. Enhancing wash efficiency and decreasing ϕ is difficult to achieve *in vitro*. However, we may still be able to minimise the concentration of undesired species by controlling the amount of each new DNA data operator added to the chemistry at each step. Consider the 2-signal system with $\phi = 0.33$, where a fixed $300nM$ of each DNA data operators is added at each step. The main reaction in the second step "*adding dataX*" is $\mathrm{LSP} + \mathrm{X} \xrightarrow{\mathrm{K_{BC}}} \mathrm{LSPX}$. Initially, there is around $210nM$ of LSP in the system. Adding $300nM$ of *dataX* DNA data operators results in a large portion of these new DNA data operators not being able to react with LSP. Instead, they become undesired species in the next step, which in turn negatively affects the yield of the target stack LSPXP. Therefore, if we reduce the amount of these new *dataX* DNA data operators, a higher yield of the target stack would be expected.

In this section, we employ the CSL logic and PRISM to maximise the yield of the target stack by tuning the amount of each DNA data operator added at each step. The idea is to first compute a maximum target stack yield when adding a sufficient amount of the DNA data operators (in our case, $300nM$). Taking the result as a reference value, we gradually decrease the amount of DNA data operators added until reaching a minimum value for which the system can still produce the desired level of target stack. We use the following CSL property:

$$\mathit{Prop}1(C) : P_{\geq 0.99}[\ F[1800, 1800]\ \text{target\_spec} \geq C\ ]\ .$$

Informally, *Prop1(C)* means that "Given a molecular level C, the probability of target stack reaching such level after 30 minutes is greater than 99%." We apply the property for $1 \leq C \leq 100$, and take the maximal C which satisfies the property as the reference level C_{ref}:

$$C_{\mathrm{ref}} = \mathrm{Max}\{C \ \mid \ \mathit{Prop}1(C) \wedge 1 \leq C \leq 100\}\ .$$

Figure 5(a) gives an example of determining the reference level for the step *"adding dataX"* (RM3). When adding 100 levels ($300nM$) of dataX, the property *Prop1(C)* : $P_{\geq 0.99}[$ $F[1800, 1800]$ LSPX $\geq$ C] is *false* for $C = 71$ ($213nM$) and beyond, and is *true* for $C = 70$ ($210nM$) and below, indicating that the system can produce a maximum level of 70 LSPX with 99% probability. Thus we take 70 as the reference level C_{ref}. Now, the following CSL property is employed to find the optimised addition concentration:

$$\mathit{Prop}2 : P_{\geq 0.99}[\ F[1800, 1800]\ \text{target\_spec} \geq C_{\mathrm{ref}}\ ]\ .$$

The property evaluates whether the system can yield with high probability ($\geq 99\%$) at least C_{ref} levels of the target stack at 30 minutes. In order to find the optimal level, we model check the system with different initial states (i.e., smaller levels) until *Prop2* is false. In this

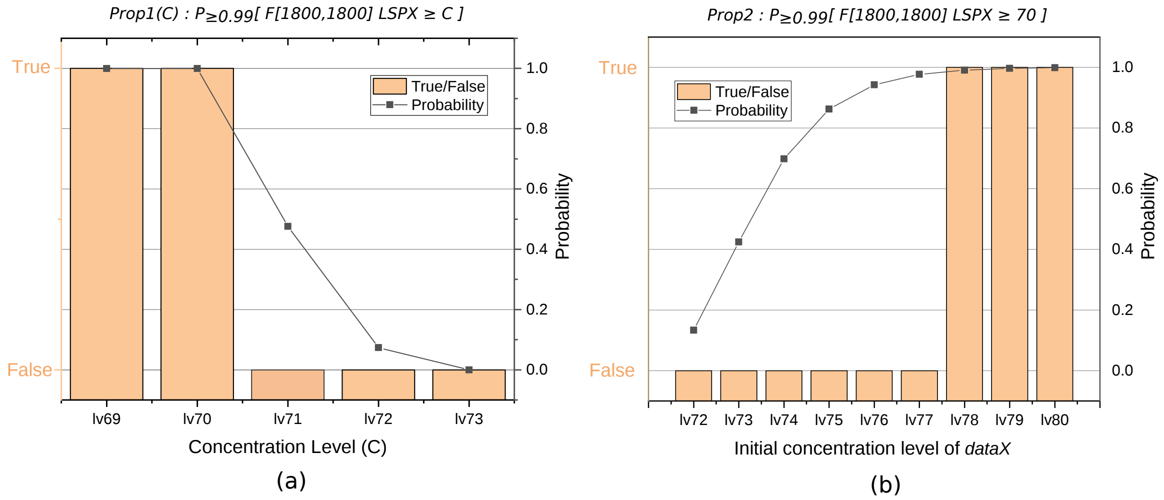


Figure 5 Optimising DNA addition levels (step *"adding dataX"* (RM3)) using probabilistic model checking. Panels: **(a)** Evaluating *Prop1(C)* for C = {60,70,71,72,73} when adding 300nM dataX; **(b)** Evaluating *Prop2* with reduced initial level of dataX, from 72 to 80. Yellow bars: satisfaction of properties; lines: satisfaction probabilities.

way we find the minimal initial level that satisfies the reference level of target stack C_{ref}. In Figure 5(b) we evaluate *Prop2* using the reference level obtained in (a) for a range of initial levels. The result suggests that adding 78 levels (234nM) of dataX is the minimum value for which the system produces the same amount of LSPX when adding 300nM of dataX. The optimised value, however, can avoid up to 64nM of dataX remaining in the chemistry, which can harm the system yield in the next step.

We applied our approach to optimise the 2-signal system with $\phi = 0.33$ and $\mu = 0.1$. We compare the mean level of target stacks in the optimised system with the standard system with fixed concentrations, as shown in Figure 6. In the last step, the yield of target stack in the optimised system is 40% higher than the yield of the standard system.

5 Conclusion and future work

In this work, we applied probabilistic model checking to model and optimise a DNA stack nano-device. An iterative CTMC-based workflow is developed within the PRISM probabilistic model checker to model the system. Each iteration describes a single DNA operation step: add new DNA data operator, incubation, and wash. The initial state of the $(i + 1)^{th}$ step is determined by the model checking result (transient probability) of the i^{th} step. Level-based CTMCs and simplified reaction models are applied to reduce the state space size. We have used our framework to develop a 2-signal DNA stack system model. The model is validated against stochastic simulation of a rule-based model implementing the same system. We use probabilistic model checking to formally analyse and optimise our 2-signal system. Results suggest that when a fix amount (300nM) of each DNA data operator are added in each operation step, an incubation time of 10-15 minutes is required for the first several operation steps, while for the following steps the incubation time can be reduced to 5 minutes. We have experimentally validated the result by comparing the gel electrophoresis obtained from the optimised and unoptimised (30 min incubation time) systems. We also applied probabilistic

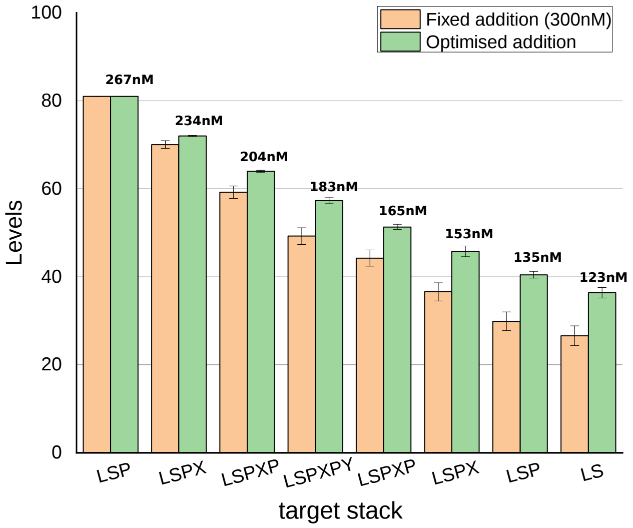


Figure 6 Mean level of target stacks after adding a fixed $300nM$ of each DNA data operator in each operation step (yellow bars) *vs.* adding the optimised level (green bars). The values on top of the green bars are the actual (optimised) levels added to the systems.

model checking to maximise the yield of target stacks by optimising the amount of DNA data operators added at each reaction step: our optimised model shows a 40% increase in target stack yield.

Experimental validation on improving the yield of DNA nanostructures is an ongoing work. The major challenges of the validation are two-fold. First, the concentration difference between the optimised and unoptimised protocols in some cases (i.e., LSP and LSPX in Figure 6) is small, which makes the analysis of the results difficult. For example, it is hard to figure out whether the differences in the experimental output are caused by the model creating the correct schematic or the stochastic nature of pipetting error. Second, the model is influenced by two main parameters: ϕ and μ. Both these factors exhibit a large influence over the model and are difficult to optimise and control in the experimental setting. There are two strategies that we could implement in future to potentially solve these problems. One approach to explore the pipetting stochasticity would be to use a liquid handling robot to carry out the operations instead of a manual pipette, as robots have much higher accuracy than human operators and are not prone to fatigue and other factors. It may also improve the bead loss and washing efficiency. Another strategy to mitigate the bead loss could be to use another DNA nanostructure to tether the device instead of beads. Finally, an enzymatic wash could be explored to reduce the noise and in turn the off target species by digesting potential byproducts and leaving on the target species intact.

References

1 Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994. doi:10.1126/science.7973651.

2 Oana Andrei and Muffy Calder. A Model and Analysis of the AKAP Scaffold. *Electronic Notes in Theoretical Computer Science*, 268:3–15, 2010. Proceedings of the 1st International Workshop on Interactions between Computer Science and Biology (CS2Bio'10). doi:10.1016/j.entcs.2010.12.002.

3 Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert K. Brayton. Model-checking continous-time markov chains. *ACM Trans. Comput. Log.*, 1(1):162–170, 2000. doi:10.1145/343369.343402.

4 Christel Baier, Edmund M. Clarke, Vasiliki Hartonas-Garmhausen, Marta Z. Kwiatkowska, and Mark Ryan. Symbolic model checking for probabilistic processes. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440. Springer, 1997. doi:10.1007/3-540-63165-8_199.

5 Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003. doi:10.1109/TSE.2003.1205180.

6 Muffy Calder, Vladislav Vyshemirsky, David Gilbert, and Richard Orton. Analysis of Signalling Pathways Using Continuous Time Markov Chains. In Corrado Priami and Gordon Plotkin, editors, *Transactions on Computational Systems Biology VI*, pages 44–67. Springer Berlin Heidelberg, 2006. doi:10.1007/11880646_3.

7 Luca Cardelli. Strand algebras for DNA computing. In Russell J. Deaton and Akira Suyama, editors, *DNA Computing and Molecular Programming, 15th International Conference, DNA 15, Fayetteville, AR, USA, June 8-11, 2009, Revised Selected Papers*, volume 5877 of *Lecture Notes in Computer Science*, pages 12–24. Springer, 2009. doi:10.1007/978-3-642-10604-0_2.

8 Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8, September 2013. doi:10.1038/nnano.2013.189.

9 George M. Church, Yuan Gao, and Sriram Kosuri. Next-Generation Digital Information Storage in DNA. *Science*, 337(6102):1628–1628, 2012. doi:10.1126/science.1226355.

10 Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2001.

11 Parker Dave, Norman Gethin, and Kwiatkowska Marta. Prism user manual, 2017. URL: http://www.prismmodelchecker.org/manual/.

12 Harold Fellermann, Annunziata Lopiccolo, Jerzy Kozyra, and Natalio Krasnogor. In vitro implementation of a stack data structure based on DNA strand displacement. In Martyn Amos and Anne Condon, editors, *Unconventional Computation and Natural Computation - 15th International Conference, UCNC 2016*, volume 9726 of *Lecture Notes in Computer Science*, pages 87–98. Springer, 2016. doi:10.1007/978-3-319-41312-9_8.

13 Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976. doi:10.1016/0021-9991(76)90041-3.

14 John Heath, Marta Kwiatkowska, Gethin Norman, David Parker, and Oksana Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257, 2008. doi:10.1016/j.tcs.2007.11.013.

15 Savas Konur, Marian Gheorghe, Ciprian Dragomir, Laurentiu Mierla, Florentin Ipate, and Natalio Krasnogor. Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems. *ACS Synthetic Biology*, 4(1):83–92, 2015. doi:10.1021/sb500134w.

16 Jerzy Kozyra, Harold Fellermann, Ben Shirt-Ediss, Annunziata Lopiccolo, and Natalio Krasnogor. Optimizing nucleic acid sequences for a molecular data recorder. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 1145–1152. ACM, 2017. doi:10.1145/3071178.3071345.

17 Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In Marco Bernardo and Jane Hillston, editors, *Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*, pages 220–270. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-72522-0_6.

18 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011. doi:10.1007/978-3-642-22110-1_47.

19 Matthew R. Lakin, David Parker, Luca Cardelli, Marta Kwiatkowska, and Andrew Phillips. Design and analysis of dna strand displacement devices using probabilistic model checking. *Journal of The Royal Society Interface*, 9(72):1470–1485, 2012. doi:10.1098/rsif.2011.0800.

20 Annunziata Lopiccolo, Ben Shirt-Ediss, Emanuela Torelli, Abimbola Feyisara Adedeji Olulana, Matteo Castronovo, Harold Fellermann, and Natalio Krasnogor. A last-in first-out stack data structure implemented in DNA. *Nature Communications*, 12(1):4861, 2021. doi:10.1038/s41467-021-25023-6.

21 Lulu Qian, David Soloveichik, and Erik Winfree. Efficient Turing-Universal Computation with DNA Polymers. In Yasubumi Sakakibara and Yongli Mi, editors, *DNA Computing and Molecular Programming*, pages 123–140. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-18305-8_12.

22 Lulu Qian and Erik Winfree. A Simple DNA Gate Motif for Synthesizing Large-Scale Circuits. In Ashish Goel, Friedrich C. Simmel, and Petr Sosik, editors, *DNA Computing and Molecular Programming*, pages 70–89. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-03076-5_7.

A Complete reaction models for 2-signal DNA stack system

RM1 (add *push* data operators P):

$$\boxed{\mathrm{LS + P \xrightarrow{k_A} LSP\,(target)}, \quad \mathrm{S + P \xrightarrow{k_A} SP}}$$

RM2 (add *dataX* data operators X):

$$\boxed{\begin{array}{ll}
(1) & \mathrm{LSP + X \xrightarrow{k_{BC}} LSPX \;\; (target)}, \\
(2) & \mathrm{P + X \xrightarrow{k_{BC}} P\sigma}, \quad \mathrm{X + P \xrightarrow{k_A} W\sigma}, \\
(3) & \mathrm{X \; + \; P\sigma \xrightarrow{k_A} W\sigma}, \quad \mathrm{P + W\sigma \xrightarrow{k_{BC}} P\sigma}, \quad \mathrm{W\sigma \xrightarrow{k_{BC}} P\sigma}, \quad \mathrm{P\sigma \xrightarrow{k_A} W\sigma}, \\
(4) & \mathrm{LSPX + P \xrightarrow{k_A} LS\sigma}, \quad \mathrm{LSP + W\sigma \xrightarrow{k_{BC}} LS\sigma}, \quad \mathrm{LSPX + P\sigma \xrightarrow{k_A} LS\sigma}
\end{array}}$$

RM3 (add *push* data operators P):

$$\boxed{\begin{array}{ll}
(1) & \mathrm{LSPX + P \xrightarrow{k_A} LSPXP \;\; (target)}, \\
(2) & \mathrm{P + X \xrightarrow{k_{BC}} P\sigma}, \quad \mathrm{X + P \xrightarrow{k_A} W\sigma}, \\
(3) & \mathrm{X \; + \; P\sigma \xrightarrow{k_A} W\sigma}, \quad \mathrm{P + W\sigma \xrightarrow{k_{BC}} P\sigma}, \quad \mathrm{W\sigma \xrightarrow{k_{BC}} P\sigma}, \quad \mathrm{P\sigma \xrightarrow{k_A} W\sigma}, \\
(4) & \mathrm{LSPXP + X \xrightarrow{k_{BC}} LS\sigma}, \quad \mathrm{LSPX + P\sigma \xrightarrow{k_A} LS\sigma}, \quad \mathrm{LSPXP + W\sigma \xrightarrow{k_{BC}} LS\sigma}
\end{array}}$$

where the bi-molecular rate constants were found to be approximately $K_A \approx K_{BC} \approx 3 \times 10^4 Ms^{-1}$, and $K_{ABC} \approx 2.5 \times 10^5 M^{-1}s^{-1}$. The bi-molecular strand displacement rate constants K_1 and K_2 were assumed equal to hybridisation constants K_A and K_{BC} respectively, due to the long 28nt "toeholds" of the strand displacement reaction [20]. Moreover, the chain annihilation reactions were approximated as a single step bimolecular reaction whose rate constant was set to the rate constant of the initial hybridisation event as an upper bound i.e, $k_A^{chain} = k_A$ and $k_{BC}^{chain} = k_{BC}$.

RM4 (add *dataY* data operators Y):

$$\boxed{\begin{array}{ll}
(1) & \mathrm{LSPXP} + \mathrm{Y} \xrightarrow{\mathrm{k_{BC}}} \mathrm{LSPXPY}\,(\text{target}),\\
(2) & \mathrm{P} + \mathrm{Y} \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{Y} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma,\\
(3) & \mathrm{Y} \ + \ \mathrm{P}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma, \quad \mathrm{P} + \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{P}\sigma, \quad \mathrm{P}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{W}\sigma,\\
(4) & \mathrm{LSPXPY} + \mathrm{P} \xrightarrow{\mathrm{k_A}} \mathrm{LS}\sigma, \quad \mathrm{LSPXP} + \mathrm{W}\sigma \xrightarrow{\mathrm{k_{BC}}} \mathrm{LS}\sigma, \quad \mathrm{LSPXPY} + \mathrm{W}\sigma \xrightarrow{\mathrm{k_A}} \mathrm{LS}\sigma
\end{array}}$$

RM5 (add *read* data operators R):

$$\boxed{\begin{array}{ll}
(1) & \mathrm{LSPXPY} + \mathrm{R} \xrightarrow{\mathrm{k_1}} \mathrm{LSPXP} + \mathrm{RY}\,(\text{target}),\\
(2) & \mathrm{LSPXP} + \mathrm{Y} \xrightarrow{\mathrm{k_{BC}}} \mathrm{LSPXPY} + \mathrm{RY}, \quad \mathrm{R} + \mathrm{Y} \xrightarrow{\mathrm{k_{ABC}}} \mathrm{RY}
\end{array}}$$

RM6 (add *pop* data operators Q):

$$\boxed{\begin{array}{ll}
(1) & \mathrm{LSPXP} + \mathrm{Q} \xrightarrow{\mathrm{k_2}} \mathrm{LSPX} + \mathrm{PQ}\,(\text{target}), \quad \mathrm{LSPX} + \mathrm{R} \xrightarrow{\mathrm{k_1}} \mathrm{LSP} + \mathrm{RX}, \quad \mathrm{LSP} + \mathrm{Q} \xrightarrow{\mathrm{k_2}} \mathrm{LS} + \mathrm{PQ}\\
(2) & \mathrm{Q} + \mathrm{R} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}, \quad \mathrm{R} + \mathrm{Q} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}\\
(3) & \mathrm{R} + \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}, \quad \mathrm{Q} + \tau\mathrm{R} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}, \quad \tau\mathrm{R} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}, \quad \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}\\
(4) & \mathrm{LSPXP} + \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}^{chain}}} \mathrm{LS}\sigma + \mathrm{RX} + \mathrm{PQ}, \quad \mathrm{LSPX} + \tau\mathrm{R} \xrightarrow{\mathrm{k_{A}^{chain}}} \mathrm{LS}\sigma + \mathrm{RX} + \mathrm{PQ}
\end{array}}$$

RM7 (add *read* data operators R):

$$\boxed{\begin{array}{ll}
(1) & \mathrm{LSPX} + \mathrm{R} \xrightarrow{\mathrm{k_1}} \mathrm{LSP} + \mathrm{RX}\,(\text{target}), \quad \mathrm{LSP} + \mathrm{Q} \xrightarrow{\mathrm{k_2}} \mathrm{LS} + \mathrm{PQ},\\
(2) & \mathrm{Q} + \mathrm{R} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}, \quad \mathrm{R} + \mathrm{Q} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}\\
(3) & \mathrm{R} + \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}, \quad \mathrm{Q} + \tau\mathrm{R} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}, \quad \tau\mathrm{R} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}, \quad \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}\\
(4) & \mathrm{LSPX} + \tau\mathrm{R} \xrightarrow{\mathrm{k_{A}^{chain}}} \mathrm{LS}\sigma + \mathrm{RX} + \mathrm{PQ}
\end{array}}$$

RM8 (add *pop* data operators Q):

$$\boxed{\begin{array}{ll}
(1) & \mathrm{LSP} + \mathrm{Q} \xrightarrow{\mathrm{k_2}} \mathrm{LS} + \mathrm{PQ}\,(\text{target})\\
(2) & \mathrm{Q} + \mathrm{R} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}, \quad \mathrm{R} + \mathrm{Q} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}\\
(3) & \mathrm{R} + \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}, \quad \mathrm{Q} + \tau\mathrm{R} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}, \quad \tau\mathrm{R} \xrightarrow{\mathrm{k_A}} \tau\mathrm{Q}, \quad \tau\mathrm{Q} \xrightarrow{\mathrm{k_{BC}}} \tau\mathrm{R}
\end{array}}$$

B Determine initial state of CTMC for RM1

The initial state for generating CTMC from the first reaction model is assumed the following steps have been performed:

- 100 levels of (300nM) *linker* DNA data operators incubate with 100ul sepharose beads for sufficient time. It's assumed that all the DNA data operators can irreversible bind to beads which yields 100 levels of bead-bounded L stacks.
- A wash event is perform by assuming μ percent of the L stacks is removed. Given $\mu = 0.1$, 90 levels of L remain in the chemistry after the wash.

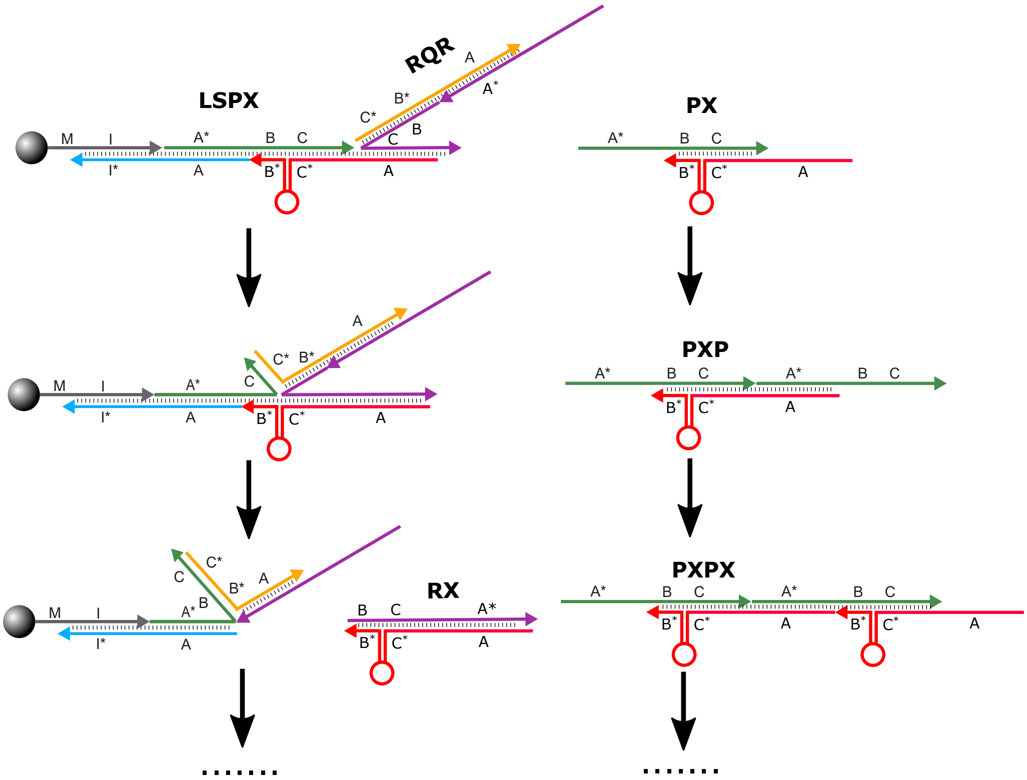
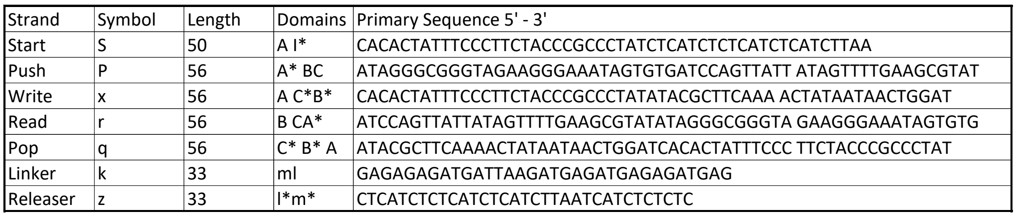


Figure 7 Examples of a hypothesised "chain annihilation" multi-stage reaction (left), and P, X polymer formation (right).

- 100 levels of *start* DNA data operators are added to hybridise with L. It is assumed that the irreversible linker-start reaction proceeded to 100% completion over the incubation time producing 90 levels of LS with 10 levels of supernatant S remain in the chemistry.
- A second wash event is perform by removing 10% LSP stacks, and 80% S (assuming $\phi = 0.2$). The result (i.e, LS = 81, S = 2) will be used as the initial state in the first CTMC.

C Experimental Validation

Table 2 DNA sequences for in vitro validation [20].



Strand	Symbol	Length	Domains	Primary Sequence 5' - 3'
Start	S	50	A I*	CACACTATTTCCCTTCTACCCGCCCTATCTCATCTCTCATCTCATCTTAA
Push	P	56	A* BC	ATAGGGCGGGTAGAAGGGAAATAGTGTGATCCAGTTATT ATAGTTTTGAAGCGTAT
Write	x	56	A C*B*	CACACTATTTCCCTTCTACCCGCCCTATATACGCTTCAAA ACTATAATAACTGGAT
Read	r	56	B CA*	ATCCAGTTATTATAGTTTTGAAGCGTATATAGGGCGGGTA GAAGGGAAATAGTGTG
Pop	q	56	C* B* A	ATACGCTTCAAAACTATAATAACTGGATCACACTATTTCCC TTCTACCCGCCCTAT
Linker	k	33	ml	GAGAGAGATGATTAAGATGAGATGAGAGATGAG
Releaser	z	33	I*m*	CTCATCTCTCATCTCATCTTAATCATCTCTCTC

For the sake of simplification, we only use data X to build stack in the experiments. This is because stacks are identified by their length, and it's unable to distinguish X stack (e.g, LSPXPX) and Y stack (e.g, LSPXPY) at the current settings as they are different in hairpin only. Such simplification, however, would not affect our model checking results, as the model excludes the situation that non-target stack becomes target stack in the later operation steps.

The Vgel simulator [20] is used to create a computer-generated image of what the DNA stack model output would like post releasing the stack from the beads. An example Vgel image is given in Figure 8. This Vgel can be combined with the species output to create a labelled image as we know the lengths of the strands when combined together to be: when signals are being pushed to the stack

- SP: 50 + 56 = 106
- SPX: 50 + 56 + 56 = 162
- SPXP: 50 + 56 + 56 + 56 = 218
- SPXPX: 50 + 56 + 56 + 56 = 274

When signals are being read from the stack or popped the length should drop by 56

- SPXPXR: 274 - 56 = 218
- SPXPXRQ: 218 - 56 = 162
- SPXPXRQR:162 - 56 = 106

Note that Linker is not part of this calculation as this creates its own complex of length 33. There are also other complexes which are formed by side reactions.

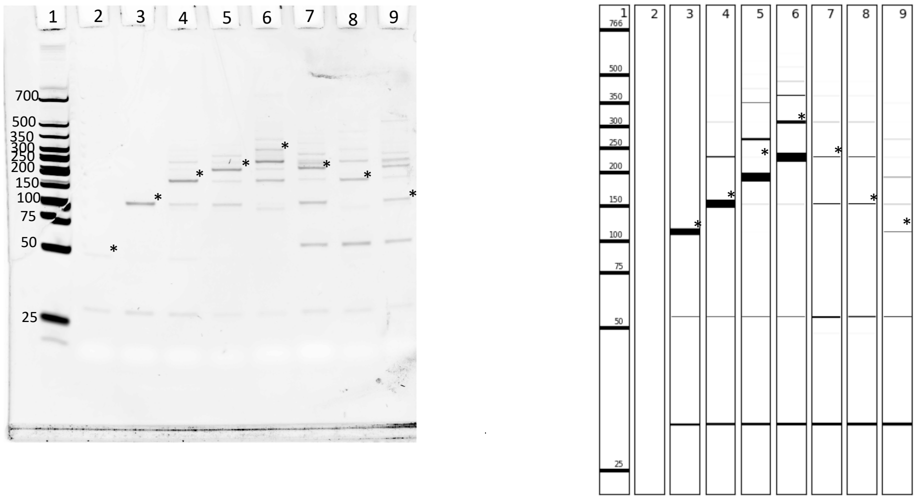


Figure 8 Full stack operation 30 minutes. A comparison of the gel electrophoresis and Vgel for a full stack operation with a 30 minute incubation. For each lane the target species has an asterisk next to it. Every other unlabeled species is a byproduct. Gel electrophoresis image: Lane 1: NEB low molecular ladder; Lane 2: S; Lane 3: SP; Lane 4: SPX; Lane 5: SPXP; Lane 6: SPXPX; Lane 7: SPXPXR; Lane 8: SPXPXRQ; and Lane 9:SPXPXRQR. Electrophoresis conditions: Gel:10 % Novex TBE Gel 1X TBE @200V for 35minutes Staining dye: Sybr Gold Concentrations: all DNA data operators were nominally 300nM Vgel image:Lane 1: NEB low molecular ladder. Lane 2: S; Lane 3: SP; Lane 4: SPX; Lane 5: SPXP; Lane 6: SPXPX; Lane 7: SPXPXR; Lane 8: SPXPXRQ; and Lane 9: SPXPXRQR.

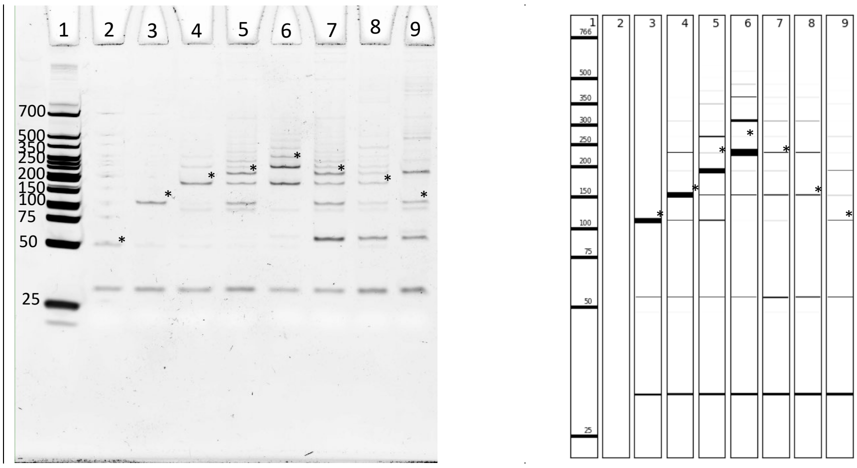


Figure 9 Variable timing as per the generated schema. A comparison of the gel electrophoresis and Vgel for a full stack operation with the incubation timings taken from the schema supplied by the model. For each lane the target species has an asterisk next to it. Every other unlabeled species is a byproduct. Gel electrophoresis image: Lane 1: NEB low molecular ladder; Lane 2: S; Lane 3: SP; Lane 4: SPX; Lane 5: SPXP; Lane 6: SPXPX; Lane 7: SPXPXR; Lane 8: SPXPXRQ; and Lane 9: SPXPXRQR. Electrophoresis conditions: 10% Novex TBE Gel in a buffer of 1X TBE @200V for 35minutes. Staining dye: Sybr Gold Concentrations:all DNA data operators were nominally 300nM Vgel image: Lane 1: NEB low molecular ladder; Lane 2: S; Lane 3: SP; Lane 4: SPX; Lane 5: SPXP; Lane 6: SPXPX; Lane: 7: SPXPXR; Lane 8: SPXPXRQ; and Lane 9: SPXPXRQ.

On Turedo Hierarchies and Intrinsic Universality

Samuel Nalin
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, FR-45067 Orléans, France

Guillaume Theyssier
I2M, CNRS, Université Aix-Marseille, France

Abstract

This paper is about turedos, which are Turing machines whose head can move in the plane (or in a higher-dimensional space) but only in a self-avoiding way, by putting marks (letters) on visited positions and moving only to unmarked, therefore unvisited, positions. The turedo model has been introduced recently as a useful abstraction of oritatami systems, which where established a few years ago as a theoretical model of RNA co-transcriptional folding. The key parameter of turedos is their lookup radius: the distance up to which the head can look around in order to make its decision of where to move to and what mark to write. In this paper we study the hierarchy of turedos according to their lookup radius and the dimension of space using notions of simulation up to spatio-temporal rescaling (a standard approach in cellular automata or self-assembly systems). We establish that there is a rich interplay between the turedo parameters and the notion of simulation considered. We show in particular, for the most liberal simulations, the existence of 3D turedos of radius 1 that are intrinsically universal for all radii, but that this is impossible in dimension 2, where some radius 2 turedo are impossible to simulate at radius 1. Using stricter notions of simulation, intrinsic universality becomes impossible, even in dimension 3, and there is a strict radius hierarchy. Finally, when restricting to radius 1, universality is again possible in dimension 3, but not in dimension 2, where we show however that a radius 3 turedo can simulate all radius 1 turedos.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Formal languages and automata theory

Keywords and phrases Turedos, intrinsic universality, Higher-dimensional Turing machines, Oritatami systems

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.6

1 Introduction

The field of biomolecular computing has given rise to several theoretical models that describe growing process of (molecular) assemblies governed by local interaction or gluing rules. One of the most studied one, the abstract Tile Assembly model (aTAM) [19], describes a process where the growth can happen anywhere asynchronously. It was successfully implemented in vitro as DNA self-assembly [18]. In oritatami systems, introduced more recently [9, 8] and inspired from RNA origami [11, 7], the growth happens at a unique given point of the assembly and in a sequential manner. Despite their obvious differences as models of biomolecular systems, they share common features as computational models. Both are limited by the fact that they can only grow shapes (and not erase them), but both were shown capable of embedding universal computations in different ways [10, 12, 3, 14, 19]. Recently, it was also shown through new results on oritatami systems that the infinite limit shapes both models can generate from finite seeds have the same computational complexity and the same possible densities of occupied position in the plane [17]. The key ingredients of these new results were, on one hand, the introduction of a new model, called *turedo*, which abstracts away low level details of oritatami systems and is easier to program, and on the other hand, a proof that oritatami systems can simulate a large family of turedos. This underlines the interest of turedos but also the surprising capabilities of models of sequential

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham; Article No. 6; pp. 6:1–6:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

growth process in the plane. The present work is entirely focused on the turedo model and aims at better understanding its limitations and its expressive power as a growth process and computational model.

Turedos. Intuitively, a turedo is a Turing machine whose head can move in the plane or in the 3-dimensional space but only in a self-avoiding way, *i.e.* without going back to a previously visited position. More precisely, the turedo's head can only move to empty or unmarked positions, and it must put a mark on each visited position when leaving it. The key ingredient of a turedo, and what makes its main computational power, is its *lookup radius*: when deciding a move and what mark to put on the visited position, the turedo has access to the local configuration of marks around its position, up to some finite distance. Like any Turing machine, a turedo also has a set of internal head states. Without entering into details, an oritatami system consists of a "molecule" made of "beads" that can attract each other. The molecule grows at each step following a periodic bead sequence and folds as follows: the δ most recently produced beads are free to move around to look for the position that maximizes the number of bonds they can make with each other ; then the first (oldest) bead among the δ most recent ones is fixed according to that position, a new free bead is added and the process iterates. This behavior can be realized in a turedo of radius $\delta + 1$. The main result of [17] is that oritatami of delay 3 can simulate turedos of radius 1. In fact, turedos also naturally capture variants of oritatami systems: for instance, one can imagine negative (repulsive) bonds in the process of maximizing gluing strength, or add local rules forbidding two bead types to be neighbor of each other. Therefore negative results on turedos become negative results on oritatami systems, but also potential variants of them.

Simulations and universality. The main question we address here is how the capabilities of turedos change with their radius and what is the role of the dimension of space. We are interested in qualitative differences and don't compare turedos move by move and cell by cell. We rather use a notion of simulation allowing spatio-temporal rescaling, similar to the one used in cellular automata [4, 1], in aTAM [6, 5, 16] or in the simulation result of turedo by oritatamis [17]. Intuitively a cell can be simulated by a block of cells, and a step by a finite number of steps. We are naturally interested in hierarchy results (existence of turedos that can't be simulated by lower radius ones), or on the contrary by universality results (existence of turedos that can simulate all turedos or a large class of them). For instance, the existence of intrinsically universal systems in the aTAM model was much studied and it was shown that it crucially depends on natural parameters of the model [6, 16, 5, 13]. However, as natural as it might seem, a formal notion of simulation is never a neutral choice when tackling these questions [1, 4, 2]. Thus, in addition to the parameters of the turedo model we also study the influence of the notion of simulation itself. For this we identify various ingredients, in particular the possibility of fuzz, *i.e.* the tolerance allowed for the simulating turedo to visit some regions of space close to the ongoing assembly that represent empty cells of the simulated turedo and have therefore not yet been visited by it. Note that a similar notion of fuzz appeared in the intrinsic universality results on aTAM [5]. We end up with three notions of simulation: the rigorous, the fuzzless and the liberal ones.

Our contributions. After formalizing all the concepts mentioned so far (Section 2), we establish a surprisingly diverse general picture of the capabilities of turedos of simulating each other. Separating the negative results (Section 3) from the positive ones (Section 4), our results are the following:

1. under fuzzless simulation, intrinsic universality is impossible whatever the dimension, there is a radius hierarchy, and actually the impossibility strikes at radius 2: no turedo can fuzzlessly simulate all radius 2 turedos (Theorem 5);
2. when restricting to radius 1, rigorous intrinsic universality is possible in dimension 3 (Theorem 11), but not in dimension 2 (Theorem 6);
3. however, we built a 2D turedo of radius 3 which is able to rigorously simulate all 2D turedo of radius 1 (Theorem 10);
4. for liberal simulations, we establish intrinsic universality in dimension 3 and a complete hierarchy collapse at radius 1 (Corollary 13);
5. finally we show that there is a 2D turedo of radius 2 which is impossible to simulate at radius 1, even under liberal simulations (Theorem 9).

Besides the above results, our contribution also lies in the constructions and proof techniques. For instance, the negative result 5 is based on a general lemma for 2D turedos of radius 1, which bounds the quantity of information (using Kolmogorov complexity) that can be carried from the seed to another connected component of the plane when the plane is divided by a 4-connected path. As another example on the constructive side, result number 3 above uses a novel construction technique (called heat sink trick) that fully exploits the potential of radius 3 and could be used in any place where two unbounded streams of information have to be crossed. Finally, in Section 5, we discuss various questions left open and present what we believe are promising future research directions.

2 Definitions

We denote by $\mathbb{N}$ the set of natural numbers (including 0), by $\mathbb{N}_+$ the positive ones, and by $\mathbb{Z}$ the set of integers. We consider turedos on $\mathbb{Z}^d$ for $d = 2$ or 3 (and in particular, we don't use the hexagonal lattice on the plane, mostly to simplify notations and dimension change). We fix a blank symbol $\perp$used to represent empty positions and common to all turedos. The ball of radius r in dimension d, denoted $B_d(r)$, is the set of positions reachable in r elementary moves (moves along vector of the canonical base of $\mathbb{Z}^d$) from the origin. $B_d(1)$ will always be the set of possible head moves in dimension d. We denote by $c[z; S]$ the pattern of shape S around position z in configuration c, *i.e.* the map $z' \in S \mapsto c_{z+z'}$.

▶ **Definition 1** (Turedo). *A* turedo *of dimension d and radius r is a triple $T = (A, Q, \delta)$ where A is its finite* alphabet *with $\perp \in A$, Q is its finite set of* head states *and*

$$\delta : Q \times A^{B_d(r)} \to Q \times A \setminus \{\perp\} \times B_d(1)$$

is its local transition map*. A* global state *of T is a triple $(c, z, q) \in \mathcal{G}_T = A^{\mathbb{Z}^d} \times \mathbb{Z}^d \times Q$ specifying a configuration, a position and a head state. The* global transition map *$F_T : \mathcal{G}_T \to \mathcal{G}_T$ associated to T is defined by:*

$$F_T(c, z, q) = \begin{cases} (c, z, q) & \text{if } c(z) \neq \perp \text{ or } c(z+\mu) \neq \perp \\ (c', z+\mu, q') & \text{else,} \end{cases}$$

where $(q', a, \mu) = \delta(q, c[z; B_d(r)])$ and configuration c' is defined by $c'(z) = a$ and $c'(z') = c(z')$ for all $z' \neq z$.

The *domain* of a global state (c, z, q) is the set of non-blank positions of c plus the head position, formally: $\mathcal{D}(c, z, q) = \{z\} \cup \bigcup\{z' : c(z') \neq \perp\}$ A global state (c, z, q) is finite if its domain is finite. We are interested in orbits starting from finite initial global states, called finite seeds.

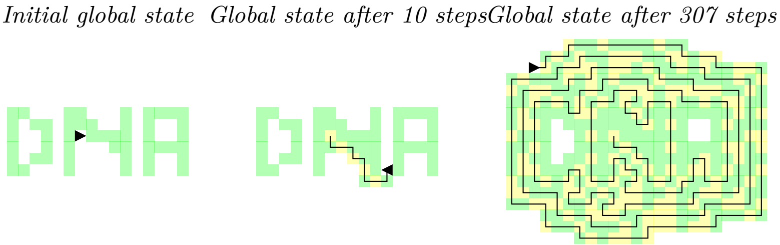


Figure 1 Example of orbit of the spiral-XOR turedo (example 2) starting from a finite seed. Green represents 0, yellow represents 1 and white represents $\perp$. The head holds a direction (where the black triangle is pointing to) and its self-avoiding trajectory since the beginning is drawn as a black path.

▶ **Example 2** (The spiral-XOR turedo). Let $A = \{\perp, 0, 1\}$ and $Q = \{\leftarrow, \uparrow, \rightarrow, \downarrow\}$. The *spiral-XOR* turedo has radius 1 and the following local rule. The head holds a direction $d \in Q$ and tries to move in that direction and leave behind, as letter of A, the sum modulo 2 of the states of neighboring (non $\perp$) positions. When it can do the d move, it changes its internal state (counter-clockwise), when it can't it takes the first available move (clockwise) and doesn't change its state. Of course if there is no neighbor in state $\perp$ then the turedo is blocked. The following table shows the local transition map up to rotation of d (the red arrow can be rotated and all blue arrows are defined relatively to the red arrow):

state	1st empty neighbor	new state	letter	move
↑	↑ (+0)	← (-1)	$\sum$ mod2	↑ (+0)
↑	→ (+1)	↑ (+0)	$\sum$ mod2	→ (+1)
↑	↓ (+2)	↑ (+0)	$\sum$ mod2	↓ (+2)
↑	← (+3)	↑ (+0)	$\sum$ mod2	← (+3)

See Figure 1 for an example of orbit.

The main focus of this paper is to understand the role of radius and dimension in the computational complexity of turedos. We will denote by $\mathrm{TUR}_d(r)$ the set of turedos of dimension d and radius r, and $\mathrm{TUR}_d = \bigcup_{r\geq 1} \mathrm{TUR}_d(r)$.

Before formalizing simulation, we first need to define *block encodings* which are ways to represent global states of a simulated turedo by blocks in the simulator. Any given $b \in \mathbb{N}_+^d$ defines a rectangular block $R_b = \{z \in \mathbb{N}^d : 0 \leq z_i < b_i \text{ for } 1 \leq i \leq d\}$ and Z^d can be tiled by translated copies of R_b in a regular way by placing them on the sublattice $b \otimes \mathbb{Z}^d$ where $\otimes$ denotes the component-wise product. Each position $z \in \mathbb{Z}^d$ can be uniquely decomposed into $z = \rho_b(z) + \mu_b(z)$ where $\rho_b(z) \in b \otimes \mathbb{Z}^d$ is the reference point of a block and $\mu_b(z) \in R_b$ is an offset inside it.

▶ **Definition 3** (Block encoding). *Let us fix a dimension d. Given two pairs of alphabets and state sets (A_1, Q_1) and (A_2, Q_2) with $\perp \in A_1 \cap A_2$, a* block encoding *of global states $\mathcal{G}_1 = A_1^{\mathbb{Z}^d} \times \mathbb{Z}^d \times Q_1$ into $\mathcal{G}_2 = A_2^{\mathbb{Z}^d} \times \mathbb{Z}^d \times Q_2$ is given by a* block size *$b \in \mathbb{N}_+^d$ and two* partial onto *maps:*

- *the* headless block decoding map *$\alpha : D_\alpha \subseteq A_2^{R_b} \to A_1$ verifying $\perp^{R_b} \in D_\alpha$ and $\alpha(\perp^{R_b}) = \perp$,*
- *the* head block decoding map *$\beta : D_\beta \subseteq A_2^{R_b} \times R_b \times Q_2 \to Q_1 \times A_1$.*

A global state $(c, z, q) \in \mathcal{G}_2$ is valid *for the encoding if it is made only of patterns from D_α far from the head and D_β around the head, precisely if: $(c[\rho_b(z); R_b], \mu_b(z), q) \in D_\beta$ and $c[\rho_b(z'); R_b] \in D_\alpha$ for all $z' \in \mathbb{Z}^d$ such that $\rho_b(z') \neq \rho_b(z)$.*

Finally, the global decoding map Γ *associates to any valid global state* $(c_2, z_2, q_2) \in \mathcal{G}_2$ *a global state* $(c_1, z_1, q_1) \in \mathcal{G}_1$ *defined by application of decoding maps* α *or* β *on each block according to the presence of the head in the block, i.e. :*

- $b \otimes z_1 = \rho_b(z_2)$,
- $(q_1, c_1(z_1)) = \beta(c_2[\rho_b(z_1); R_b], \mu_b(z_2), q_2)$,
- $c_1(z) = \alpha(c_2[\rho_b(z_2); R_b])$ *for all* $z \neq z_1$*.*

The map α and β being partial and onto intuitively means that not all global states are valid, and that any global state of $A_1^{\mathbb{Z}^d} \times \mathbb{Z}^d \times Q_1$ can be encoded. Note that the headless block decoding map α always decodes blank blocks $\perp^{R_b}$ as blank state $\perp$. Denote by $\mathcal{D}_b(c_2, z_2, q_2)$ the *block domain* of global state (c_2, z_2, q_2) which is the set of blocks that are not entirely blank, *i.e.* $\mathcal{D}_b(c_2, z_2, q_2) = \{z : b \otimes z = \rho_b(z_2) \text{ or } c_2[b \otimes z; R_b] \neq \perp^{R_b}\}$.

We can now define simulations precisely using block encodings. Intuitively, we require the simulator to be able to reproduce any orbit of the simulated turedo starting from a finite seed, and using a (fixed) finite number of steps to simulate one step. Since we want the initial seed of the simulator to be neutral and without any pre-computed information about the future of the simulated orbit, we ask that its block domain correspond to the domain of the simulated seed.

▶ **Definition 4** (Simulation). *Let d be a fixed dimension. We say that a d-dimensional turedo* T_2 simulates *a d-dimensional turedo T_1 if there is a block encoding of $\mathcal{G}_{T_1}$ into $\mathcal{G}_{T_2}$ of bock size b and global decoding map Γ, and a time scaling factor $k \in \mathbb{N}_+$, such that for each finite global state $(c_1, z_1, q_1) \in \mathcal{G}_{T_1}$ and each global state $(c_2, z_2, q_2) \in \mathcal{G}_{T_2}$ verifying:*

- *corresponding block domain: $\mathcal{D}(c_1, z_1, q_1) = \mathcal{D}_b(c_2, z_2, q_2)$,*
- *correct encoding: $(c_1, z_1, q_1) = \Gamma(c_2, z_2, q_2)$,*

then it holds $\forall t \in \mathbb{N}, F_{T_1}^t(c_1, z_1, q_1) = \Gamma\big(F_{T_2}^{kt}(c_2, z_2, q_2)\big)$. Such simulations are the base upon which we define three variants (two restrictions and one generalization).

We say that a simulation is fuzzless *if the headless block decoding map is such that $\alpha(u) = \perp \iff u = \perp^{R_b}$, i.e. that the only block coding $\perp$ is $\perp^{R_b}$.*

We say that a simulation is rigorous *if the movements of the head of T_2 in simulating orbits strictly remains inside blocks corresponding to the simulated head position of T_1, even at intermediate steps, precisely: if z_1^t denotes the head position of $F_{T_1}^t(c_1, z_1, q_1)$ and z_2^t that of $F_{T_2}^t(c_2, z_2, q_2)$, it holds for all t' with $kt \leq t' \leq k(t+1) : z_2^{t'} \in (b \otimes z_1^t + R_b) \cup (b \otimes z_1^{t+1} + R_b)$.*

Finally, a liberal simulation *is a generalized simulation where we only ask that for each finite global state $(c_1, z_1, q_1) \in \mathcal{G}_{T_1}$ there exists a global state $(c_2, z_2, q_2) \in \mathcal{G}_{T_2}$ with corresponding block domain and correct encoding such that it holds $\forall t \in \mathbb{N}, F_{T_1}^t(c_1, z_1, q_1) = \Gamma\big(F_{T_2}^{kt}(c_2, z_2, q_2)\big)$.*

Liberal simulations can have fuzz and make non-rigorous head movements. Note that fuzzless simulations are equivalent to simulations where the block domain in the simulator orbit remains identical to the domain of the simulated orbit, precisely: $\forall t \in \mathbb{N} : \mathcal{D}_b(F_{T_2}^{kt}(c_2, z_2, q_2)) = \mathcal{D}(F_{T_1}^t(c_1, z_1, q_1))$.

Note also that a rigorous simulation is necessarily fuzzless because the head of the simulator has no opportunity, even at intermediate time steps, to visit blocks not corresponding to the domain of the simulated configuration. The power of fuzzless simulations compared to rigorous ones is to allow the head to go back to blocks that were previously visited. Thus the head can potentially retrieve information from non adjacent blocks that were written a long time ago. With rigorous simulation on the contrary, the head has only access to adjacent blocks during a simulation cycle.

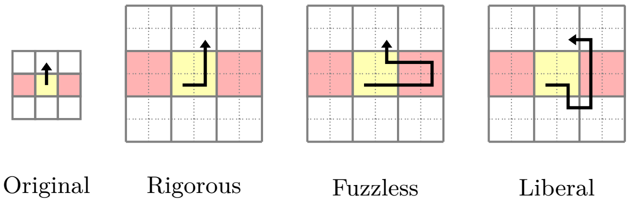


Figure 2 Differences in allowed head movements in rigorous, fuzzless and liberal simulations with 2×2 blocks. The colors have the following meaning: in red the positions or blocks which are not empty initially, in yellow the positions or blocks coding a non-$\perp$ letter during the orbit; in white the positions or blocks coding $\perp$; in black the movement of the head.

We denote by $\leq$ the liberal simulation, by $\leq_{FL}$ the fuzzless simulation and by $\leq_R$ the rigorous simulation. Among many properties of these simulation relations, we are particularly interested in universality: the capacity of a single turedo to simulate a whole set of turedos. Given a dimension d and a radius r, we denote by $\mathcal{U}_d^{\leq}(r)$ the set of turedos $T \in \mathrm{TUR}_d$ such that for any $T' \in \mathrm{TUR}_d(r)$ it holds $T' \leq T$. We denote by $\mathcal{U}_d^{\leq}$ the set of turedos $T \in \mathrm{TUR}_d$ such that for any $T' \in \mathrm{TUR}_d$ it holds $T' \leq T$. We use similar notations for simulation relations $\leq_R$ and $\leq_{FL}$.

3 Separation Results

3.1 No Fuzz, no Fun

The fuzzless condition gives much importance to larger radii, simply because a turedo's head surrounded by blocks coding $\perp$ cannot read information far away without moving inside these blocks and thus creating fuzz. The following theorem exploits this obvious limitation to show two immediate consequences: first, there is a radius hierarchy (new behaviors appear at radius $r+1$ that cannot be simulated at radius r) and thus no general fuzzless universality; second, universality is impossible even at radius 2: any turedo (whatever its radius) will fail to simulate some radius-2 turedo. The dimension plays no role in these results.

▶ **Theorem 5.** *For any $d \geq 2$ and $r \geq 1$, we have the following:*

- *there is $T_{r+1} \in TUR_d(r+1)$ such that for all $T_r \in TUR_d(r)$, $T_{r+1} \not\leq_{FL} T_r$; in particular, $\mathcal{U}_d^{\leq_{FL}} = \emptyset$.*
- *for any $T_r \in TUR_d(r)$ there exists $T_2 \in TUR_d(2)$ such that $T_2 \not\leq_{FL} T_r$; in particular, $\mathcal{U}_d^{\leq_{FL}}(r) = \emptyset$ for any $r \geq 2$.*

3.2 Dimension 2 and Radius 1: the Jordan Curve Burden

A turedo's head in dimension 2 always moves drawing a 4-connected path. When the turedo has radius 1, it has no way to read information across such a path (while it could with a larger radius). Therefore head movements for turedos of radius 1 turn into potential information barriers. The precise way in which this simple observation affects the simulation power of such turedos depends on the type of simulation considered.

Let us first consider rigorous simulations. Any turedo T (whatever its radius) can obviously do the following elementary copy-and-move operation (see Figure 3):

- move to the right to some position z;
- read the letter a present at position $z + (1,0)$;
- then move to position $z + (0,1)$ and leave behind letter a at position z.

In particular, if the head continues its way and later arrives at position $z - (0,1)$ from the south, it can read the information a copied at position z.

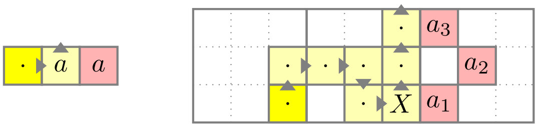


Figure 3 Example of a copy-and-move operation (on the left) and its rigorous simulation by a turedo of radius 1 with 3×3 block size (on the right). The color convention is as follows: in red the letters present in the seed, in dark yellow the initial position of the head, and in light yellow, the positions visited by the head during the orbit. The only position on the south side of the middle block that can depend on a is the lower right corner, marked with a X.

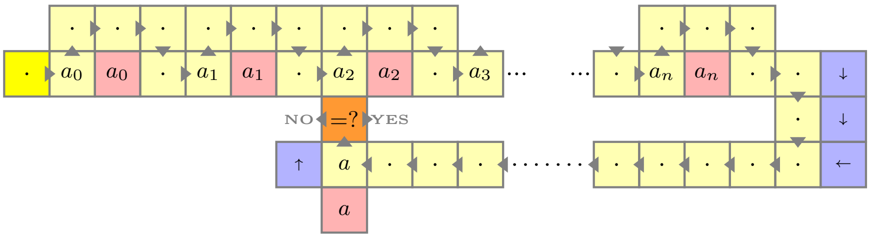


Figure 4 Behavior of turedo T' on the seed $\sigma(\vec{a}, 2, a)$. The red and blue colors indicate letters present in the seed. The dark yellow color indicates the initial position of the head and light yellow cells represent the path of the head until the last step of the orbit. The orange cell correspond to the position of the head before moving left or right according to the result of the test $a \overset{?}{=} a_2$.

However, if we suppose that some $T_1 \in \mathrm{TUR}_2(1)$ simulates T under rigorous simulations with block size b, the movement of its head inside block $b \otimes z$ to simulate the above copy-and-move step must be the following (see Figure 3):

- coming from the left side of the block, it draws some path inside it until it reaches the right border (if not it cannot read any information from the adjacent block to the right);
- then it must move north, otherwise it would be trapped in the south part of the blocks by the 4-connected path drawn so far that connects the left and right sides of the block;
- it must finally escape through the north side.

This head movement is such that at most 1 letter of T_1 is written on the south side of block $b \otimes z$ after having had the opportunity to read some information from the adjacent block that encodes letter a. In particular, if T_1 has smaller alphabet than T this is not enough to completely encode a on the south side of block $b \otimes z$. So, this is a limitation that has to be dealt with if later in the simulation the head arrives from the south and has to read from the south side of block $b \otimes z$. A single copy-and-move is not enough to get a contradiction because the simulation of T_1 could be organized so as to transport the complete information about a along the way and have it on hands already when arriving at the south of block $b \otimes z$. However, by repeating such copy-and-move steps, one can saturate the simulator and show the following theorem that states that there is no universal turedo of radius 1 among turedos of radius 1 for rigorous simulations.

▶ **Theorem 6.** *For any $T \in TUR_2(1)$ there is $T' \in TUR_2(1)$ such that $T' \not\leq_R T$. In particular $\mathcal{U}_2^{\leq_R}(1) \cap TUR_2(1) = \emptyset$.*

Proof. Let Q be the state set of T, A be the alphabet of T and consider any alphabet A_+ with $m = |A| < |A_+| = m_+$. Then it is straightforward to construct a turedo $T' \in \mathrm{TUR}_2(1)$ of alphabet $A' = A_+ \cup \{\downarrow, \leftarrow, \uparrow\}$ that has the following behavior (see Figure 4):

- for any $n \in \mathbb{N}$, any $\vec{a} = (a_0, \ldots, a_n) \in A_+^{n+1}$, $a' \in A_+$ and $0 \leq i \leq n$, consider the finite seed $\sigma(\vec{a}, i, a')$ with head in position $(0, 0)$, a_j in position $(3j + 2, 0)$ for $0 \leq j \leq n$, a' in position $(3i + 1, -3)$ and $\downarrow$ in positions $(3(n + 1) + 2, 0)$ and $(3(n + 1) + 2, -1)$, $\leftarrow$ in position $(3(n + 1) + 2, -2)$ and finally $\uparrow$ at position $(3i, -2)$;
- from such a seed, T' starts a sequence of $n + 1$ copy-and-move steps that results in having a copy of a_j at position $(3j + 1, 0)$ for $0 \leq j \leq n$; the end of this phase occurs at time step $5(n + 1)$ and the head reaches position $(3(n + 1), 0)$;
- then T' reaches the first $\downarrow$ and follows the move indications of arrows (down, down, left), until it reaches the up arrow, and moves from position $(3i + 1, -2)$ to $(3i + 1, -1)$;
- finally, at position $(3i + 1, -1)$ it moves right if $a_i = a'$ and left otherwise (it can do so because it has copied the value of a' when leaving position $(3i + 1, -2)$).

Let's call $t_{n,i}$ the time step at which occurs this final left or right move ($t_{n,i}$ only depends on i and n): at time $t_{n,i}$, the head of T' must be either at position $(3i, -1)$ or $(3i + 2, -1)$. Thus T' implements on seed $\sigma(\vec{a}, i, a')$ the test of whether $a_i = a'$. We are going to show that T cannot simulate T' under rigorous simulations. Suppose by contradiction that $T' \leq_R T$ with block size b and time scaling factor k. Given $n \in \mathbb{N}$ and $0 \leq i \leq n$, denote by $A_n \subseteq \mathbb{Z}^2$ the set of positions that are on the right side of block $b \otimes (3(n + 1), 0)$ (the block corresponding to the position reached by T' at the end of the copy-and-move sequence as detailed above). Denote by $B_{i,n} \subseteq \mathbb{Z}^2$ the set of positions that are on the south side of block $b \otimes (3i + 1, 0)$. Finally, denote by $C_{i,n} \subseteq \mathbb{Z}^2$ the set of positions made of the union of blocks $(3i + 1, -3)$, $(3i, -2)$, $(3(n + 1) + 2, 0)$, $(3(n + 1) + 2, -1)$, $(3(n + 1) + 2, -2)$ (*i.e.* those corresponding to position a' or an arrow $\{\downarrow, \leftarrow, \uparrow\}$ in the seed $\sigma(\vec{a}, i, a')$). Consider now $n \in \mathbb{N}$, $\vec{a}, \vec{c} \in A_+^{n+1}$ $a' \in A_+$ and $0 \leq i \leq n$, and take any two global states g_1 and g_2 of T that correctly simulate the orbits of T' on seed $\sigma(\vec{a}, i, a')$ and $\sigma(\vec{c}, i, a')$ respectively and that are identical on $C_{i,n}$ Considering time step $t_0(n) = 5k(n + 1)$ corresponding to the end of the copy-and-move sequence, if global states $T^{t_0(n)}(g_1)$ and $T^{t_0(n)}(g_2)$ are identical on domains A_n and $B_{i,n}$ and have the same head state, then both orbits must make the same final decision to move to the left block or the right block at the final time step $kt_{n,i}$, precisely: the head in global state $T^{kt_{n,i}}(g_1)$ is in the same block as the head in global state $T^{kt_{n,i}}(g_2)$ (and it must be either $b \otimes (3i, -1)$ or $b \otimes (3i + 2, -1)$). Indeed, by the property of rigorous simulations and the behavior of T', the only positions with content written before $t_0(n)$ that the head of T can possibly read between time step $t_0(n)$ and $kt_{n,i}$ are positions in $A_n \cup B_{i,n} \cup C_{i,n}$, so the orbit starting from step $t_0(n)$ is completely determined by the content of the configuration in that domain and the internal state of T at time $t_0(n)$.

▷ Claim 7. There must exist $n \in \mathbb{N}$, $0 \leq i \leq n$, $\vec{a} \in A_+^{n+1}$, $a' \in A_+$ and $\vec{c} \in A_+^{n+1}$ with $a_j = c_j$ for all $j < i$ and $a_i \neq c_i$, and two global states g_1 and g_2 of T that correctly simulate seeds $\sigma(\vec{a}, i, a')$ and $\sigma(\vec{c}, i, a')$ respectively, and also such that $T^{t_0(n)}(g_1)$ and $T^{t_0(n)}(g_2)$ have same head state and are identical on domain $A_n \cup B_{i,n} \cup C_{i,n}$.

Proof of the claim. In this proof, we fix for each $a \in A_+$ a unique block of A^{R_b} that encodes it, and for any seed of T' we only consider a unique global state of T that simulates it. First, there are only a bounded number (bound in n) of possible content of a configuration on domain A_n and state of T, so for each n there must exist $u \in A^{A_n}$ and $q \in Q$, a set $X_n \subseteq A_+^n$ of size $\Omega(m_+^n)$ such that for each $0 \leq i \leq n$ and each $\vec{a} \in X_n$, the corresponding global state g of T simulating T' on seed $\sigma(\vec{a}, i, n)$, is such that $T^{t_0(n)}(g)$ is equal to u on domain A_n and with head state q.

Second, we claim that for large enough n there must be some i and a prefix $a_0, \ldots, a_{i-1} \in A_+^i$ such that there are at least $m + 1$ choices of $a_i \in A_+$ such that $a_0, \ldots, a_i$ can be completed into an element $\vec{a} \in X_n$. Indeed, otherwise we would have $|X| \leq m^n$ which would contradict the fact that $|X| \in \Omega(m_+^n)$ for large enough n since $m < m_+$.

Now consider the set of global states that simulates the seeds $\sigma(\vec{a}, i, a')$ where $\vec{a} \in X_n$ are the $m+1$ completed vectors from the common prefix $a_0, \ldots, a_{i-1}$, and $a' \in A_+$. They are identical on the blocks corresponding to the common prefix $a_0, \ldots, a_{i-1}$ of the seed they simulate. As already said, these global states at step $t_0(n)$ are also identical on domain A_n and have same head state. Moreover, on domain $B_{i,n}$ and still at step $t_0(n)$, they agree because of the common prefix $a_0, \ldots, a_{i-1}$, except possibly on the lower-right corner where they can take at most m different values (see Figure 3 and discussion at the beginning of this section). We deduce that among the $m+1$ choices for a_i, at least 2 must correspond to global states that completely agree on $B_{i,n}$. Denote by a' and c' these two choices and consider $\vec{a}$ and $\vec{c}$ to be the vectors completing the prefixes $a_0, \ldots, a_{i-1}, a'$ and $a_0, \ldots, a_{i-1}, c'$ respectively. The claim follows by choosing seeds $\sigma(\vec{a}, i, a')$ and $\sigma(\vec{c}, i, a')$. ◁

The theorem follows from the claim by contradiction: as shown above, global states g_1 and g_2 force the same behavior of T starting from time $t_0(n)$, but at the same time their orbits should not end up in the same block because they simulate seeds of T' that do not have the same answer to the final equality test. ◀

We will now establish a strong separation between $\mathrm{TUR}_2(1)$ and $\mathrm{TUR}_2(2)$ even under liberal simulations. We first establish a lemma expressing bounds on information leakage between two regions separated by a 4-connected path. It is formulated using Kolmogorov complexity. Recall that the (plain) Kolmogorov complexity of a string $u \in \{0,1\}^*$ is the length of the shortest program that outputs u, more precisely the length of the shortest $v \in \{0,1\}^*$ such that a suitable fixed universal Turing machine outputs u on input v (see [15]). For any $X \subseteq \mathbb{Z}^2$ and any (partial) configuration $c \in Q^X$ of finite domain, we denote by $K(c)$ its Kolmogorov complexity, which is the Kolmogorov complexity of the finite binary string u that encodes c as a list of pairs $(z, c(z))$ such that $c(z) \neq \bot$ given in lexicographical order.

▶ **Lemma 8.** *Let $C_0 \in \mathbb{N}$ be some constant and $T \in TUR_2(1)$. Then there is another constant $C \in \mathbb{N}$ with the following property. Consider any 4-connected path ρ of $\mathbb{Z}^2$ that divides $\mathbb{Z}^2$ in 2 or more connected components, and any finite global state $s \in \mathcal{G}_T$ with head at position $(0,0)$, and whose domain $\mathcal{D}(s)$ lies entirely in one of the connected components defined by ρ, denoted A_0. Suppose moreover that for some $n \in \mathbb{N}$, the orbit from global state s to global state $(c, z, q) = F_T^n(s)$ is such that the head visits path ρ at most C_0 times. Then, the restriction of c to the complement of A_0 has 'small' Kolmogorov complexity: $K(c_{|\mathbb{Z}^2 \setminus A_0}) \leq C \log(n)$.*

Note that if the seed s has 'large' Kolmogorov complexity, for instance $\Omega(n)$, then n steps are far from enough to transmit all the information about s to another connected component under the hypothesis of the lemma. The power of this lemma lies in the fact that constant C does not depend on the path ρ nor on the seed s. In particular, one can choose ρ depending on s to apply the lemma. Turedos of radius 2 can overcome the limitation of Lemma 8 because they can transmit information over a path without writing on it. It turns out that this is enough to separate $\mathrm{TUR}_2(2)$ from $\mathrm{TUR}_2(1)$ even under liberal simulations.

▶ **Theorem 9.** *There is $T_2 \in TUR_2(2)$ such that for any $T_1 \in TUR_2(1)$: $T_2 \not\preccurlyeq T_1$.*

Proof. Let's consider the turedo $T_2 \in \mathrm{TUR}_2(2)$ that behaves as follows on a seed made of a vertical word u of length n (see Figure 5):

- it copies u to the right by making zigzags and does this n times (it implements a unary counter initialized to the length of u while making copies);
- it then moves one cell to the right without making copies (and thus leaving an empty column above);

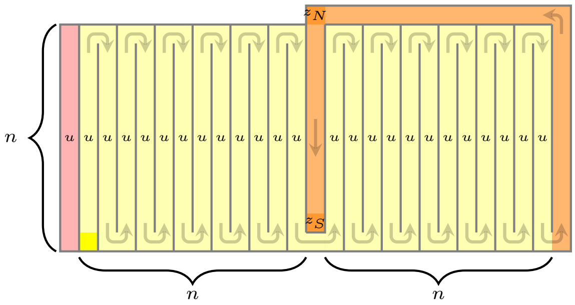


Figure 5 Orbit of turedo T_2 starting from a seed u of length n (in red) with the head initially in the position shown in dark yellow. The last part of the orbit is shown in orange.

- it then does again n copies of u by zigzag while moving to the right (note that the first copy can be done because T has radius 2);
- at the end of the last copy it goes around the last bloc of n copies by the north side until it encounters the empty column and then goes down into it until it is blocked.

Denote by z_N and z_S the northmost and southmost positions of the last sequence of n south moves of the head (see Figure 5), and by t_N and t_S the respective time steps at which the head is at position z_N and z_S. Note that t_S is $O(n^2)$ and it is the final step of the orbit considered here.

Now suppose by contradiction that there is some $T_1 \in \mathrm{TUR}_2(1)$ such that $T_2 \leq T_1$. Denote by k the time rescaling factor and b the block size involved in this simulation. Let's suppose that n is large enough (to be given precisely later) and that u has large Kolmogorov complexity, let's say $\Omega(n)$. Consider a global initial state s_1 for T_1 from which starts a correct simulation of the run of T_2. When the simulation reaches step kt_N, the head of T_1 is inside bloc $b \otimes z_N$ and there must be a finite 4-connected path $p_1, \ldots, p_m$ of empty positions from this position to some position inside bloc $b \otimes z_S$ because T_2 has to simulate the state changes made by T_1 between steps t_N and t_S along the vertical segment of positions from z_N to z_S. Let ρ denote the infinite path that extends $p_1, \ldots, p_m$ infinitely to the north from p_1 and infinitely to the south from p_m.

We claim that there is a bound C_0 depending only on T_1, b and k, but not on n and neither on u, such that the run of T_1 starting from global state s_1 until time step kt_N crosses at most C_0 times path ρ. First, by choice of $p_1, \cdots, p_m$, such crossings can only happen at positions of ρ that are either at the north of p_1 or at the south of p_m. The simulation is liberal, so the head of T_1 has some freedom of move but it must always remain at a bounded distance from the block corresponding to the simulated head position of T_2 during intermediate steps, precisely: if the head of T_2 is at position z at time step t, then the head of T_1 must be inside block $b \otimes z$ at time step kt and therefore at distance at most k of block $b \otimes z$ during time steps between kt and $k(t+1)$. A position is therefore potentially reachable by T_1 before time step kt_N only if it is at distance at most k from a block $b \otimes z$ such that position z in the run of T_2 is visited before time step t_N. The key observation is that in the run of T_2, there are only finitely many positions that are visited before time t_N and at distance less than k from either z_N or any position at the north of it, or from z_S or any position at the south of it. From this we deduce that $\rho \setminus \{p_1, \ldots, p_m\}$ is crossed a bounded number of times C_0 by the head of T_1 before time step kt_N. The claim that ρ is crossed at most C_0 times before time step kt_N follows since $\{p_1, \ldots, p_m\}$ are by definition empty before this time step.

Finally, note that path $p_1, \dots, p_m$ cannot move away more than distance k from blocks $b \otimes z_N$ to $b \otimes z_S$, so if n is large enough, the domain of s_2 is guaranteed to lie entirely inside the left connected component A_0 of $\mathbb{Z}^2 \setminus \rho$. Similarly, the blocks containing the encoding of the rightmost copy of u must lie entirely inside $\mathbb{Z}^2 \setminus A_0$. In particular, the configuration c of T_1 reached at step kt_N must be such that $K(c_{|\mathbb{Z}^2 \setminus A_0}) \in \Omega(n)$ by choice of u. However, Lemma 8 applied at step kt_N to T_1 and ρ gives: $K(c_{|\mathbb{Z}^2 \setminus A_0}) \leq C \log(kt_S) \in O(\log(n))$ which is a contradiction for large enough n. ◀

4 Universality Results

4.1 Radius 3 in 2D under Rigorous Simulations: the Heat Sink Trick

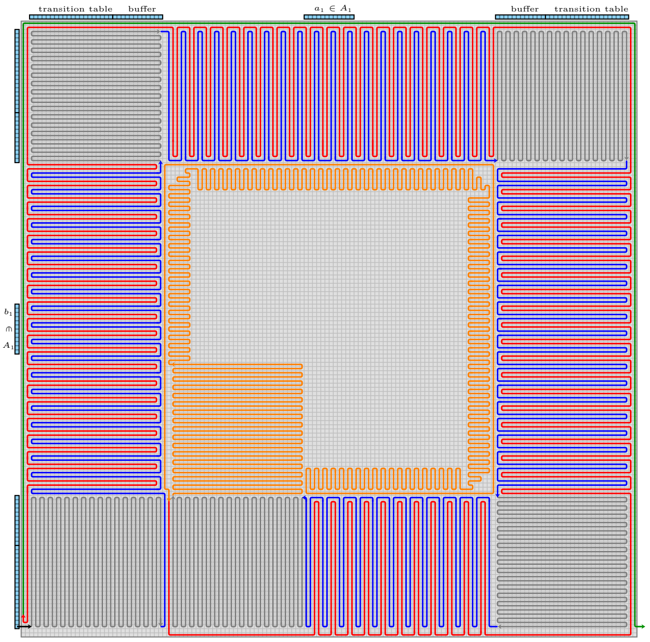


Figure 6 Behavior, in one block, of the presented 2D radius 3 turedo $T_3 \in \mathcal{U}_2^{\leq R}(1)$. The light blue rectangles represent the information to read of the neighboring cells. In this example, a letter of A_1 is encoded by 3 letters of A_3. We can see the 2 padding cells, the transition table (here of arbitrary small size for readability of the figure), the buffer of size 12 (to contain 4 letters of A_1 encoded), the padding, the letter of the block encoded by 3 letters of A_3 repeated 4 times each and again the the padding, the buffer, the table and the last 2 padding cells. The arrows represent the path followed by the turedo, entering the block in the bottom left of the figure. The blue part reads the content of the adjacent blocks, the grey one allows for turning, the orange one is where the computing takes place, the red one fetches the computed letters (and writes them on the faces that will not be visited again before exiting the block) and the green one finishes to write the letters and exits to the next block.

Theorem 6 shows that no turedo of radius 1 can be universal for $\mathrm{TUR}_2(1)$ under rigorous simulations. We show that this is however possible with radius 3. In order to achieve universality, four key behaviors must be performed by our turedo at each simulation step. It needs to read all necessary information of neighboring blocks, compute the next simulation step, write the computed letter and exit the current block (entering the correct next one). Figure 6 illustrate those behaviors. To rigorously simulate a turedo of radius 1, it seems natural to perform the 3 behaviors interacting with neighbors on the edge of the block, each on its layer, motivating a radius 3. But as we are dealing with universality, the so called necessary information is not only the letters of neighboring blocks but also the transition table defining the simulated turedo. Carrying this information has a direct impact on the width of the reading and writing layer, we propose an intertwined zigzag to merge the space occupied by those two, keeping the radius 3 and taking full advantage of it.

▶ **Theorem 10.** $\mathcal{U}_2^{\leq_R}(1) \cap TUR_2(3) \neq \emptyset$.

Proof. We show that there is $T_3 \in \mathrm{TUR}_2(3)$ such that for all $T_1 \in \mathrm{TUR}_2(1)$: $T_1 \leq_R T_3$. Denote $T_3 = (A_3, Q_3, \delta_3)$. Let's take $T_1 \in \mathrm{TUR}_2(1)$ some 2D turedo, $T_1 = (A_1, Q_1, \delta_1)$, a configuration $c_1 \in A_1^{\mathbb{Z}^2}$ and describe how T_3 simulates it with square blocks $R_{(n,n)}$ and $n = 0 \bmod 4$. We first focus on the organization of transmittable information in a given block B, *i.e.* the transition table δ_1 and the letter $a_1 \in A_1$ in this position in c_1. To be accessible to the neighboring blocks, this information is present on the outside edges of B, repeated on each edge such that $B(0,i) = B(i, n-1) = B(n-1, n-1-i) = B(n-1-i, 0)$. Considering a partial onto letter decoding map $\gamma : A_3^m \to A_1$ with $m \in \mathbb{N}$, the organization on one edge of B is the following. The first two positions $B(0,0)$ and $B(0,1)$ are empty or irrelevant, then the next $3m|A_1|$ positions from $B(0,2)$ to $B(0, 3m|A_1|+1)$ are the encoding of the transition table with γ (which we assume to be a multiple of 4 without loss of generality). Positions $B(0, 3m|A_1|+2)$ to $B(0, m(3|A_1|+4)+1)$ are reserved for a buffer in which 4 letters will be encoded (the ones contained in the 4 neighboring blocks). Then $m(3|A_1|+4)+2$ positions are empty or irrelevant, from $B(0, m(3|A_1|+4)+2)$ to $B(0, 2m(3|A_1|+4)+3)$, to allow the block to be spacious enough for the computation. Following that is written $u \in A_3^{4m}$ such that $u(4i+k) = \gamma(a_1)(i)$ for all $0 \leq k < 4$ (the redundancy is present to ensure proper reading later). Then again, from $B(0, 2m(3|A_1|+6)+4)$ to $B(0, 3m(3|A_1|+4)+2m+5)$ is some irrelevant padding followed by the $4m$ sized buffer, the $3m|A_1|$ sized transition table and 2 irrelevant position, finishing at $B(0, m(12|A_1|+20)+7)$. An illustration of this distribution is represented in figure 6.

Let's now describe the behavior of T_3 in a block to achieve universality. As all necessary information to compute is contained in a $m(3|A_1|+4)$ letters long word on A_3, we base our construction on two types of zigzags of this size : the square and the heat sink. A square gadget of even size k is a back and forth $k/2$ times of a k sized line during which the content of the original line is replicated on each four sides of the created square. This copy is possible thanks to the radius of T_3 being greater than 2. Its main use is to keep and spread information while cornering. A heat sink gadget is a zigzag with each back and forth being spaced by 2. The radius 3 of T_3 allows the heat sink to still copy information from the previous zag during a zig. Its purpose is to acquire and transmit data laterally while being intertwined with another heat sink. Both gadget and the simulation of a block are illustrated in figure 6 to illustrate the following explanation.

T_3 enters a block B at position $B(0,2)$ (or $B(2, n-1)$, $B(n, n-3)$, $B(n-3, 0)$ up to rotation), it first continues forward by 2 (until reaching $B(2,2)$) then turns 90° counter-clockwise and starts a square gadget, initialized by the copy of the transition table and empty buffer available at distance 3. Those squares will be performed at each corners of the block with in between a heat sink which will both transmit the transition table already collected earlier and approach the outside edge of the neighboring block at distance 3, reading its information and filling the buffer. (The heat sink has access to only half of this information but it has been taken care of by the redundancy detailed earlier). Once the information of all four neighboring blocks collected, right after the last letter has been read, T_3 turns toward the center of the block, using the space left by the padding, performing one more square gadget. With a big enough transition table and a well chosen encoding, computing the next transition of T_1 can be performed in a square the size of the buffer and transition table encoded. As the only way to leave the center of the block is now a path of width 1, T_3 must transmit its computation before rejoining the edge of the block. It does so with a zigzag that follows along the the inside of the reading heat sink. This information is then retrieved by the writing process, with a heat sink intertwined with the reading one. The writing process writes on the edge of the block on the faces before the exit one and at distance 1 after. When following a square gadget, it copies the transition table and writes an empty buffer (which is possible because the square not only corners but copies information on all its sides), when following a reading gadget, it creates a heat sink gadget of its own, fetching the computed information and writing it on the face of the block. Lastly, once all the writing has been done, T_3 finishes filling its block by going back to the exit edge, following the writing path which had been shifted to the inside by 1 to allow for this, copying everything from the reading path to effectively write it on the edge making it attainable and finally T_3 exits the block at distance 2 from the corner. ◀

4.2 The Power of Third Dimension and Liberal Simulations

Having a third dimension available allows for a lot more freedom to simulate a turedo. Let us first focus on $\mathrm{TUR}_3(1)$ and rigorous simulations. In dimension 2 we had to use the heat sink trick to gather all required information and we used a radius of 3 to accommodate for a reading, a writing and an exiting layer. The third dimension allows us to shrink the radius to 1 thanks to its crossing capabilities. The trick to achieve this is to have a marker to indicate if the neighboring block is empty or not, to prevent trying to read in spaces where the turedo will have to write later on.

▶ **Theorem 11.** $\mathcal{U}_3^{\leq_R}(1) \cap TUR_3(1) \neq \emptyset$.

Proof. We show that there is $T \in \mathrm{TUR}_3(1)$, $T = (A, Q, \delta)$, such that for all $T' \in \mathrm{TUR}_3(1)$: $T' \leq_R T$. Denote $T' = (A', Q', \delta')$. For a block B, the transmittable information is organized on the faces as follow. In the center of each faces, a marker is written, indicating that information is present. On the face of the block facing the next block (the exit face), added to the presence marker, is an empty cross of width 1 with only at its end a stop marker indicating the end the block. Surrounding it is a cross of width 3 in which is written the transition table and a buffer big enough to accommodate for the encoding of 6 letters of A' and one state from Q' (with the size of the transition table and the buffer plus 1 left before writing). On the face of the block facing other empty blocks, the same crossing pattern is used to only write the presence marker and the computed letter at distance 2 of the center of the face. See figure 7 for this organization on an example.

T has the following behavior in a block, also illustrated in figure 7. First, T performs a reading phase : entering a block by the middle of a face, it travels straight to the edge of the face until reading the stop marker. We assume T entering by the middle of the face as it truly enters at distance one of the middle but this shift of one position in one direction can be remembered and corrected immediately after entering. This allows T to turn and go back toward the center, reading the transition table and the buffer. All this information gathered, T follows a fixed path composed of shrinking zigzags, staying in the planes formed by the 3D cross centered in the block (hence the peculiar way we place information, to reserve space for this path). Doing so, it reaches the center of each face while carrying the transition table and filling the transition buffer when a presence marker is read. As the order in which the faces are visited is fixed, T can store in its head state which neighboring blocks are empty. This

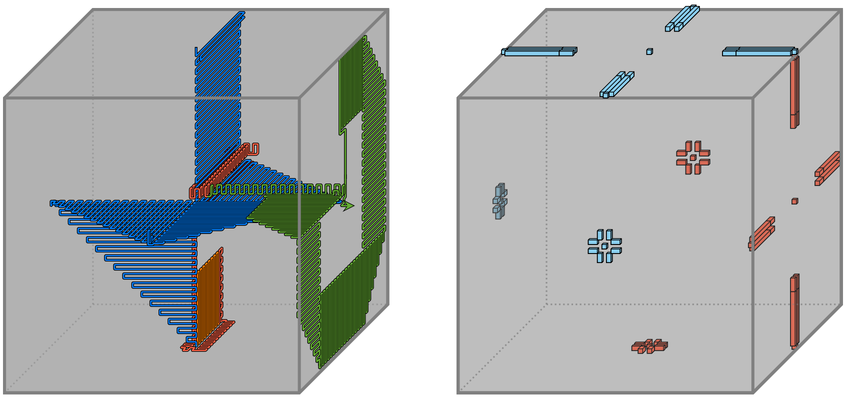


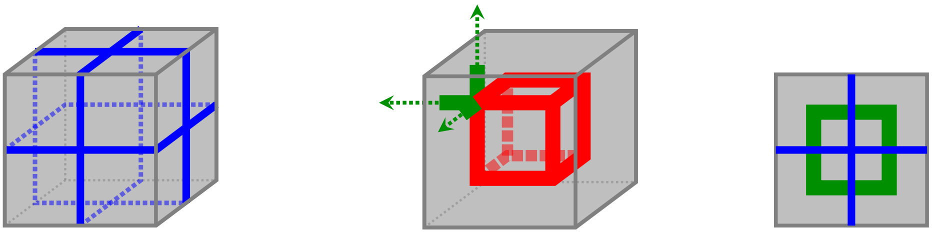
Figure 7 One block of the presented 3D radius 1 turedo $T \in \mathcal{U}_3^{\leq R}(1)$. At this stage T comes from the block above, the front and left blocks are also non-empty and the others are empty. T will exit the block by entering the next block at its right. On the right is represented the disposition of information. The light blue cubes are information to read in adjacent blocks and the red ones are information written by T in the considered block. In this example, a letter of A' is encoded by 2 letters of A, the transition table is of arbitrary small size for readability and they are written in a cross on the faces of blocks. On the left is the behavior of T in the same considered block, in blue is the reading phase (building also a skeleton), in orange the computation phase, in red the writing phase and in green the exit phase.

reading phase has the added benefit to have created a skeleton in the block, enabling easy travel for the following phases. Next T enters the computing phase consisting of a square zigzag, using the transition table and the now filled buffer carried previously and so present in all the branches of the centered 3D cross. Once the next step of the simulation computed, T can carry the letter to encode following the skeleton and write it on all previously identified faces except the exit one. The last phase, the exit one, consists in carrying the encoded letter to the exit face, retrieve on the centered 3D cross the transition table and the buffer (reset with the new state and letter of the computed transition) and writing all this information as previously presented thanks to square and triangular zigzags. Finally T exits to the next block, at distance one of the center of the face (to get around the reading skeleton). ◀

The combination of 3D and liberal simulations allows to shrink the radius of any turedo to 1. In this construction, the computation of simulated transitions is done internally in the simulating turedo's head. The challenging part however is in acquiring the states of distant neighbors. Thankfully the liberal nature of the simulation allows travel through empty blocks and the 3D enables crossing paths without intersection. Still, a rigorous organization is needed in order to prevent overlapping.

▶ **Theorem 12.** *For any radius r and any $T_r \in TUR_3(r)$ there is $T_1 \in TUR_3(1)$ such that $T_r \leq T_1$.*

Proof. Let $r \in \mathbb{N}_+$ and $T_r \in \mathrm{TUR}_3(r)$, $T_r = (A_r, Q_r, \delta_r)$. We build $T_1 = (A_1, Q_1, \delta_1)$, with Q_1 big enough to encode in one state a position $z \in \mathbb{Z}^3$ modulo $2r$ of each dimension and the $|B_3(r)|$ neighboring letters. This allows for an instant computation of δ_r once all needed information is gathered. Let b be the block size, the critical aspect of this simulation is for T_1 to visit all blocks $b \otimes z'$ with $z' \in z + B_3(r)$ for each simulation step. Therefore we have to assign non intersecting exploration paths for all positions at distance less than $2r$. To achieve this, we define $\mathcal{C} = \{0, ..., 8r^3 - 1\}$ a set of colors and we assign the color $(z_1 \bmod 2r) + 2r(z_2 \bmod 2r) + 4r^2(z_3 \bmod 2r)$ to the block $b \otimes (z_1, z_2, z_3)$. By taking $b = 8r^3l + 7$ with $l \in \mathbb{N}_+$, in each block, for each color $c \in \mathcal{C}$, we can reserve tubes of width l following the edges of a centered cube of edges of length $lc + 1$ and the direct extension of said edges to the face of the block (see figure 8b). This creates reserved spaces for each color consisting of centered nested cubes. Those cubes fill a space of $8r^3l$, we add 1 to have a proper center and 6 to have some padding near the faces of the blocks (we will discuss its necessity later). Note that l actually doesn't need to be large, $l = 10$ is enough. On each face of the block $b \otimes z$, the letter $a \in A_r \subset A_1$ is repeated in a cross pattern (see 8a).



(a) The letter $a \in A_r \subset A_1$ is repeated on all faces in a cross pattern.

(b) Reserved space for the exploration path of color $c \in \mathcal{C}$.

(c) One face of a block.

Figure 8 Representation of the reserved space for color c in block $b \otimes z$. In red are the tubes of width l on the edges of the centered cube of edges of length $lc + 1$. In green are the extensions of said tubes, allowing to reach the reserved space for color c in the neighboring blocks, completing the exploration path of color c (8b). Those extensions also allow the exploration path to access the blue cross containing the letter of position z in the configuration (to either read or write)(8c).

$F_{T_r}(c_r, z_r, q_r)$ is simulated as follow. Assuming T_1 knows the color $c \in \mathcal{C}$ of block $b \otimes z_r$ (which is possible as its position in all directions modulo $2r$ is stored in its head state), we can define a reference starting position to explore the neighboring blocks by ordering the directions of $B_3(1)$. Moreover, this ordering allows to decide a depth-first exploration of the blocks $b \otimes z'$ with $z' \in B_3(r)$ passing through each block at most seven times. Once the exploration done, back in block $b \otimes z$, the head of T_1 contains all necessary information to compute δ_r and all that remains to do is to write the computed letter in a cross pattern on the faces of the block. This is possible following a eulerian path, crossing only on the center of the faces, hence the padding of 2 defined earlier. The 1 padding left is for T_1 to align itself with its next color (which is possible since it knows its current color and has computed to in which block to go next). ◀

By combining Theorem 11 and Theorem 12, we get the existence of an intrinsically universal 3D turedo for liberal simulations as expressed in the following corollary.

▶ **Corollary 13.** $\mathcal{U}_3^{\leq} \cap TUR_3 \neq \emptyset$.

5 Discussion

The problem tackled in this work depends on three parameters (radius, dimension, simulation). Our results give a rather clear picture of the simulation hierarchies in 3D, but we left several open question in the 2D case, in particular: is there a turedo in $\mathrm{TUR}_2(2)$ which is universal for $\mathrm{TUR}_2(1)$ under rigorous simulations? what if we allow liberal simulations? Actually even Theorem 6 raises questions: the simulation impossibility makes a crucial use of non-connected seeds, does this impossibility remain if we just look at simulations of orbits starting from connected seeds?

In this work we chose the square lattice in 2D (to simplify and as we also consider the 3D case) whereas oritatami are mostly considered on the hexagonal lattice. We don't expect any significant difference on the simulation hierarchy result by changing from square to hexagonal lattice on a given model (either turedos or oritatami). However, it is not clear that the delay hierarchy for oritatami behaves like the radius hierarchy for turedos. In particular, we don't know if an analog of Theorem 10 holds for oritatami. The key difference between a large radius turedo and a large delay oritatami is that the turedo can gather information locally across obstacles, while the oritatami can only probe information around that can be reached by a path of empty positions (because it can only probe by trying to position a small strand of beads).

We end this paper by suggesting the following two directions in order to better understand the gap between turedos and oritatami systems: what if we restrict turedos to 'see' only neighboring positions that can be reached through a path of r empty positions? and what if we enrich oritatami systems by a more general 'magnetic' attraction law between beads where pairs of distant beads can still contribute to the total amount of attraction that the free strand at the end of the molecule is trying to maximize (let's say by a quadratic decrease with distance up to some radius)?

References

1 Florent Becker, Diego Maldonado, Nicolas Ollinger, and Guillaume Theyssier. Universality in freezing cellular automata. In *Sailing Routes in the World of Computation - 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30 - August 3, 2018, Proceedings*, pages 50–59, 2018. doi:10.1007/978-3-319-94418-0_5.

2 Laurent Boyer and Guillaume Theyssier. On factor universality in symbolic spaces. In *Mathematical Foundations of Computer Science 2010*, pages 209–220. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-15155-2_20.

3 Matthew Cook, Yunhui Fu, and Robert T. Schweller. Temperature 1 self-assembly: Deterministic assembly in 3D and probabilistic assembly in 2D. In *SODA2011: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete*, pages 570–589, 2011.

4 Marianne Delorme, Jacques Mazoyer, Nicolas Ollinger, and Guillaume Theyssier. Bulking ii: Classifications of cellular automata. *Theor. Comput. Sci.*, 412(30):3881–3905, 2011. doi:10.1016/j.tcs.2011.02.024.

5 David Doty, Jack H. Lutz, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *FOCS2012: Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, pages 302–310, 2012.

6 Pierre Étienne Meunier and Damien Woods. The non-cooperative tile assembly model is not intrinsically universal or capable of bounded Turing machine simulation. In *STOC 2017: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 328–341, 2017.

7 Cody Geary, Guido Grossi, Ewan K. S. McRae, Paul W. K. Rothemund, and Ebbe S. Andersen. RNA origami design tools enable cotranscriptional folding of kilobase-sized nanoscaffolds. *Nature Chemistry*, 13:549–558, 2021.

8 Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki. Programming biomolecules that fold greedily during transcription. In *MFCS2016: Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science*, volume 58 of *LIPIcs*, pages 43:1–43:14, 2016.

9 Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki. Oritatami: A computational model for molecular co-transcriptional folding. *International Journal of Molecular Sciences*, 9(2259), 2019. doi:10.3390/ijms20092259.

10 Cody Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinonsuke Seki. Proving the Turing universality of oritatami cotranscriptional folding. In *ISAAC 2018: Proceedings of the 29th International Symposium on Algorithms and Computation*, volume 123 of *LIPIcs*, pages 23:1–23:13, 2018.

11 Cody Geary, Paul W. K. Rothemund, and Ebbe S. Andersen. A single-stranded architecture for cotranscriptional folding of RNA nanostructures. *Science*, 345:799–804, 2014.

12 Cody W. Geary, Pierre-Étienne Meunier, Nicolas Schabanel, and Shinnosuke Seki. Proving the turing universality of oritatami co-transcriptional folding. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, volume 123 of *LIPIcs*, pages 23:1–23:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ISAAC.2018.23.

13 Jacob Hendricks, Matthew J. Patitz, and Trent A. Rogers. Universal simulation of directed systems in the abstract tile assembly model requires undirectedness. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, October 2016. doi:10.1109/focs.2016.90.

14 James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers. Computability and complexity in self-assembly. *Theory Comput. Syst.*, 48(3):617–647, 2011. doi:10.1007/s00224-010-9252-0.

15 Ming Li and Paul Vitányi. Algorithmic complexity. In *Texts in Computer Science*, pages 101–195. Springer New York, 2008. doi:10.1007/978-0-387-49820-1_2.

16 Pierre-Etienne Meunier, Matthew J. Patitz, Scott M. Summers, Guillaume Theyssier, Andrew Winslow, and Damien Woods. Intrinsic universality in tile self-assembly requires cooperation. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 752–771. SIAM, 2014. doi:10.1137/1.9781611973402.56.

17 Daria Pchelina, Nicolas Schabanel, Shinnosuke Seki, and Guillaume Theyssier. Oritatami Systems Assemble Shapes No Less Complex Than Tile Assembly Model (ATAM). In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 51:1–51:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2022.51.

18 Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2:2041–2053, 2004.

19 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

A Proof of Theorem 5

Proof. We prove the result for $d = 2$, the argument can be generalized to higher dimension straightforwardly (by completing 2D configurations by $\perp$ everywhere else).

For the first item, simply consider a turedo $T_{r+1} \in \mathrm{TUR}_2(r+1)$ that has the following behavior when the head in position $(0,0)$ has only $\perp$ letters at the north and at the south of its current position: read the letters at positions $(r+1,0)$ and $(0,r+1)$ and move to the north is they are equal and to the south otherwise. Consider any turedo $T_r \in \mathrm{TUR}_2(r)$ and any block size $b \in \mathbb{N}_+^2$. To simulate T_{r+1} fuzzlessly, T_r has to move either to block $b \otimes (0,1)$ or block $b \otimes (0,-1)$ depending on blocks $b \otimes (r+1,0)$ and $b \otimes (0,r+1)$ without entering into any other neighboring block: this is impossible, because with radius r turedo T_r can't have any information about either $b \otimes (r+1,0)$ or $b \otimes (0,r+1)$ before making a decisive move (by entering inside either $b \otimes (0,1)$ or $b \otimes (0,-1)$) so it will fail to correctly simulate the orbit of at least one seed.

The second item can be proved similarly, using a pumping trick on the alphabet: for any fixed $T_r \in \mathrm{TUR}_2(r)$ with alphabet of cardinal k, choose $T_2 \in \mathrm{TUR}_2(2)$ with an alphabet strictly larger than k^{r^2} so that at least one dimension of the block size b of any potential simulation of T_2 by T_r has to be at least $r+1$ (otherwise there is simply no way to code all letters of T_2 on different blocks of size b). Then, choosing T_2 to have the same behavior as T_{r+1} above, we get the same contradiction: there is a direction of b, let's say the vertical one, which overwhelms the radius r of T_r so T_r won't be able to read the content of block $b \otimes (0,2)$ before making a decisive move and will therefore fail to correctly simulate at least one orbit. ◀

B Proof of Lemma 8

Proof. We show that $c_{|\mathbb{Z}^2\setminus A_0}$ can be computed from the following description $\mathcal{D}$:

- the finite list of positions on ρ that are visited by the head during the n first steps of the run starting from s;
- the list of events corresponding to each such position z given as a triple: time at which the head leaves position z, letter written at that step, and move made by the head at that step.

This description is of size $O(\log(n))$ because both positions z and time steps occurring in the above lists are bounded by n by definition (recall that the head is initially at $(0,0)$).

Because T is of radius 1 and ρ is 4-connected, each time the head of the turedo is neither in A_0 nor on ρ, the local transition does not depend on the current configuration on domain A_0. A Turing machine can therefore compute $c_{|\mathbb{Z}^2\setminus A_0}$ from this description by maintaining the following partial information step by step:

- the current configuration restricted to domain $\mathbb{Z}^2 \setminus A_0$,
- the partial information on the head position z: the exact position if $z \notin A_0$ or the state "undefined" else.

This information is straightforward at the initial step since $\mathcal{D}(s) \subseteq A_0$ so the head is in A_0 and the configuration is $\perp$ everywhere outside A_0. The partial information at step n is enough to give $c_{|\mathbb{Z}^2\setminus A_0}$ and it is updated from one step i to the next $i+$ as follows:

- if the partial information on the head at step i is undefined and time step $i+2$ does not appear in the lists of $\mathcal{D}$, then don't change the partial information (the head is in A_0 and won't move to ρ at step $i+1$);
- if the head information is undefined but step $i+2$ appears in $\mathcal{D}$, then updates the head position to the position on ρ that corresponds to the item stamped by time steps $i+2$ in $\mathcal{D}$;
- if the head position is on ρ then some item in the list of $\mathcal{D}$ must be stamped by time step $i+1$ and gives all the information to update both the head position and the configuration on ρ;
- finally if the head position is neither in A_0 nor on ρ, then the knowledge of the current configuration restricted to domain $\mathbb{Z}^2 \setminus A_0$ is enough to update the partial information (position and partial configuration). ◀

Computing Real Numbers with Large-Population Protocols Having a Continuum of Equilibria

Xiang Huang[1]
Department of Computer Science, University of Illinois Springfield, USA

Rachel N. Huls
Department of Mathematics, University of Illinois Springfield, USA

Abstract

Bournez, Fraigniaud, and Koegler [6] defined a number in [0,1] as computable by their Large-Population Protocol (LPP) model, if the proportion of agents in a set of marked states converges to said number over time as the population grows to infinity. The notion, however, restricts the ordinary differential equations (ODEs) associated with an LPP to have only finitely many equilibria. This restriction places an intrinsic limitation on the model. As a result, a number is computable by an LPP if and only if it is *algebraic*, namely, not a single transcendental number can be computed under this notion.

In this paper, we *lift* the finitary requirement on equilibria. That is, we consider systems with a continuum of equilibria. We show that essentially all numbers in [0,1] that are computable by bounded general-purpose analog computers (GPACs) or chemical reaction networks (CRNs) can also be computed by LPPs under this new definition. This implies a rich series of numbers (e.g., the reciprocal of Euler's constant, $\pi/4$, Euler's γ, Catalan's constant, and Dottie number) are all computable by LPPs. Our proof is constructive: We develop an algorithm that transfers bounded GPACs/CRNs into LPPs. Our algorithm also fixes a gap in Bournez et al.'s construction of LPPs designed to compute any arbitrary algebraic number in [0,1].

2012 ACM Subject Classification Theory of computation → Interactive computation; Theory of computation → Probabilistic computation; Theory of computation → Abstract machines

Keywords and phrases Population protocols, Chemical reaction networks, Analog computation

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.7

Funding *Xiang Huang*: This research was supported in part by NSF grants 1545028 and 1900716.
Rachel N. Huls: This research was supported in part by the University of Illinois Springfield Leadership Lived Experience (LLE) initiative.

Acknowledgements We thank Titus Klinge for providing the motivating example in Section 1.

1 Introduction

1.1 GPAC/CRN Computable Number

The computation of real numbers provides a theoretical benchmark for an abstract model's computational power. In 1936, Turing [32] considered computable numbers while researching a discrete model now known as the Turing machine. Then, Shannon [30] introduced the general-purpose analog computer (GPAC) to the world. The GPAC is a mathematical abstraction of Bush's differential analyzer. Bush [8] developed this machine to obtain numerical solutions of ordinary differential equations (ODEs). After much revision [29, 27, 18, 17], we can now characterize GPACs by polynomial initial value problems (PIVPs):

[1] corresponding author

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham; Article No. 7; pp. 7:1–7:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\begin{cases}\mathbf{x}' = \mathbf{p}\big(\mathbf{x}(t)\big) \\ \mathbf{x}(0) = \mathbf{x}_0,\end{cases} \qquad t \in \mathbb{R} \tag{1}$$

where $\mathbf{p}$ is a vector of multivariate polynomials, $\mathbf{x}$ is a vector of variables, and $\mathbf{x}_0$ is its initial value. Bournez et al. [7] have shown that GPACs are provably equivalent to Turing machines in terms of computability and complexity. Chemical reaction networks (CRNs), as a restricted form of GPACs, are widely used in molecular programming applications. The deterministic semantic of CRNs usually assumes well-mixed solutions and mass-action kinetics. CRN dynamics can be modeled by polynomial systems with extra constraints [21] (see the preliminary section for more details). Although the constraints could seemingly weaken the computational power of CRNs, they are able to simulate GPACs. Hence, the CRN model is also Turing complete [13]. The key idea is to encode every variable x (possibly having negative values) in a GPAC by the difference between a pair of *positive* variables x^+ and x^- in the copycat CRN. That is, $x(t) = x^+(t) - x^-(t)$ for all $t \geq 0$.

A straightforward idea to "compute" a number α by a GPAC/CRN is to designate a variable x such that $x(t) \to \alpha$ as $t \to \infty$. Further down this line, Huang et al. defined the class of real-time computable real numbers by chemical reaction networks [24], wherein the real-time computability notion essentially requires $x(t)$ to converge exponentially fast to α. Huang et al. then demonstrate that GPACs and CRNs compute the same class of real numbers in real-time [23]. Notably, famous transcendental numbers like e and π are among this class. Later, Huang [22] added Euler's γ and Dottie number (the unique real root of the equation $\cos(x) = x$) to the list. Note that in order to get a meaningful measure of time, in Huang et al.'s work, all variables in ODEs associated with GPACs or CRNs are *bounded*. Hence, the notion of time aligns with the time parameter t of the ODEs since no more than a linear speedup is allowed. This notion prevents the so-called Zeno phenomena [19] caused by inordinate speedup. Another, perhaps equivalent, metric is the *arc length* of $\mathbf{x}(t)$. A *complexity theory* [7] of analog computation can then be built upon this metric.

Also, note that the notion Huang et al. used has a *descriptive* nature: the initial values are all zero, and rate constants are integrals (rationals). Hence, one can only encode a finite amount of information in the initial value and rate constant. In this sense, a system that computes a number α also gives a finite description of α. This idea will prove crucial for later discussions.

1.2 Large Population Protocols

Population protocols (PPs) [2] can be treated as special CRNs with two inputs, two outputs, and a unit rate constant per reaction. Observe that population protocols *conserve* the population since interactions do not change agent counts. This property imposes a vital constraint on the population: the sum of all agents remains constant. After normalization, we can say that this sum is *one* without loss of generality.

Classical PPs are concerned with computing predicates over their input configuration in a discrete, stochastic setting. Therefore, computing real numbers does not make sense under this setting. In a series of papers [3, 5, 6], Bournez et al. developed a setting they termed a Large-Population Protocol (LPP). LPPs are a new analytical setting rather than a new computational model. They are still population protocols, but their behaviors are analyzed as the population grows to infinity. Bournez et al. aimed for asymptotic results independent of the initial configuration. Our approach departs from theirs in that dependence on initial values is considered. Note that we should approach with caution when defining what is meant by "initial configuration" since LPPs need to talk about different population sizes along the way.

The LPP setting provides a playground for computing real numbers by population protocols: a number $\nu \in [0,1]$ is said to be computable by an LPP if the proportion of agents in a set of *marked states* converges to ν over time as the population grows to infinity. An additional requirement is that the ODE induced by the *balance equation* of the LPP must have finitely many equilibria. This condition enforces (exponential) stability and detaches the dependence on initial values.

Bournez et al. specifically asked[6] "... is it possible to compute solutions of trigonometric equations? E.g., can we design a protocol insuring that, asymptotically, a ratio $\frac{\pi}{4}$ of the nodes are in some prescribed state?" Their answer is *negative*. They proved that, under their definition, LPPs compute exactly the algebraic numbers in [0,1]. Closely related is a result by Fletcher et al. [15], where a class of numbers, called Lyapunov numbers, are found to be the set of algebraic numbers. It is not a coincidence: both works involve ODEs with either finitely many or isolated equilibria. Hence, Tarski's quantifier elimination over real closed fields can be performed. Alternatively, one can find more modern elimination algorithms involving Gröbner bases in standard algebraic geometry texts [10, 31].

Why is there a gap between LPPs (cannot compute transcendentals) and GPAC/CRN (can compute e and π)? The following example helps explore the question.

▶ **Motivating Example.** *Let $F(t) = \frac{1}{2}e^{e^{-t}-1}$, $E(t) = \frac{1}{2}e^{-t}$, and $G(t)$ be a function such that its derivative "cancels" with E and F's derivative. We have the first-order system and the corresponding chemical reaction network:*

$$\textit{ODE:} \quad \begin{cases} F' = -2FE \\ E' = -E \\ G' = 2FE + E \end{cases} \qquad \textit{CRN/PP:} \quad \begin{cases} F + E \xrightarrow{2} G + E \\ E \to G. \end{cases}$$

It is evident that with initial values $F(0) = \frac{1}{2}$, $E(0) = \frac{1}{2}$, and $G(0) = 0$, the solution of $F(t) \to \frac{1}{2}e^{-1}$ as $t \to \infty$. Correspondingly, in the LPP setting, if we let $F(0) = \lfloor \frac{N}{2} \rfloor$, $E(0) = \lfloor \frac{N}{2} \rfloor$, and $G(0) = 0$ for a population of size N, we can observe from the experimental results in Figure 1 that the proportion of F gets very close to $\frac{1}{2}e^{-1}$ when N becomes larger. In fact, the so-called Kurtz's Theorem [26], which says ODE solution $F(t)$ equals the proportion random variable $F^{(N)}(t)/N$ almost surely when N goes to infinity, guarantees the limit is $\frac{1}{2}e^{-1}$. Although the CRN in the above example is not technically a PP, we can translate it into a probabilistic PP (Construction 1, see also [11], Theorem 1). We maintain the protocol in the current form for simplicity. ⌟

So the above LPP can "compute" $\frac{1}{2}e^{-1}$, a *transcendental* number! Is it a counter-example to Bournez et al.'s algebraic result? Not really. Since the system of ODEs in the example has the whole F-G plane (with $E = 0$) as equilibria, the system has a *continuum* of equilibria. Note that the key reason GPACs/CRNs in [23] can compute transcendentals is that they are systems with a continuum of equilibria.

1.3 Computing with a Continuum of Equilibria

The Motivating Example invites us to *extend* the notion of LPP-computable numbers: We can drop the finitary requirement on equilibria and therefore include systems with a continuum of equilibria. The seminal work [4] by Sanjay and Bernstein provides an in-depth study of this type of system.

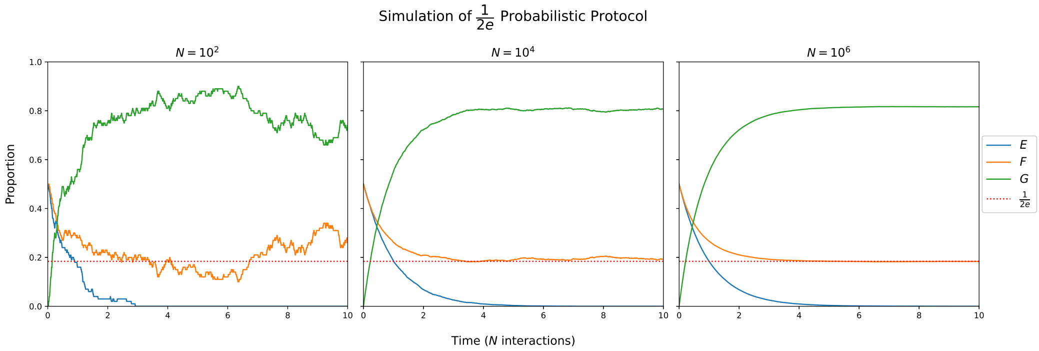


Figure 1 We use the ppsim Python package [11] to simulate our motivating example (transformed into a probabilistic PP). This figure demonstrates that the transcendental number $\frac{1}{2}e^{-1}$ can be traced by the proportion of a species as the population size grows.

The move has several consequences. An immediate one is that computation now *depends* on initial values: since the equilibria "stick together", we need the initial values to determine which equilibrium the system converges to. Another one is stability: we lose asymptotic stability. As pointed out in [4], "... Since every neighborhood of a non-isolated equilibrium contains another equilibrium, a non-isolated equilibrium cannot be asymptotically stable. Thus asymptotic stability is not the appropriate notion of stability for systems having a continuum of equilibria." However, it is inaccurate to immediately conclude that such systems must be fragile and sensitive to perturbations. Indeed, Sanjay and Bernstein investigated *semistability*, a more appropriate notion of stability for these systems.

We now state our new finding under the extended notion of LPP-computable numbers.

▶ **Main Theorem.** *LPPs compute the same set of numbers in [0,1] as GPACs and CRNs.*

Our proof is constructive: We give an explicit algorithm to translate a bounded GPAC or CRN into an LPP. A critical step is the construction of a cubic form (homogeneously degree 3 polynomial) system that conserves the population. Then, we transform the system from the cubic form into the quadratic form. Finally, the system can be rewritten as a probabilistic population protocol.

The key feature of our construction is that we always guarantee the system is either *CRN-implementable* or *PP-implementable*. Bournez et al. constructed a system in [6] that claimed to compute any algebraic number. While their system is a GPAC, it is not a PP in general. As a by-product, our construction fixes this flaw.

Related work

Besides Bournez et al.'s work [6, 3, 5], Gupta, Nagda, and Devaraj [20] develop a framework for translating a subclass of differential equation systems into practical protocols for distributed systems (stochastic CRNs in most cases and, in some examples, population protocols). They also restated Kurtz's theorem and considered a large-population setting. The subclasses they considered are *complete* and *completely partitionable*, which means the derivatives of the system sum to zero, and any term (monomial) T occurs as a pair $\{+T, -T\}$ in the system, a very nice property to have. Doty and Severson [11] provided an algorithm to translate CRNs that consist solely of unimolecular ($X \to Y$) and bimolecular reactions ($X + Y \to Z + W$) to PPs. Those reactions correspond to an ODE system's linear terms (e.g., x) and quadratic terms (e.g., xy). Their assumption, however, restricts the application of their algorithm from more general GPACs. Firstly, the CRNs they considered preserve populations. *Constant terms* in GPACs , which correspond to reactions like $\emptyset \to X$, *change* the population. Another consideration is the treatment of square terms. Some quadratic systems can not be *directly* written as a CRN with bimolecular reactions (i.e. if positive square terms like x^2 occur in x'). Later in this paper, we will discuss how the deep entanglement of the two (constant terms and squared terms) makes translating a general GPAC into a PP very difficult.

Our construction does not make any assumptions in the above work on GPACs, except that all variables are bounded.

The rest of this paper is organized as follows. Section 2 discusses preliminaries as well as some notations and conventions necessary to present our main result; Section 3 includes the main theorem, a four-stage constructive proof, and a few immediate corollaries. Section 4 provides some concluding remarks and discussion on future directions.

2 Preliminaries

We review some key definitions and basic facts in this section.

2.1 GPACs, CRNs, Population Protocols, and Their ODE Characterization

We have discussed that GPACs can be characterized by polynomial initial value problems (PIVPs) in the previous section. To align with the ODE descriptions of GPACs, we will use the ODE characterization of CRNs and PPs throughout this paper. This choice clarifies the discussion of our construction and proof in the coming sections.

We start with a few notations and conventions in this paper.

▶ **Notation** (Variable Vector)**.** *We use the notation $\mathbf{x}(t) = (x_1(t), x_2(t), \cdots, x_n(t))$ to denote a vector of variables in (usually the solution of) an ODE system, where $n \in \mathbb{N}$ is the vector's dimension and the parameter t is regarded as "time" in the system. The first component x_1 is usually designated for tracing the number we wish to compute.*

A typical definition of a CRN or PP specifies a pair (S, R), with S being the set of species or states and R being the set of reactions or interactions between states. In general, a reaction for an abstract CRN looks like $X + Y \xrightarrow{k} X + W + Z$, where k is the *reaction constant*. There is no restriction on the number of reactants (input) or products (output). Reactions in population protocols, however, are strictly limited to those with two inputs and two outputs: $X + Y \to W + Z$.

The so-called *deterministic mass-action* semantic of a CRN defines a polynomial differential system, which governs the CRN's dynamics. One can find descriptions of this semantic in many works of literature (e.g., [23]). We will use these polynomial differential systems directly throughout the paper.

▶ **Convention.** *We manipulate our use of the variable vector: when the context is clear, we simply say $(\mathbf{x}, \mathbf{x}')$, or $\mathbf{x}$ itself is a GPAC (CRN, PP), without specifying the associated polynomial differential system $\mathbf{x}'(t) = (x_1'(t), \cdots, x_n'(t))$. If we don't specify the initial values that means we assume them to be zero. We would assume the correspondence between a variable x and the species X that it describes. In some cases, we will call a variable x a species and $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ a set of species.*

Now, we introduce notation and terminology for polynomials.

▶ **Notation** (Positive Polynomials). *We denote $\mathbb{P}^+ \subset \mathbb{Q}[\mathbf{x}] = \mathbb{Q}[x_1, \cdots, x_n]$ the set of multivariable polynomials with positive (rational) coefficients for each monomial (terms). For instance, $x_1x_2 + x_3 + 1 \in \mathbb{P}^+$, while $x_1x_2 - x_3 + 1 \notin \mathbb{P}^+$.*

▶ **Notation** (Variable Occurrence in a Polynomial or Monomial). *Let p be a polynomial (or monomial) and x be a variable, we use the convenient notation $x \in p$ to express x occurs in p.*

Homogeneous polynomials play a fundamental role in our construction. We call them forms for short.

▶ **Terminology** (Forms). *In mathematics, "form" is another name for homogeneous polynomials. A quadratic form is a homogeneous polynomial of degree two, for example, $4x^2+2xy-3y^2$; a cubic form is a homogeneous polynomial of degree three, for example, $x^3+3x^2y+3xy^2+y^3$.*

▶ **Terminology** (Conservative Systems). *A conservative system has a (non-trivial) quantity that is constant along each solution. In this paper, we call a system that maintains constant total mass, i.e., a system $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ that satisfies*

$$\sum_{i=1}^{n} x_i' = 0,$$

a conservative system.

We say a function $f(t)$ is implementable (generable) by a GPAC (CRN, LPP) if there exist a GPAC (CRN, LPP) and a variable (species, state) x in it, such that $f(t) = x(t)$ for all t. We choose the word *implementable* over *generable* (as in [18]) to reflect the idea that we *program* a GPAC (CRN, LPP) to *implement* a function.

▶ **Definition 1** (GPAC-implementable Functions [18]. See also Definition 3 in [13].). *A function $f : \mathbb{R}_{\geq 0} \to R$ is GPAC-implementable if it is the first component i.e. x_1, of the $\mathbf{x}(t) = (x_1, x_2, \cdots, x_n)$ solution for the following differential equation:*

$$\begin{cases} \mathbf{x}' = \mathbf{p}\big(\mathbf{x}(t)\big) \\ \mathbf{x}(0) = \mathbf{x}_0, \end{cases} \qquad t \in \mathbb{R} \tag{2}$$

where $\mathbf{p}$ is a vector of multivariate polynomials and $\mathbf{x}$ is a variable vector with $\mathbf{x}_0$ as its initial value.

Note that if a variable x has initial value a, we can introduce a new variable $y = x - a$ to make $y(0) = 0$. Therefore, without loss of generality, we often assume $\mathbf{x}(0) = \mathbf{0}$ in this paper.

▶ **Theorem 2** (CRN-implementable Functions [21], Theorem 3.1 and 3.2). *A function is CRN-implementable if and only if it is GPAC-implementable with further restriction such that the derivative of each component x_i in its variable vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ is of the form*

$$x_i' = p - qx_i, \quad \textit{where } p,\ q \in \mathbb{P}^+. \tag{3}$$

The restriction is rooted in reaction modeling: a negative term in x' means to *destroy* some molecular x at some rate. However, a reaction cannot destroy a non-reactant, so x must appear as a reactant in the reaction. Such restriction extends to Population Protocols together with some extra constraints.

▶ **Corollary 3** (PP-implementable Functions). *A function is PP-implementable, if and only if*

- **i.** *it is CRN-implementable, i.e., there is a CRN* $(\mathbf{x}, \mathbf{x}')$ *computing it.*
- **ii.** *for all* x_i *in* $\mathbf{x} = (x_1, x_2, \cdots, x_n)$*,* x'_i *does not possess a positive* x_i^2 *term.*
- **iii.** $\mathbf{x}'$ *is a quadratic form system, and*
- **iv.** $\mathbf{x}'$ *is conservative.*

Proof. For the “only if” direction, all conditions other than condition (ii) are clear. To see that x'_i can not have an x_i^2, we consider two X_i’s on the left-hand side of a reaction. The reaction can not increase the amount of X_i, since that would require at least $k+1$ X_i’s as products on the right-hand side. As a result, a positive x_i^2 term can not occur in x'_i. See Theorem 16 for the “if” direction of the proof. ◀

In this paper, we will also call a CRN containing only unimolecular reactions of the form $X \rightarrow Y$ a *unimolecular* population protocol (UPP). Similarly, a CRN containing only termolecular reactions of the form $X + Y + Z \rightarrow U + V + W$ is called a *termolecular* population protocol (TPP). In general, a k-PP is a CRN that contains only reactions of the form $X_1 + \cdots + X_k \rightarrow Y_1 + \cdots + Y_k$. Generally, we have the following characterization.

▶ **Corollary 4** (k-PP-implementable Functions). *A function is k-PP-implementable, if and only if*

- **i.** *it is CRN-implementable, i.e., there is a CRN* $(\mathbf{x}, \mathbf{x}')$ *computing it.*
- **ii.** *for all* x_i *in* $\mathbf{x}$*,* x'_i *does not possess a positive* x_i^k *term.*
- **iii.** $\mathbf{x}'$ *is homogeneously degree* k*, and*
- **iv.** $\mathbf{x}'$ *is conservative.*

From above, we see that turning a GPAC into a PP means having to dance with more and more chains. We must carefully appease all restrictions, which makes the translation process difficult.

We first revisit some useful concepts regarding PPs.

2.2 Probabilistic Large-Population Protocols

▶ **Definition 5** (Balance Equations [6]). *Let* (Q, R) *be a large-population protocol, where* Q *is the state (species, variables) set and* R *is a set of reaction rules, each with two reactants and two products. The* balance *equation of a deterministic LPP is a function* $b : \mathbb{R}^{|Q|} \rightarrow \mathbb{R}^{|Q|}$ *such that*

$$b(x) = \sum_{(q_1,q_2)\in Q^2} \left(x_{q_1} x_{q_2} \big(- e_{q_1} - e_{q_2} + \sum_{(q_3,q_4)\in Q^2} \delta_{q_1,q_2,q_3,q_4} (e_{q_3} + e_{q_4}) \big) \right) \tag{4}$$

where $\delta_{q_1,q_2,q_3,q_4} = 1$ *if* $(q_1, q_2) \rightarrow (q_3, q_4)$*, and 0 otherwise;* $(e_q)_{q\in Q}$ *is the canonical base of* $\mathbb{R}^{|Q|}$ *and* x_{q_i} *is the proportion of the population in state* q_i*.*

The balance equation is a description of the system’s dynamics. Intuitively, it says whenever q_1 and q_2 bump into each other, if there is a reaction $(q_1, q_2) \rightarrow (q_3, q_4)$ in the PP, then the pair (q_1, q_2) turns into (q_3, q_4); otherwise, we interpret (q_1, q_2) as a *null reaction* and the agents remain unchanged. We adopt an *asymmetric* interpretation of reactions in PPs throughout the paper. That is, the ordered pairs (q_i, q_j) and (q_j, q_i), when used as reactants, do not necessarily result in the same products. The assumption is not essential: the choice is to align with the existing literature (e.g., [6]) for convenience.

The above balance equation is for deterministic PPs, where δ_{q_1,q_2,q_3,q_4} is either 0 or 1. It is more convenient to use probabilistic transition rules in many scenarios. We introduce probabilistic LPPs (PLPPs).

▶ **Definition 6** (Probabilistic LPP (PLPP) [6]). *A PLPP is an LPP with transition rules in the form*

$$q_i \; q_j \rightarrow \alpha_{i,j,k,l} \; q_k \; q_l$$

and for every $(q_i, q_j) \in Q^2$, we have

- *for every $(q_k, q_l) \in Q^2$, $\alpha_{i,j,k,l} \in \mathbb{Q}$ and $\alpha_{i,j,k,l} > 0$, and*
- $\sum\limits_{(q_k,q_l)} \alpha_{i,j,k,l} = 1.$

The balance equation remains mostly the same as Equation (4). To simplify the notations, we often take $Q = [n] = \{1, 2, \cdots, n\}$ and will still use q_i or use terms like "state i" when referring to a state.

$$b(x) = \sum_{(i,j)\in[n]^2} \left(x_i x_j \big(-e_i - e_j + \sum_{(k,l)\in[n]^2} \alpha_{i,j,k,l}(e_k + e_l) \big) \right) \tag{5}$$

The ODE associated with a PLPP can be written as

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = b(\mathbf{x}), \tag{6}$$

where $\mathbf{x} = (x_1(t), \cdots, x_n(t)) \in \mathbb{R}^n$ is a variable vector and its i-th component, x_i, tracks the proportion of the total population in state i over time t. We will often call $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ or $\mathbf{x}$ an LPP (PLPP). ⌟

A simple observation results directly from the definition of LPP or PLPP.

▶ **Observation 7** ("The one trick"). *Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ be an LPP or PLPP, then $\sum\limits_{i\in[n]} x_i = 1$.*

Although obvious, the above is the single *most important* observation in this paper. Many constructions and proofs rely on it. This observation is frequently used to re-write the constant term in an ODE. We will rewrite a constant c in either of the following ways, depending on our needs:

- $c = c \cdot 1 = c(x_1 + \cdots + x_n)$, or
- $c = c \cdot 1 \cdot 1 = c(x_1 + \cdots + x_n)(x_1 + \cdots + x_n)$. ⌟

▶ **Observation 8.** *The r-th component of Equation (6) has the form*

$$\frac{\mathrm{d}x_r}{\mathrm{d}t} = f(\mathbf{x}) - 2x_r, \tag{7}$$

where $f(\mathbf{x})$ is a quadratic form in $\mathbb{P}^+$.

An intuitive way to think about equation 7:

- The term $-2x_r$ represents all reactions that *consume* x_r. A complete set of rules for a PP must include all combinations of pairs (x_r, x_k), or (x_k, x_r), where k iterates through all $[n]$. These combinations sum to $-2x_r$.
- The $f(x)$ represents all reactions that *produce* x_r. This observation is critical in the constructive proof of our main theorem.

We now give a formal definition of LPP-computable numbers.

2.3 Large-Population Protocol Computable Numbers: Revisited

We extend the definition from [6]. Note that the definition also applies to large-population *unimolecular* protocols (LPUPs).

▶ **Definition 9.** *A real number ν is said to be computable by an LPP (PLPP, LPUP) if there exists an LPP (PLPP, LPUP) such that $\mathbf{x}(t) = (x_1(t), x_2(t), \cdots, x_n(t)) \in [0,1]^n$ and*

$$\lim_{t\to\infty} \sum_{i\in M} x_i(t) = \nu,$$

where $M \subseteq \{1, \cdots, n\}$ represents the subset of states marked in $\mathbf{x}$. Moreover, all the states x_i must be initialized to some positive rational $r_i \in \mathbb{Q} \cap [0,1]$, in the sense that $\lim_{N\to\infty} x_i^{(N)}(0) = r_i$, when $x_i^{(N)}(0)$ is the initial fraction of state i at the stage when the population is N.

A major change is our removal of the finitary requirement on equilibria. A note originally in [6] discusses the rationale for the requirement of finitely many equilibria. This assumption is needed to *"avoid pathological cases, in particular the case of idle systems $q\ q' \to q\ q'$ for all q and q'. Indeed, in idle systems, all initial states are equilibria, and such a system would compute any real of [0,1], depending on the initial configuration."*

This pathological behavior could occur, if one is allowed to initialize the system with any real numbers. However, we can prevent this case by restricting the initial configuration to rational numbers. Also note that by forcing the initial counts of a species X to be a fraction of the total population N, we exclude those systems (e.g.,[12, 1]) that crucially depend on small counts in a very large population. These systems are not modeled correctly by ODEs. For instance, one can not initialize X to have 100 molecules. Any state initialized to a constant amount will be a fraction that approaches zero, as N grows to infinity. In our definition, such X will be initialized to have *zero* molecules for a fixed N.

▶ Remark. It is worth noting the above setup satisfies the assumptions of Kurtz's Theorem [26]. Therefore, the long-term behavior of an LPP is governed by an ODE, which is induced by its *balance equation*. As a result, in this paper, we often blur the boundary between the continuous semantics (by ODE) and stochastic semantics when discussing limiting behavior. See Section 4 for initial discussion on further investigation on the two semantics.

For the rest of the paper, whenever we mention LPP computable number, we mean the extended notion defined above, unless specified otherwise.

2.4 Basic Results

2.4.1 Large-Population UPPs Compute Only Rationals

As a toy case study, we investigate the class of numbers computable by large-population unimolecular protocols (LPUP). We find that unimolecular protocols have limited power.

▶ **Lemma 10.** *A number is LPUP-computable if and only it is rational.*

Proof. Suppose $\nu = \frac{p}{q} \in [0,1]$ is a rational number. Lemma 3 of Bournez et al. [6] constructs an LPP with q states such that each state converges $\frac{1}{q}$ for an exponentially stable equilibrium. We modify their construction to obtain a unimolecular LPP. Let $X = \{X_1, \ldots, X_q\}$ be the state set and the transitions be of the form $X_i \to X_{(i \mod q)+1}$ (see Figure 2 for a visualization). Then each state will converge to $\frac{1}{q}$. To compute ν, mark states $X_1, \ldots, X_p$. Thus, there exists a deterministic unimolecular LPP that computes ν. (Alternatively, one could initialize an idle system with the desired rationals, since now the extended notion of LPP-computable number allows a continuum of equilibria and rational intial values.)

Suppose a deterministic unimolecular protocol $\mathcal{P}$ computes some number $\nu \in [0, 1]$. Then there is a graph G corresponding to the reactions of $\mathcal{P}$ such that nodes are states and directed edges are transitions. The deterministic nature ensures that self-loops exist only if a node is isolated from all other nodes. Since each node has the outdegree 1, G is a functional graph. Functional graphs are a set composed of rooted trees attached to a cycle ([14], page 129; or see Figure 3 for an example). Traversing the graph from any starting point will eventually lead to being stuck in a cycle. The proportion of mass that flows into a cycle will become uniformly distributed among the cycle's finite number of nodes, computing a rational number. Thus, deterministic unimolecular protocols compute rational numbers. ◀

Note that the above lemma can be extended to probabilistic unimolecular protocols, but we omit this work for the sake of simplicity.

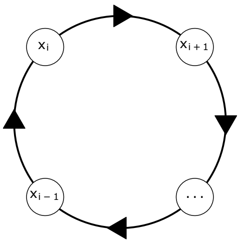


Figure 2 Visualization of transitions in a unimolecular protocol.

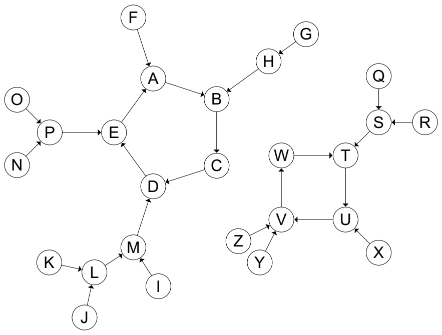


Figure 3 Example of a Functional Graph.

2.4.2 Product Protocols and Multiplication

We will show that LPP-computable numbers are closed under multiplication. The proof is done through product protocols. Since our Main Theorem shows that LPP-computable numbers are essentially GPAC-computable, many other arithmetic operations can be done by taking a detour via GPACs. However, the resulting protocols are not so intuitive. In addition, the product protocol is an important technique for our later construction in Section 3.

▶ **Lemma 11** (Product of LPP-computable Numbers is LPP-computable). *Let P_x and P_y be LPPs that compute α and β respectively. Denote their respective state vectors by $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ and $\mathbf{y} = (y_1, y_2, \cdots, y_m)$, initial configurations by $\mathbf{x}(0)$ and $\mathbf{y}(0)$, and differential systems by $\mathbf{x}'$ and $\mathbf{y}'$. Assume any differential equation in $\mathbf{x}'$ or $\mathbf{y}'$ is a quadratic form. Then there exists an LPP, P_z, that computes the product $\alpha\beta$.*

Proof. Let $p_{x_i}, q_{x_i}, p_{y_j}, q_{y_j} \in \mathbb{P}^+$. Then we can write $x_i' = p_{x_i} - x_i q_{x_i}$ for all $x_i' \in \mathbf{x}'$. Since x_i' is a quadratic form, p_{x_i} is a quadratic form and q_{x_i} is a linear form. This notation generalizes to $y_j' \in \mathbf{y}'$ as well. Introduce new state variables $z_{i,j} := x_i y_j$ for all $(x_i, y_j) \in \mathbf{x} \times \mathbf{y}$ where $\times$ is the Cartesian product (view $\mathbf{x}$ and $\mathbf{y}$ as sets). Then we have the state vector $\mathbf{z} = (z_{1,1}, z_{1,2}, z_{2,1}, \ldots, z_{n,m})$. Suppose $M_{\mathbf{x}} \subset \mathbf{x}$ and $M_{\mathbf{y}} \subset \mathbf{y}$ are the marked states. For each $z_{i,j} = x_i y_j \in \mathbf{z}$, mark $z_{i,j}$ if $x_i \in M_{\mathbf{x}}$ and $y_j \in M_{\mathbf{y}}$. For the initial values, let $\mathbf{z}(0) = \mathbf{x}(0) \cdot \mathbf{y}(0)$.

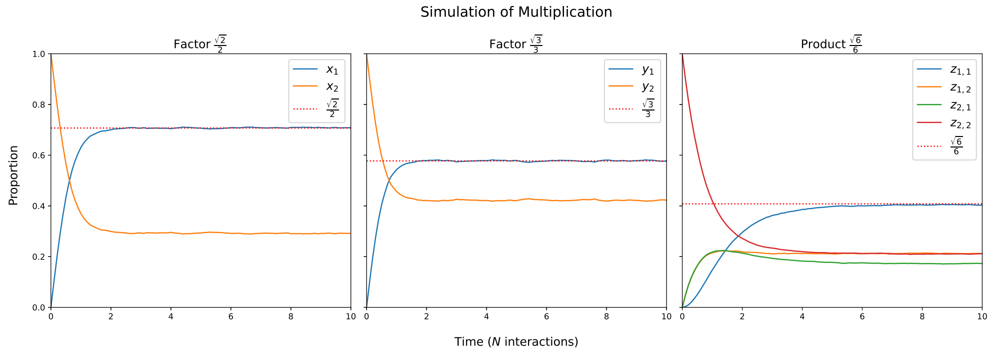


Figure 4 With ppsim, we can simulate a product protocol with input LPPs $\mathbf{x}$ and $\mathbf{y}$ that compute $\frac{1}{\sqrt{2}}$ and $\frac{1}{\sqrt{3}}$. Note that the factor LPPs originate from [5], and the product LPP is a result of applying Lemma 11. These simulations affirm the closure of LPPs under multiplication.

To obtain the differential system $\mathbf{z}'$, perform the following steps. First, differentiate $z_{i,j} = (x_i y_j)$. By the chain rule, we have

$$z'_{i,j} = (x_i y_j)' = x'_i y_j + x_i y'_j.$$

Next, substitute the equations for x'_i and y'_j. Then, we have the cubic form,

$$z'_{i,j} = y_j(p_{x_i} - x_i q_{x_i}) + x_i(p_{y_j} - y_j q_{y_j}).$$

Apply the one-trick (Proposition 7) to obtain a homogeneously degree four polynomial such that every term is exactly degree two in variable x and exactly degree two in variable y. We have,

$$z'_{i,j} = y_j \left(\sum_{k=1}^{m} y_k \right) (p_{x_i} - x_i q_{x_i}) + x_i \left(\sum_{k=1}^{n} x_k \right) (p_{y_j} - y_j q_{y_j}).$$

After expanding the polynomial, group the positive and negative terms. Note that we can factor $x_i y_j$ from the collection of negative terms, which yields

$$z'_{i,j} = \left(\sum_{k=1}^{m} y_k y_j p_{x_i} + \sum_{k=1}^{n} x_k x_i p_{y_j} \right) - (x_i y_j) \left(q_{x_i} \sum_{k=1}^{m} y_k + q_{y_j} \sum_{k=1}^{n} x_k \right).$$

By substituting $z_{i,j}$ for the factored $x_i y_j$, we have

$$z'_{i,j} = \left(\sum_{k=1}^{m} y_k y_j p_{x_i} + \sum_{k=1}^{n} x_k x_i p_{y_j} \right) - z_{i,j} \left(q_{x_i} \sum_{k=1}^{m} y_k + q_{y_j} \sum_{k=1}^{n} x_k \right),$$

which satisfies condition (i) from Corollary 3.

Let $p_{z_{i,j}} = \left(\sum_{k=1}^{m} y_k y_j p_{x_i} + \sum_{k=1}^{n} x_k x_i p_{y_j} \right)$. Without loss of generality, consider $p^*_{z_{i,j}} = \sum_{k=1}^{m} y_k y_j p_{x_i}$. Note that every monomial in $p^*_{z_{i,j}}$ is of the form $x_u x_v y_j y_k$ where $x_u x_v$ is a monomial in p_{x_i}. Then we can substitute either $z_{u,j} z_{v,k}$ or $z_{u,k} z_{v,j}$ for each $x_u x_v y_j y_k$ in $p^*_{z_{i,j}}$. Since x_i^2 is not a valid monomial of p_{i_x}, we will not have $z_{i,j}^2$ in $p^*_{z_{i,j}}$. Thus, $p_{z_{i,j}}$ is a quadratic form in variable z that satisfies condition (ii) of Corollary 3.

Let $q_{z_{i,j}} = \left(q_{x_i} \sum_{k=1}^{m} y_k + q_{y_j} \sum_{k=1}^{n} x_k \right)$. Without loss of generality, consider $q^*_{z_{i,j}} = q_{x_i} \sum_{k=1}^{m} y_k$. Note that every monomial in $q^*_{z_{i,j}}$ is of the form $x_u y_k$, where x_u is a monomial in q_{x_i}. Then we can substitute $z_{u,k}$ for each $x_u y_k$ in $q^*_{z_{i,j}}$. Thus, we can rewrite $z'_{i,j}$ as

$$z'_{i,j} = p_{z_{i,j}} - z_{i,j} q_{z_{i,j}},$$

where $z'_{i,j}$ is a quadratic form and meets all conditions of Corollary 3.

Hence, we have defined the state set, initial configuration, and differential system of a PP-implementable protocol that computes $\alpha\beta$. ◀

3 Translating Bounded GPACs into PPs

3.1 Construction Overview

We will prove our main theorem in this section. It suffices to show that LPPs can simulate GPACs. Since one can transform a bounded GPAC into a bounded CRN with zeroed initial values [23], it is sufficient to consider only this type of CRN. We develop a four-stage algorithm that translates CRNs into LPPs.

- Input: A CRN that computes a target number α. We now enter a cascade of transformations:
- Stage 1: into a CRN-implementable quadratic system,
- Stage 2: into a TPP-implementable cubic form system,
- Stage 3: into a PP-implementable quadratic form system, and
- Stage 4: into a PLPP.

3.2 Basic Operations

First, we discuss some basic operations as building blocks for our constructive proof. The notations used are as described in the preliminary section. The naming of operations "ε-trick" and "λ-trick" (the use of the Greek letters ε and λ) originates in the proof of Lemma 5 in [6].

▶ **Operation 1** (Constant Dilation, "ε-trick"). *Let $\mathbf{x}(t)$ be the solution to the ODE $\mathbf{x}'(t) = p(\mathbf{x}(t))$ and $\varepsilon \in \mathbb{Q}^+$ a positive constant, then $\mathbf{x}(\varepsilon t)$ is the solution to the ODE*

$$\mathbf{x}'(t) = \varepsilon p(\mathbf{x}(t)).$$

The operation is essentially a change of variable $t \to \varepsilon t$ and a direct result of the chain rule. The behavior of the ODE system will not change; it only alters the flow of time. Indeed, t in the new system corresponds to εt of the original system. In our construction, we will use this operation to shrink the coefficient of an ODE system. The operation can be seen in [11, 6].

We can do a more general change of variable (composition.)

▶ **Operation 2** (Time Dilation ([28], Theorem 3)). *Let $\mathbf{x}(t)$ be the solution to the GPAC $\mathbf{x}(t)' = p(\mathbf{x}(t))$, $g(t)$ be a GPAC-implementable function, and $G(t) = \int_0^t g(s)ds$, then $\mathbf{x}(G(t))$ is the solution to the GPAC*

$$\mathbf{x}'(t) = p(\mathbf{x}(t)) \cdot g.$$

In our construction, we must choose $g(t)$ such that its integral satisfies

$$G(t)|_{t=\infty} = \int_0^\infty g(s)ds = \infty.$$

and we call such a $G(t)$ as "nonterminating time".

A time dilating operation, as above, controls the flow of time with the function $g(t)$. More accurately, t in the new system corresponds to $G(t) = \int_0^t g(s)ds$ of the original system. If we want the new system to mimic the old system's limiting behavior (i.e., to achieve the value $\lim_{t\to\infty} x_1(t)$), we must guarantee $G(\infty) = \infty$. Thus, time in the new system is nonterminating and can go to infinity in the eyes of the old system. Otherwise, the new system would terminate somewhere in the middle, relative to the old system. To obtain nonterminating time, it is enough to consider functions that converge slowly, e.g., $g(t) = \frac{1}{1+t}$. Alternatively, one could ensure g is bounded from below, i.e., $g(t) > c > 0$, for some constant c.

Time dilation for CRNs can be found in [25] (page 15, Theorem 3.7). ⌟

Next, we introduce an operation that shrinks variables to [0,1].

▶ **Operation 3** (Variable Shrinking, or "λ-trick"). *Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ be a bounded CRN that computes a number α in $[0,1]$. That is, $\lim_{t\to\infty} x_1(t) = \alpha$. To transform the CRN into a LPP, we must ensure all other components, some of which may not not converge, fall in [0,1] for all t. We can pick a $0 < \lambda < 1$ sufficiently small and a constant $c > 0$ such that*

$$x_1 + \lambda(x_2 + \cdots + x_n) < 1 - c.$$

This is possible since all variables are bounded. Then we

- *make a change of variables: $x_2 \leftarrow \lambda x_2, \cdots, x_n \leftarrow \lambda x_n$, and*
- *introduce a new variable $x_0(t)$ such that $x_0(t) + x_1(t) + \cdots + x_n(t) = 1$ by*
  1. *initialing $x_0(0) = 1$ (recall $x_i(0) = 0$ for $i = 1, \cdots, n$), and*
  2. *adding the ODE for x_0: $x_0' = -\sum_{i=1}^n x_i'$.*

Note that x_1 remains unchanged in the operation, so it still computes α. ⌟

The biggest challenge in translating a CRN into an LPP is: the CRN does not necessarily preserve population, while the LPP requires it. In fact, our input CRN initializes at zero. However, x_1 needs to compute some number α as all other components remain positive throughout time; evidently, not preserving the population. A straightforward idea to "balance" the system, is to introduce a variable x_0, such that its derivative cancels all changes to the population size by other variables. That is, $x_0' = -\sum_{i=1}^n x_i'$.

However, the resulting x_0 is not CRN-implementable in general. As a result, it is not transformable to an LPP. The rationale is clear in the derivative of x_i. In general, x_i' has the form

$$x_i' = p_i - q_i \cdot x_i, \quad \text{where } p, q \in \mathbb{P}^+.$$

A typical p_i would contain at least one monomial m. Evidently, $x_0 \notin m$ since the variable x_0 is newly introduced. Then m will contribute to a negative term in x_0' and $x_0 \notin m$, making x_0 not CRN-implementable and, in turn, not PP-implementable. That is why Bournez et al.'s construction in [6] (Lemma 2) fails to produce a population protocol in general. To address the this issue, we purposefully *multiply* x_0 with every x_i' and introduce the following operation.

▶ **Operation 4** (Balancing Dilation, or "Garbage Collection" ("g-trick")). *Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ be a CRN-computable system, (i.e., every component x_i is CRN-computable). Then the following operation (multiplying an* unknown *variable x_0 determined by an ODE) produces a conservative CRN-implementable system.*

$$\begin{cases} x_i' = (p_i - q_i \cdot x_i) \cdot x_0, & \textit{for } i \in \{1, \cdots, n\} \\ x_0' = -\sum_{i=1}^n x_i'. \end{cases}$$

What makes the operation different from a normal time dilation operation is, the function x_0 is not known in advance: It needs to adjust/customize according to the system it is dilating. In our construction, we also need to combine the operation with the λ-trick to achieve nonterminating time. This operation also turns a non-conservative system into a conservative *CRN-implmentable* one.

We will present our construction in the following subsections.

3.3 Stage 1: Conversion to CRN-implementable Quadratic Systems

The following is a CRN version of [9] (Theorem 1).

▶ **Theorem 12.** *Any solution of a CRN is a solution of a CRN-implementable system of degree at most two.*

With this result, we can assume a CRN's associated polynomial system is quadratic with out loss of generality.

3.4 Stage 2: Conversion to TPP-implementable Cubic Form Systems

▶ **Theorem 13.** *Any solution of a quadratic CRN is a solution of a TPP-implementable cubic form system.*

We sketch the proof here. Please see the appendix for more detail.

Proof. Given a quadratic CRN $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, we first apply the λ-trick to "make room" for a new variable x_0. That is, we choose a $0 < \lambda < 1$ and a constant $0 < c < 1$, such that $x_1 + \lambda(x_2 + \cdots + x_n) < 1 - c$. Then apply the one-trick and balancing dilation to get a cubic form:

1. Introduce a new variable x_0
2. Rewrite each p_i (quadratic), q_i (linear) in $x_i' = p_i - q_i \cdot x_i$ to forms:
   - for a linear monomial in p_i of the form $a \cdot x_j$, apply the "one"-trick and rewrite it as $ax_j \sum_{k=0}^n x_k$.
   - for a constant term in p_i of the form a, rewrite it as $a \cdot (\sum_{k=0}^n x_k) \cdot (\sum_{k=0}^n x_k)$.
   - for a constant term a in q_i, rewrite it as $a \cdot \sum_{k=0}^n x_k$.

   We call the resulting forms P_i and Q_i respectively.
3. Apply balancing dilation with x_0. The new system looks like

$$\begin{cases} x_i' = (P_i - Q_i x_i) \cdot x_0, & \text{for } i = 1, \cdots, n \\ x_0' = -\sum_{i=1}^n x_i'. \end{cases} \quad \text{and} \quad \begin{cases} x_i(0) = 0, & \text{for } i = 1, \cdots, n \\ x_0(0) = 1. \end{cases}$$

It is a TPP-implementable cubic form system. See the appendix for a full proof. ◀

▶ Remark 14. Note that in the above proof x_0 is bounded from below, away from zero. We can observe that:

1. The variable x_0 starts from 1 and remains greater than $c > 0$ for all t. It will not introduce new equilibria.
2. The new system has at most a linear slowdown compared to the old system, since $\int_0^t x(0)\,\mathrm{d}t > \int_0^t c\,\mathrm{d}t = ct$.

The variable x_0 acts like a source, distributing the total mass (sum = 1) to all other variables (initialized at 0).

3.5 Stage 3: Conversion to PP-implementable Quadratic Form Systems

▶ **Theorem 15.** *A number computable by a TPP-implementable function given by Stage 2 can be computed by the sum of a finite set of PP-implementable functions.*

Proof. Sketch of the proof. We utilize the self-product of the TPP-implementable cubic form system. Let $\mathbf{x} = (x_0, x_1, \cdots, x_n)$ be the system resulting from the previous stage that computes a real number α with initial values $x_0 = 1$ and 0 otherwise. We consider the system $\left(z_{i,j}\right)_{(i,j)\in[n+1]^2}$ where $z_{i,j} = x_i \cdot x_j$ and $[n+1] = \{0, 1, \cdots, n\}$. Let $z_{0,0}(0) = 1$ and

0 otherwise. The marked set $M_z = \{z_{1,j} | 0 \le j \le n\}$. Then we can verify the z-system is PP-implementable and the sum of the marked variables traces α. Check the appendix for a full proof. ◀

3.6 Stage 4: Converting PP-implementable Quadratic Form Systems to PLPPs

In this subsection, we turn the PP-implementable quadratic form given as a result of the previous stage into a PLPP. The following construction is from the proof of Lemma 5 in [6].

▶ **Construction 1.** *Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ be a PP-implementable quadratic form. We denote the derivative x_i as*

$$x_i' = p_i - q_i x_i, \quad i = 1, 2, \cdots, n, \tag{8}$$

where p_i is a quadratic form and q_i is a linear form.

We will apply the ε-trick (Operation 1) to shrink the coefficient of the above system, where $\varepsilon \in \mathbb{Q}$ is an undetermined parameter. In order to align with the form in Equation (7), we add and subtract a term $2x_i$. Now the system has the form

$$x_i' = \varepsilon(p_i - q_i x_i) + 2x_i - 2x_i, \quad i = 1, 2, \cdots, n. \tag{9}$$

$$= f_i(\mathbf{x}), \quad \text{denote } f_i(\mathbf{x}) = \varepsilon(p_i - q_i x_i) + 2x_i. \tag{10}$$

We then use the one-trick to rewrite $f_i(\mathbf{x}) = \varepsilon(p_i - q_i x_i) + 2x_i \sum_{j=1}^n x_j$. Note that the term brings in $2x_i x_j$ for $j = 1, \cdots, n$. We will then pick ε sufficiently small

1. *to ensure the $2x_i x_j$ terms dominate the monomials in $-qx_i$ such that $f_i(\mathbf{x}) \in \mathbb{P}^+$ for all i.*
2. *and to ensure*

$$\frac{\sum_{i,j \ne k} C(x_j x_k, f_i)}{4} \le 1 \quad \text{and} \quad \frac{\sum_{i,j} C(x_j^2, f_i)}{2} \le 1 \tag{11}$$

where $C(x_j x_k, f_i)$ denotes the coefficient of $x_j x_k$ in f_i, and $i, j, k \in \{1, \cdots, n\}$.

Next, we construct a rule set for a PLPP. For an ordered pair (x_j, x_k) and a term $\alpha_{i,j,k} x_j x_k$ in $f_i(\mathbf{x})$ we do the following:

- *If $j = k$, add a rule $(x_j, x_j) \to \frac{\alpha_{i,j,k}}{2}(x_i, x_i)$;*
- *otherwise, add two rules $(x_j, x_k) \to \frac{\alpha_{i,j,k}}{4}(x_i, x_i)$ and $(x_k, x_j) \to \frac{\alpha_{i,j,k}}{4}(x_i, x_i)$;*
- *In the above rules, if $s = \sum_{i=1}^n \alpha_{i,j,k} < 1$ for an order pair (k, l), we add an idle reaction*

$$(x_j, x_k) \to (1 - s)(x_j, x_k);$$

If a term $x_j x_k$ appears in $f_i(\mathbf{x})$, it implies that the pair (x_j, x_k) will produce x_i. We use the all-in greedy strategy, that is, we produce two x_i's and let the pair (x_k, x_j) do the same. So we need to divide the coefficient $\alpha_{i,j,k}$ by 4 for $j \ne k$; yet, when $k = j$ we don't have a flipped order pair, so we only divide the coefficient by 2.

There need to make sure that the rule set generated above forms a PLPP.

▶ **Theorem 16.** *Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ be a quadratic system satisfying $\sum_{i=1}^n x_i = 1$. Then it can be turn into a PLPP by Construction 1 if and only if it is PP-implementable.*

Proof. See appendix. ◀

The following theorem can help in transforming a PLPP to an LPP.

▶ **Theorem 17** ([6], Lemma 4). *Let $\nu \in [0, 1]$, and assume that there exists a PLPP computing ν, with rational probabilities. Then there exists a (deterministic) LPP computing ν.*

Note that although the above theorem is based on the original notion of LPP-computable number with the finite-equilibria constraint, the construction in the proof of the theorem still correctly encodes the balance equation. So the theorem also holds under the new definition.

We therefore finish the proof of our main theorem.

▶ **Main Theorem.** *LPPs compute the same set of numbers in [0,1] as GPACs and CRNs.*

▶ **Corollary 18.** *Algebraic numbers are LPP computable.*

Proof. Algebraic numbers are computable computable by a GPAC [6] (Lemma 5) so they are also LPP-computable by the main theorem. ◀

The above corollary fix the gap in Bournez et al.'s construction.

▶ **Corollary 19.** *Famous math constants $\frac{\pi}{4}$, e^{-1}, Euler's γ, Dottie number, and Catalan's constant are LPP-computable.*

This result answers Bournez et al.'s question about whether solutions of trigonometric equations (Dottie number) and $\frac{\pi}{4}$ are "computable" asymptotically by population protocols.

4 Conclusion and Discussion

In this paper, we extend the notion of LLP-computable numbers and connect it with GPAC/CRN-computable numbers. Moreover, we show that LLPs and GPACs/CRNs essentially compute the same set of real numbers. The result, to some degree, says LPPs, or more straightforwardly, population protocols with the mass-action semantic, can simulate GPACs in some way. We would ask a natural question: What is the next "weak" or minimal (according to some measure) model that can simulate GPACs? Unimolecular protocols are provably incapable since they only compute rational numbers. However, a model discussed in [16], which can be viewed as a two-state ("black" or "white") k-PP with restrictions on reactions (the product of a reaction must either be all black or all white), can compute algebraic numbers such as $\frac{3-\sqrt{5}}{2}$ but not rational numbers like $\frac{1}{5}$. Diving into the sub-models of population protocols and characterizing the power spectrum will be an interesting avenue for further exploration.

The ODE systems we used to compute transcendental numbers have a continuum of equilibria. We have not addressed the semistability nor the system's convergence rate in this paper. Regarding the latter, since our construction essentially rewrites the system with a new set of auxiliary variables, the system's solutions in some sense remain the same, except for the *linear* slowdown introduced in Stage 2. It is hoped that an LPP's convergence rate is similar to the input CRN. If an input CRN converges exponentially fast to a number α, we desire the translated LPP also converges (exponentially) fast. We would need a thorough stochastic analysis along this line in future work.

The definition of GPAC/CRN-computable numbers and LPP-computable numbers have a major difference: We can trace a real number by a *set* of variables in an LPP, rather than one, as in a GPAC/CRN. Currently, the proof of our main theorem relies on using a set of marked variables. Our question: Is the difference intrinsic? Or can we compute the same set of numbers using *one* variable?

Another area of exploration is protocols with a large population (tending to infinity) that also have "scarce" variables with a limited population. The behaviors of these systems, due to the existence of the scarce variables, can not be governed by Kurtz's theorem. Therefore, a new analytic tool or theoretical benchmark (probably not a computable number) awaits development.

Our construction in Section 3 is not optimal. To implement the translation algorithm in a software package, one would want to have a better algorithm for Stage 1. To apply the λ-trick, one needs to know the bounds of the variables, which would require numerical methods. A theory to predict the bounds is desired but difficult to achieve due to the non-linear nature of ODEs.

References

1 Dan Alistarh, Bartłomiej Dudek, Adrian Kosowski, David Soloveichik, and Przemysław Uznański. Robust detection in leak-prone population protocols. In *International Conference on DNA-Based Computers*, pages 155–171. Springer, 2017.

2 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

3 Guillaume Aupy and Olivier Bournez. On the number of binary-minded individuals required to compute. *Theoretical Computer Science*, 412(22):2262, 2011.

4 Sanjay P. Bhat and Dennis S. Bernstein. Nontangency-based lyapunov tests for convergence and stability in systems having a continuum of equilibria. *SIAM Journal on Control and Optimization*, 42(5):1745–1775, 2003.

5 Olivier Bournez, Philippe Chassaing, Johanne Cohen, Lucas Gerin, and Xavier Koegler. On the convergence of population protocols when population goes to infinity. *Applied Mathematics and Computation*, 215(4):1340–1350, 2009.

6 Olivier Bournez, Pierre Fraigniaud, and Xavier Koegler. Computing with large populations using interactions. In *Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science*, pages 234–246. Springer, 2012.

7 Olivier Bournez, Daniel S. Graça, and Amaury Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length. *Journal of the ACM*, 64(6):38, 2017.

8 Vannevar Bush. The differential analyzer. A new machine for solving differential equations. *Journal of the Franklin Institute*, 212(4):447–488, 1931.

9 David C. Carothers, G. Edgar Parker, James S. Sochacki, and Paul G. Warne. Some properties of solutions to polynomial systems of differential equations. *Electronic Journal of Differential Equations (EJDE)[electronic only]*, 2005:Paper–No, 2005.

10 David Cox, John Little, and Donal OShea. *Ideals, Varieties, and Algorithms: an Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer Science & Business Media, 2013.

11 David Doty and Eric Severson. Ppsim: A software package for efficiently simulating and visualizing population protocols. In *International Conference on Computational Methods in Systems Biology*, pages 245–253. Springer, 2021.

12 Bartłomiej Dudek and Adrian Kosowski. Universal protocols for information dissemination using emergent signals. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 87–99, 2018.

13 François Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *Proceedings of the 15th International Conference on Computational Methods in Systems Biology*, pages 108–127. Springer International Publishing, 2017.

14 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

15 Willem Fletcher, Titus H. Klinge, James I. Lathrop, Dawn A. Nye, and Matthew Rayman. Robust real-time computing with chemical reaction networks. In *International Conference on Unconventional Computation and Natural Computation*, pages 35–50. Springer, 2021.

16 Lucas Gerin and Marie Albenque. On the algebraic numbers computable by some generalized ehrenfest urns. *Discrete Mathematics & Theoretical Computer Science*, 14, 2012.

17 Daniel S. Graça. Some recent developments on shannon's general purpose analog computer. *Mathematical Logic Quarterly: Mathematical Logic Quarterly*, 50(4-5):473–485, 2004.

18 Daniel Silva Graça and José Félix Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19(5):644–664, 2003.

19 Daniel S. Graça, Manuel L. Campagnolo, and Jorge Buescu. Computability with polynomial differential equations. *Advances in Applied Mathematics*, 40(3):330–349, 2008.

20 Indranil Gupta, Mahvesh Nagda, and Christo Frank Devaraj. The design of novel distributed protocols from differential equations. *Distributed Computing*, 20(2):95–114, 2007.

21 Vera Hárs and János Tóth. On the inverse problem of reaction kinetics. In *Colloquia Mathematica Societatis János Bolyai, 30: Qualitative Theory of Differential Equations*, pages 363–379, 1981.

22 Xiang Huang. *Chemical Reaction Networks: Computability, Complexity, and Randomness.* PhD thesis, Iowa State University, 2020.

23 Xiang Huang, Titus H. Klinge, and James I. Lathrop. Real-time equivalence of chemical reaction networks and analog computers. In *International Conference on DNA Computing and Molecular Programming*, pages 37–53. Springer, 2019.

24 Xiang Huang, Titus H. Klinge, James I. Lathrop, Xiaoyuan Li, and Jack H. Lutz. Real-time computability of real numbers by chemical reaction networks. *Natural Computing*, 18(1):63–73, 2019.

25 Titus H. Klinge. *Modular and Robust Computation with Deterministic Chemical Reaction Networks.* PhD thesis, Iowa State University, 2016.

26 Thomas G. Kurtz. The relationship between stochastic and deterministic models for chemical reactions. *The Journal of Chemical Physics*, 57(7):2976–2978, 1972.

27 Leonard Lipshitz and Lee A Rubel. A differentially algebraic replacement theorem, and analog computability. *Proceedings of the American Mathematical Society*, 99(2):367–372, 1987.

28 G. Edgar Parker and James S. Sochacki. Implementing the picard iteration. *Neural, Parallel & Scientific Computations*, 4(1):97–112, 1996.

29 Marion B Pour-El and Jonathan I Richards. Abstract computability and its relations to the general purpose analog computer. *Transactions of the American Mathematical Society*, 199:1–28, 1974.

30 Claude E Shannon. Mathematical theory of the differential analyzer. *Studies in Applied Mathematics*, 20(1-4):337–354, 1941.

31 Justin R. Smith. *Introduction to Algebraic Geometry.* Justin Smith, 2014.

32 Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1936.

A Optional Technical Appendix

Proof of Observation 8. Take the x_r component in Equation (6). We have

$$\begin{aligned}\frac{\mathrm{d}x_r}{\mathrm{d}t} &= \sum_{j\in[n]}\Big(-x_r x_j\Big) + \sum_{j\in[n]}\Big(-x_r x_j\Big)\\ &\quad + Proj_r\left(\textstyle\sum_{i,j\in[n]}\left(x_i x_j \sum_{k,l\in[n]} \alpha_{i,j,k,l}(e_k+e_l)\right)\right),\\ &= -2x_r + f(\mathbf{x}), \quad \text{by Observation 7.}\end{aligned}$$

where $f(\mathbf{x}) = Proj_r\left(\sum_{i,j\in[n]}\left(x_i x_j \sum_{k,l\in[n]} \alpha_{i,j,k,l}(e_k + e_l)\right)\right)$ and $Proj_r(\cdot)$ is the r-th component a vector. This completes the proof. ◀

Proof of Theorem 12. Let $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ be a CRN, then each x_i has the form $x_i' = p_i - q_i x$, where $p_i, q_i \in \mathbb{P}^+$. We in introduce variables (we call them v-variables) for each monomial as follows:

$$v_{i_1,\cdots,i_n} = x_1^{i_1} x_2^{i_2} \cdot x_n^{i_n}.$$

Specially, $v_{1,0,\cdots,0} = x_1$, $v_{0,1,0,\cdots,0} = x_2$, and so on. Note that we can rewrite p_i and q_i in each x_i' with the v-variables and denote them as P_i and Q_i. We can see that P_i and Q_i are of degree one in terms of the v-variables.

Consider the system that contains the collection of all $v_{i_1,\cdots,i_n}$'s. Notice that

$$\begin{aligned} v'_{i_1,\cdots,i_k,\cdots,i_n} &= \sum_{k=1}^{n} x_1^{i_1} \cdots x_k^{i_k-1} \cdots x_n^{i_n} \cdot x_k' \\ &= \sum_{k=1}^{n} v_{i_1,\cdots,i_k-1,\cdots,i_n}(P_k - Q_k x_k) \\ &= \sum_{k=1}^{n} P_k v_{i_1,\cdots,i_k-1,\cdots,i_n} - \sum_{k=1}^{n} Q_k(x_k \cdot v_{i_1,\cdots,i_k-1,\cdots,i_n}) \\ &= \sum_{k=1}^{n} P_k v_{i_1,\cdots,i_k-1,\cdots,i_n} - \left(\sum_{k=1}^{n} Q_k\right) \cdot v_{i_1,\cdots,i_k,\cdots,i_n}. \end{aligned}$$

Note that the v-system is a CRN-implementable system with degree less than two. ◀

Proof of Theorem 13. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a quadratic CRN-implementable system. Then we have $x_i' = p_i - x_i q_i$ such that $p_i, q_i \in \mathbb{P}^+$ for $i = 1, \ldots, n$. Since the CRN is quadratic, it follows that $\deg(p_i) \leq 2$ and $\deg(q_i) \leq 1$ for $i = 1, \ldots, n$.

First, apply the λ-trick to shrink variables $x_2, \ldots, x_n$. Choose $0 \leq \lambda \leq 1$ and a constant $0 < c < 1$, such that $x_1 + \lambda(x_2 + \cdots + x_n) < 1 - c$. Then perform the change of variables: $x_2 \leftarrow \lambda x_2, \cdots, x_n \leftarrow \lambda x_n$.

Next, we introduce a new variable x_0 such that $\sum_{i=0}^{n} x_i = 1$. Use this summation and the "one-trick" to ensure that every term in x_i' is quadratic for $i = 1, \ldots, n$. Once each $x_1', \ldots, x_n'$ is a quadratic form, we signify these changes by writing $x_i' = P_i - x_i Q_i$ where $P_i, Q_i \in \mathbb{P}^+$. Note that P_i is a quadratic form and Q_i is a linear form.

We can now perform time dilation with x_0. As a result, we have the cubic form system,

$$\begin{cases} x_i' = (P_i - Q_i x_i) \cdot x_0, & \text{for } i = 1, \ldots, n \\ x_0' = -\sum_{i=1}^{n} x_i' \end{cases}$$

where we let $x_0(0) = 1$ and $x_i(0) = 0$ for $i = 1, \ldots, n$. Note that $P_i x_0, Q_i x_0 \in \mathbb{P}^+$ and x_i is in every negative term of x_i' for $i = 1, \ldots, n$. Some algebraic manipulation reveals that

$$x_0' = -\sum_{i=1}^{n} x_i' = -\sum_{i=1}^{n}(P_i - Q_i x_i) \cdot x_0 = \sum_{i=1}^{n}(Q_i x_i x_0 - P_i x_0).$$

Clearly, $Q_i x_0, P_i x_0 \in \mathbb{P}^+$ and x_0 is in every negative term of x_0'. Thus, $\mathbf{x}'$ is CRN implementable.

Since x_0 is a factor of every term in the cubic form $\mathbf{x}'$, x_i^3 is not a positive term of x_i' for $i = 1, \ldots, n$. For the case of x_0', recall that the positive terms of x_0' come from the negative terms of $x_1', \ldots, x_n'$. Given that every negative term of x_i' will have x_i as a factor, every positive term of x_0' will have at least one $x_i \neq x_0$ as a factor. Thus, x_0^3 cannot occur as a positive term in x_0'.

Hence, we have satisfied Corollary 4 to obtain a TPP-implementable cubic form system. ◀

Proof of Theorem 15. Let $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ be the TPP-implementable cubic form system from Stage 2. Then we have

$$\mathbf{x}' = \begin{cases} x_i' = (P_i - Q_i x_i) \cdot x_0, & \text{for } i = 1, \ldots, n \\ x_0' = -\sum_{i=1}^{n} x_i' \end{cases}$$

where x_1 computes some α and the system is initialized such that $x_0(0) = 1$ and 0 otherwise.

We construct our proof based on the "one trick" given by Observation 7,

$$\sum_{i=0}^{n} x_i \cdot \sum_{j=0}^{n} x_j = 1 \cdot 1 = 1.$$

We begin by defining a new variable vector, $\mathbf{z}$. Introduce new variables $z_{i,j} = x_i x_j$ for all $(x_i, x_j) \in \mathbf{x} \times \mathbf{x}$. To compute α, define the marked states as $M_z = \{z_{1,j} \mid 0 \leq j \leq n\}$. Given $\mathbf{x}(0)$, let $z_{0,0}(0) = 1$ and 0 otherwise.

Now, we find $\mathbf{z}'$. For all $z_{i,j} \in \mathbf{z}$ we have $z_{i,j}' = (x_i x_j)' = x_i' x_j + x_i x_j'$, where $x_i', x_j' \in \mathbf{x}'$. Recall that $\mathbf{x}'$ consists of cubic forms. Therefore, $x_i' x_j + x_i x_j'$ is a quartic form in variable $\mathbf{x}$. It follows from a change of variables from $\mathbf{x}$ to $\mathbf{z}$ that $z_{i,j}'$ is a quadratic form. For each $z_{i,j} \in \mathbf{z}$, we check conditions (i) and (ii) of Theorem 3. There are three non-trivial cases.

Case 1: Assume $z_{i,j} = x_i x_j$ such that $i \neq 0$ and $j \neq 0$. Then by the chain rule and substitution of $\mathbf{x}'$, we have

$$z_{i,j}' = x_i' x_j + x_i x_j' = (x_0 P_i - x_i x_0 Q_i) \cdot x_j + x_i \cdot (x_0 P_j - x_j x_0 Q_j).$$

We then distribute and group terms by their sign to obtain:

$$z_{i,j}' = (x_j x_0 P_i + x_i x_0 P_j) - (x_i x_j x_0 Q_i + x_i x_j x_0 Q_j)$$

Note that we can factor $(x_i x_j)$ from every negative term,

$$z_{i,j}' = (x_j x_0 P_i + x_i x_0 P_j) - (x_i x_j)(x_0 Q_i + x_0 Q_j).$$

$$\begin{aligned} z_{i,j}' &= (x_j x_0 P_i + x_i x_0 P_j) - (x_i x_j)(x_0 Q_i + x_0 Q_j) \\ &= (z_{0,j} P_i + z_{0,i} P_j) - z_{i,j} \cdot (x_0 Q_i + x_0 Q_j) \end{aligned}$$

and the terms P_i, P_j, $x_0 Q_i$, and $x_0 Q_j$ can be further written into linear form of $\{z_{i,j}\}_{i,j}$'s. We will not repeat these argument in the rest of the proof, though.

The above implies $z_{i,j}$ is a factor of every negative term. Also, since x_0 can be factored from every positive term, we cannot have a positive $z_{i,j}^2$ (recall that i, j are nonzero). Thus, conditions (i) and (ii) are satisfied.

Case 2: Assume $z_{0,j}$ such that $j \neq 0$. From the chain rule and substituting equations from $\mathbf{x}'$, we have

$$z_{0,j}' = x_0' x_j + x_0 x_j' = \sum_{i=1}^{n} (x_i x_0 Q_i - x_0 P_i) \cdot x_j + x_0 \cdot (x_0 P_j - x_j x_0 Q_j)$$

Distribute and group terms by their sign to obtain,

$$z'_{0,j} = x_0^2 P_j + \sum_{i=1}^{n} x_i x_j x_0 Q_i - (x_j x_0^2 Q_j + \sum_{i=1}^{n} x_0 x_j P_i)$$

Note that we can factor $(x_0 x_j)$ from every negative term,

$$z'_{0,j} = x_0^2 P_j + \sum_{i=1}^{n} x_i x_j x_0 Q_i - (x_j x_0)(x_0 Q_j + \sum_{i=1}^{n} P_i)$$

which implies $z_{0,j}$ is a factor of every negative term. Suppose a positive term in $z'_{0,j}$ has the monomial $(x_0 x_j)^2$. By carefully choosing the change of variable (i.e., choose $z_{j,j} z_{0,0}$, not $z_{0,j}^2$), we satisfy condition (ii). If there is no positive term in $z'_{0,j}$ with the monomial $(x_0 x_j)^2$, then condition (ii) is met. Then, both conditions (i) and (ii) are met.

Case 3: Assume $z_{0,0}$. With the chain rule and substitution, we have

$$z'_{0,0} = 2x_0 \cdot x'_0 = 2x_0 \cdot \sum_{i=1}^{n} (x_0 x_i Q_i - x_0 P_i)$$

Distributing and grouping terms by their sign yields,

$$z'_{0,0} = \sum_{i=1}^{n} (2x_0^2 x_i Q_i) - \sum_{i=1}^{n} (2x_0^2 P_i)$$

Since x_0^2 can be factored from every negative term, condition (i) is satisfied. Since $i = 1, \ldots, n$, it follows that $x_i \neq x_0$. Thus, condition (ii) is satisfied.

In summary, we have defined the states $\mathbf{z}$, marked states $M_{\mathbf{z}}$, initial configuration $\mathbf{z}(0)$, and differential system $\mathbf{z}'$. Furthermore, for each $z'_{i,j} \in \mathbf{z}'$ we have established: each $z'_{i,j}$ is a quadratic form, $z'_{i,j}$ can be written to avoid a positive $z_{i,j}^2$ in $z'_{i,j}$, and the existence of $z_{i,j}$ in every negative term of $z'_{i,j}$. Hence, a TPP-implementable cubic form system can be transformed into a PP-implementable system that computes α. ◀

Proof of Theorem 16. The "only if" part is clear. Therefore, it suffices to show the "if" direction. We adopt the same symbols and notations as in Construction 1. Recall that we denote $f_i(\mathbf{x}) = \varepsilon(p_i - q_i x_i) + 2x_i \sum_{j=1}^{n} x_j$. Consider an arbitrary ordered pair (x_j, x_k). We need to show that in the construction:

1. The selection of ε satisfying the two conditions is always possible.
2. The rule set generated by the construction forms an PLPP. Specifically, for the ordered pair (x_j, x_k), we must show that $\sum_{i=1}^{n} \alpha_{i,j,k}$ =1 and $\alpha_{i,j,k} > 0$ for all i. That is, the $\alpha_{i,j,k}$'s form a discrete probability measure with j, k fixed.

We break the proof into two cases:

- Case $j \neq k$: The term $x_j x_k$ occurs in $f_i(\mathbf{x})$ from two sources.
  - Case 1: $x_j x_k \in \varepsilon(p_i - q_i x_i)$. The coefficients of such terms for all i must sum to zero, since the system is assumed to be PP-implemetable.
  - Case 2: $x_j x_k \in 2x_i(x_1 + \cdots + x_n)$. We have two sources: $x_j x_k$ occurs in $f_j(\mathbf{x})$ as a term in $2x_j(x_1 + \cdots x_k + \cdots + x_n)$ and similarly in $f_k(\mathbf{x})$ as a term in $2x_k(x_1 + \cdots x_j + \cdots + x_n)$.

  The sum of these coefficients is 4. Note that the coefficients are divided by 4 in the construction, which ensures the resulting sum of $\alpha_{i,j,k}$ is one. One can verify that the selection of ε is always possible under this case.

Case $j = k$: For term x_j^2, which occurs in $f_i(\mathbf{x})$, there are two subcases:

- $i \neq j$: Then $x_j^2 \in \varepsilon p_i$ and the coefficient must be positive.
- $i = j$: Then $x_j^2 = x_i^2 \in -\varepsilon q + 2x_i \sum_{k=0}^{n} x_k$ and coefficient is less than 2. ($x_j^2 = x_i^2$ can not occur in εp since the system is PP-implementable.)

Recall that when constructing the rule set, the coefficients are divided by 2. As a result, generated $\alpha_{i,j}$'s are positive and less than 1. Due to the system's conservative nature, the coefficients x_j^2 in $\varepsilon(p_i - q_i x_i)$ for i's in $\{1, \cdots, n\}$ will sum to zero. Since the $2x_j \sum_{k=1}^{n} x_k$ term introduces a $2x_j^2$ term, the resulting coefficient sum for x_j^2 is 2. In the construction, these coefficients are always divided by 2 to generate $\alpha_{i,j}$ in $f_i(\mathbf{x})$. Thus, the $\sum_{i=1}^{n} \alpha_{i,j} = 1$. One can then verify that the selection of ε is feasible.

In contrast, assume the system is not PP-implementable. Then there is an $x_i^2 \in \varepsilon p_i$ with a positive coefficient. Once combined with the term $2x_i^2 \in 2x_i \sum_{k=1}^{n} x_k$, the sum of coefficients will be greater than two. Then dividing by 2 will generate an $\alpha_{i,i} > 1$, which is not permitted by PLPPs. Hence, the selection of ε is not feasible in this case. ◀

Proof of Corollary 19. The first four numbers are GPAC/CRN computable (see [22]). Note that all real-time computable numbers by CRNs in [23] are computable in this paper. One just need to disregard the real-time constraint. Then by our main theorem, they are also LPP computable. It suffices to show that Catalan's constant is GPAC computable. We use the formula $G = \frac{1}{2}\int_0^\infty \frac{t}{\cosh t} dt$. We have

$$\begin{aligned} G &= \int_0^\infty \frac{t}{e^t + e^{-t}} dt, \quad \text{since } cosh(t) = \frac{e^t + e^{-t}}{2} \\ &= \int_0^\infty (t \cdot e^{-t}) \cdot \frac{1}{1 + e^{-2t}} dt. \end{aligned}$$

We let

$$R = t \cdot e^{-t}, \quad E = e^{-t}, \quad V = \frac{e^{-2t}}{1 + e^{-2t}},$$

and

$$G(t) = \int_0^t (te^{-t}) \cdot \frac{1}{1 + e^{-2t}} dt = \int_0^t R \cdot (1 - V) dt.$$

Taking derivatives against t, we get

$$\begin{cases} G' = R(1 - V) \\ R' = E - R \\ E' = -E \\ V' = (1 - V)^2 \cdot (-2E^2), \end{cases} \qquad \begin{cases} G(0) = 0 \\ R(0) = 0 \\ E(0) = 1 \\ V(0) = \frac{1}{2}. \end{cases}$$

Therefore Catalan's constant can be computed by the above PIVP. Note that the system's non-zero initial values can be transformed into a system with zeroed initial values ([23], Theorem 3). Hence by our Main Theorem, Catalan's constant is LPP computable. ◀

The Structural Power of Reconfigurable Circuits in the Amoebot Model

Andreas Padalkin
Universität Paderborn, Germany

Christian Scheideler
Universität Paderborn, Germany

Daniel Warner
Universität Paderborn, Germany

Abstract

The *amoebot model* [Derakhshandeh et al., SPAA 2014] has been proposed as a model for programmable matter consisting of tiny, robotic elements called *amoebots*. We consider the *reconfigurable circuit extension* [Feldmann et al., JCB 2022] of the geometric (variant of the) amoebot model that allows the amoebot structure to interconnect amoebots by so-called *circuits*. A circuit permits the instantaneous transmission of signals between the connected amoebots. In this paper, we examine the structural power of the reconfigurable circuits.

We start with some fundamental problems like the *stripe computation problem* where, given any connected amoebot structure S, an amoebot u in S, and some axis X, all amoebots belonging to axis X through u have to be identified. Second, we consider the *global maximum problem*, which identifies an amoebot at the highest possible position with respect to some direction in some given amoebot (sub)structure. A solution to this problem can then be used to solve the *skeleton problem*, where a (not necessarily simple) cycle of amoebots has to be found in the given amoebot structure which contains all boundary amoebots. A canonical solution to that problem can then be used to come up with a canonical path, which provides a unique characterization of the shape of the given amoebot structure. Constructing canonical paths for different directions will then allow the amoebots to set up a spanning tree and to check symmetry properties of the given amoebot structure.

The problems are important for a number of applications like rapid shape transformation, energy dissemination, and structural monitoring. Interestingly, the reconfigurable circuit extension allows polylogarithmic-time solutions to all of these problems.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Computational geometry

Keywords and phrases progammable matter, amoebot model, reconfigurable circuits, spanning tree, symmetry detection

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.8

Funding This work has been supported by the DFG Project SCHE 1592/6-1 (PROGMATTER).

1 Introduction

The *amoebot model* [6, 4] is a well-studied model for programmable matter [15] – a substance that can be programmed to change its physical properties, like its shape and density. In the geometric variant of this model, the substance (called the *amoebot structure*) consists of simple particles (called *amoebots*) that are placed on the infinite triangular grid graph and are capable of local movements through *expansions* and *contractions*.

Inspired by the *nervous* and *muscular system*, Feldmann et al. [8] introduced a *reconfigurable circuit extension* to the amoebot model with the goal of significantly accelerating fundamental problems like leader election and shape transformation. As a first step, they

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham; Article No. 8; pp. 8:1–8:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

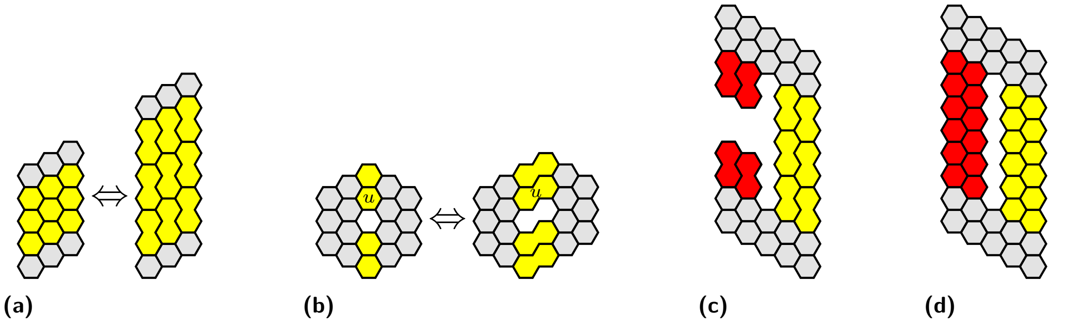


Figure 1 Feldmann et al. [8] have proposed joint movements to the amoebot model where an expanding amoebot is capable of pushing other amoebots away from it, and a contracting amoebot is capable of pulling other amoebots towards it. (a) and (b) show a joint expansion (from left to right) resp. a joint contraction (from right to left) of the yellow amoebots. The left side of (b) shows stripe $\mathrm{A}(S, u, N)$ (see Section 1.3). The stripe can expand without causing any conflicts. (c) and (d) show exemplary conflicts. (c) If the yellow amoebots contract, the red amoebots will collide. In order to avoid the collision, the red amoebots have to contract as well. (d) If the yellow amoebots expand, the amoebot structure will tear apart. In order to maintain all connections, the red amoebots have to expand as well.

showed that leader election, consensus, compass alignment, chirality agreement, and various shape recognition problems can be solved in at most $O(\log n)$ time. This paper continues this line of work by considering a number of additional problems:

First, we consider the *stripe computation problem* where, given any connected amoebot structure S, an amoebot u in S, and some axis X, all amoebots belonging to axis X through u have to be identified. Second, we consider the *global maximum problem*, which identifies an amoebot at the highest possible position with respect to some direction in some given amoebot (sub)structure. A solution to this problem can then be used to solve the *skeleton problem*, where a (not necessarily simple) cycle of amoebots has to be found in the given amoebot structure which contains all boundary amoebots. A canonical solution to that problem can then be used to come up with a canonical path, which provides a unique characterization of the shape of the given amoebot structure. Constructing canonical paths for different directions will then allow the amoebots to set up a spanning tree and to check symmetry properties of the given amoebot structure.

The problems have a number of important applications. The stripe computation problem is important to avoid conflicts in joint amoebot contractions and expansions (see Figure 1), which is critical for rapid shape transformation. A spanning tree is an important step towards energy distribution from amoebots with access to energy to amoebots without such access [5], and canonical skeleton paths as well as symmetry checks are important for structural monitoring and repair.

1.1 Geometric Amoebot Model

In the *geometric amoebot model* [4], a set of n amoebots is placed on the infinite regular triangular grid graph $G_\Delta = (V, E)$ (see Figure 2a). An amoebot is an anonymous, randomized finite state machine that either occupies one or two adjacent nodes of G_Δ, and every node of G_Δ is occupied by at most one amoebot. If an amoebot occupies just one node, it is called *contracted* and otherwise *expanded*. Two amoebots that occupy adjacent nodes in G_Δ are called *neighbors*. Amoebots are able to move through *contractions* and *expansions*. However, since our algorithms do not make use of movements, we omit further details and refer to [4] for more information.

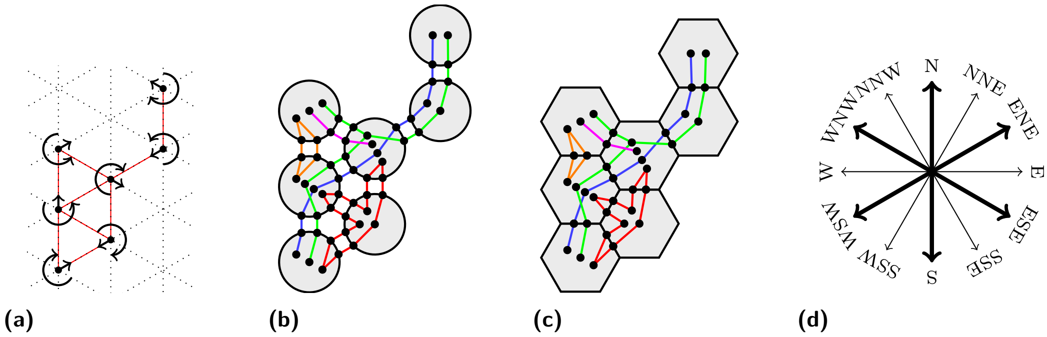


Figure 2 (a) shows an amoebot structure S. The dotted lines indicate the triangular grid G_Δ. The nodes indicate the amoebots. The arrows show their chirality and compass orientation. The red edges indicate the graph G_S. (b) and (c) show an amoebot structure with $k = 2$ external links between neighboring amoebots. The amoebots are shown in gray. The nodes on the boundary are the pins, and the ones within the amoebots the partition sets. An edge between a partition set Q and a pin p implies $p \in Q$. Each color indicates another circuit. (a) and (b) are taken from [8]. (d) shows the cardinal directions. The thick arrows indicate the cardinal directions along the main axes, and the thin ones the cardinal directions perpendicular to the main axes.

Each amoebot has a compass orientation (it defines one of its incident edges as the northern direction) and a chirality (a sense of clockwise or counterclockwise rotation) that it can maintain as it moves, but initially the amoebots might not agree on their compass orientation and chirality. In this paper, we assume that all amoebots share a common compass orientation and chirality. This is reasonable since Feldmann et al. [8] showed that all amoebots are able to come to an agreement within the considered extension (see Section 1.4).

Let the *amoebot structure* $S \subseteq V$ be the set of nodes occupied by the amoebots. By abuse of notation, we identify amoebots with their nodes. We say that S is *connected* iff G_S is connected, where $G_S = G_\Delta|_S$ is the graph induced by S. In this paper, we assume that initially, S is connected and all amoebots are contracted. Also, we assume the fully synchronous activation model, i.e., the time is divided into synchronous rounds, and every amoebot is active in each round. On activation, each amoebot may perform a movement and update its state as a function of its previous state. However, if an amoebot fails to perform its movement, it remains in its previous state. The time complexity of an algorithm is measured by the number of synchronized rounds required by it.

1.2 Reconfigurable Circuit Extension

In the *reconfigurable circuit extension* [8], each edge between two neighboring amoebots u and v is replaced by k edges called *external links* with endpoints called *pins*, for some constant $k \geq 1$ that is the same for all amoebots. For each of these links, one pin is owned by u while the other pin is owned by v. In this paper, we assume that neighboring amoebots have a common labeling of their incident external links.

Each amoebot u *partitions* its *pin set* $P(u)$ into a collection $\mathcal{Q}(u)$ of pairwise disjoint subsets such that the union equals the pin set, i.e., $P(u) = \bigcup_{Q \in \mathcal{Q}(u)} Q$. We call $\mathcal{Q}(u)$ the *pin configuration* of u and $Q \in \mathcal{Q}(u)$ a *partition set* of u. Let $\mathcal{Q} = \bigcup_{u \in S} \mathcal{Q}(u)$ be the collection of all partition sets in the system. Two partition sets are *connected* iff there is at least one external link between those sets. Let L be the set of all connections between the partition sets in the system. Then, we call $H = (\mathcal{Q}, L)$ the *pin configuration* of the system and any connected component C of H a *circuit* (see Figure 2b). Note that if each partition set of $\mathcal{Q}$

is a *singleton*, i.e., a set with exactly one element, then every circuit of H just connects two neighboring amoebots. However, an external link between the neighboring amoebots u and v can only be maintained as long as both u and v occupy the incident nodes. Whenever two amoebots disconnect, the corresponding external links and their pins are removed from the system. An amoebot is part of a circuit iff the circuit contains at least one of its partition sets. A priori, an amoebot u may not know whether two of its partition sets belong to the same circuit or not since initially it only knows $\mathcal{Q}(u)$.

Each amoebot u can send a primitive signal (a *beep*) via any of its partition sets $Q \in \mathcal{Q}(u)$ that is received by all partition sets of the circuit containing Q at the beginning of the next round. The amoebots are able to distinguish between beeps arriving at different partition sets. More specifically, an amoebot receives a beep at partition set Q if at least one amoebot sends a beep on the circuit belonging to Q, but the amoebots neither know the origin of the signal nor the number of origins. Note that beeps are enough to send whole messages over time, especially between adjacent amoebots. We modify an activation of an amoebot as follows. As a function of its previous state and the beeps received in the previous round, each amoebot may perform a movement, update its state, reconfigure its pin configuration, and activate an arbitrary number of its partition sets. The beeps are propagated on the updated pin configurations. If an amoebot fails to perform its movement, it remains in its previous state and pin configuration, and does not beep on any of its partition sets.

In this paper, we will utilize the dual graph of the triangular grid graph, i.e., a hexagonal tesselation, to visualize amoebot structures (see Figure 2c). Thereby, we reduce each external link to a single pin. Furthermore, in order to improve the comparability of circuit configurations, we add pins to each side of the hexagon.

1.3 Problem Statement and Our Contribution

Let $D_m = \{N, ENE, ESE, S, WSW, WNW\}$ be the set of all cardinal directions along the axes of G_Δ, and $D_p = \{E, SSE, SSW, W, NNW, NNE\}$ the set of all cardinal directions perpendicular to the axes of G_Δ (see Figure 2d). In the following, we state the considered problems.

First, we consider the *stripe computation problem.* Let $\mathrm{X}(v, d) \subseteq V$ denote the nodes of G_Δ that lie on the axis through the node $v \in V$ into the cardinal direction $d \in D_m \cup D_p$. For $R \subseteq V$, we call the set $\mathrm{A}(R, v, d) = R \cap \mathrm{X}(v, d)$ a *stripe* of R (see Figure 1b). Note that a stripe is not necessarily connected. Let an amoebot $u \in S$ and a cardinal direction $d \in D_m \cup D_p$ be given, i.e., each amoebot $v \in S$ knows the cardinal direction d and whether $v = u$. The goal of each amoebot $v \in S$ is to determine whether $v \in \mathrm{A}(S, u, d)$. Our *stripe algorithm* solves the stripe computation problem after $O(\log n)$ rounds.

Second, we consider the *global maxima problem.* Let a cardinal direction $d \in D_m \cup D_p$ and a non-empty set $R \subseteq S$ be given, i.e., each amoebot $v \in S$ knows the direction d and whether $v \in R$. The goal of each amoebot $v \in S$ is to determine whether $v \in \operatorname{argmin}_{w \in R} \mathrm{f}_d(R, w)$ where $\mathrm{f}_d(R, w)$ denotes the number of amoebots in R that lie in direction d from amoebot w. We call $\operatorname{argmin}_{w \in R} \mathrm{f}_d(R, w)$ the set of global maxima of R with respect to d. Our *global maxima algorithm* solves the global maxima problem after $O(\log^2 n)$ rounds w.h.p.[1]

Third, we consider the *(canonical) skeleton problem.* An amoebot u is a *boundary amoebot* iff it is adjacent to an unoccupied node in $V \setminus S$. Otherwise, we call u an *inner amoebot.* A (potentially non-simple) cycle of amoebots is a skeleton iff the cycle contains all boundary

[1] An event holds *with high probability (w.h.p.)* if it holds with probability at least $1 - 1/n^c$ where the constant c can be made arbitrarily large.

amoebots in S. Note that the skeleton may contain inner amoebots. An amoebot structure computes a skeleton C iff each amoebot knows its predecessor and successor for each of its occurrences in C. The goal of the skeleton problem is to compute an arbitrary skeleton.

Since skeletons are not unique, we define a *canonical skeleton* with respect to a cardinal direction $d \in D_m \cup D_p$ and a sign $s \in \{+, -\}$ (abbreviated as (d, s)-skeleton). We defer the definition to Section 4.1. Let a cardinal direction $d \in D_m \cup D_p$ and a sign s be given, i.e., each amoebot $v \in S$ knows the cardinal direction d and the sign s. The goal of the canonical skeleton problem is to compute the canonical skeleton. Our *canonical skeleton algorithm* solves the (canonical) skeleton problem after $O(\log^2 n)$ rounds w.h.p.

Our algorithms for the remaining problems are based on skeletons. However, they split the skeletons into paths that we call *skeleton paths*. For the canonical skeletons, we define a *canonical skeleton path* by specifying a splitting point. Our canonical skeleton algorithm determines this point in parallel to the computation of the canonical skeleton.

Fourth, we consider the *spanning tree problem*. A *tree* is a cycle-free and connected graph. A *spanning tree* of an amoebot structure S is a tree $T = (S, E_T)$ with $E_T \subseteq E$. An amoebot structure computes a spanning tree T if each amoebot $u \in S$ knows whether $\{u, v\} \in E_T$ for each neighbor $v \in \mathrm{N}(u)$. The goal of the spanning tree problem is to compute an arbitrary spanning tree. Our *spanning tree algorithm* solves the spanning tree problem after $O(\log^2 n)$ rounds w.h.p.

Finally, we consider the *symmetry detection problem*. The goal of that problem is to determine whether the amoebot structure features rotational or reflection symmetries. Our *symmetry detection algorithm* solves the the problem after $O(\log^5 n)$ rounds w.h.p.

1.4 Related Work

The reconfigurable circuit extension was introduced by Feldmann et al. [8]. Note that the amoebot model [1] and its circuit extension [11, 12, 13, 14] can in principle be realized.

Feldmann et al. [8] have proposed solutions for *leader election* (see Section 2), *consensus*, *compass alignment*, *chirality agreement*, and various *shape recognition* problems. Both, the alignment of the compasses and the agreement on a chirality requires $O(\log n)$ w.h.p. This makes our assumption of a common compass orientation and chirality reasonable.

To our knowledge the stripe computation, global maxima, (canonical) skeleton, and symmetry detection problem have not been considered within the standard amoebot model. However, regarding the global maxima problem, Daymude et al. [2] have considered the related problem of determining the dimensions of an object (a finite, connected, static set of nodes) in order to solve various *convex hull problems*. Their approach can be easily adjusted to compute the global maxima of the amoebot structure. However, it requires $O(n)$ rounds.

The spanning tree problem is widely studied in the distributed algorithms community, e.g., [10] (also see the cited papers within the reference). The *spanning tree primitive* is one of the most used techniques to move amoebots, e.g., [3, 2, 7, 9]. Beyond that, spanning trees were applied to distribute energy [5]. However, the construction requires $\Omega(D)$ rounds where D is the diameter of the amoebot structure (e.g., see [3]). Since $D = \Omega(\sqrt{n})$, our solution is a significant improvement.

2 Preliminaries

In this section, we enumerate important primitives given in previous papers.

Global Circuit. If each amoebot partitions its pins into one partition set, we obtain a single circuit that interconnects all amoebots. We call this circuit the *global circuit* [8].

Leader Election. We make use of a generalized version of the leader election algorithm proposed by Feldmann et al. [8]:

▶ **Theorem 1.** *Let $C_1, \dots, C_m$ be sets of candidates. For each $i \in \{1, \dots, m\}$, let $\mathcal{C}_i$ be the circuit that connects all candidates of set C_i. Let $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ hold for all $i \neq j$. An amoebot structure elects a leader from each set of candidates after $\Theta(\log n)$ rounds w.h.p.*

In the classical leader election problem, the amoebot structure S has to elect a leader only from the set $C_1 = S$.

Chains. We call an ordered sequence $C = (u_0, \dots, u_{m-1})$ of m amoebots a *chain* iff (i) all subsequent amoebots u_i, u_{i+1} are neighbors, (ii) each amoebot in C except u_0 knows its predecessor, and (iii) each amoebot in C except u_{m-1} knows its successor.

Boundary Sets. We adopt the definition for the *boundary sets* from [7]: Consider $G_{V \setminus S} = G_\Delta|_{V \setminus S}$. The connected components of $G_{V \setminus S}$ are called empty regions. The number of empty regions is finite since S is finite. Let $R_1, \dots, R_m$ denote the empty regions. For $i \in \{1, \dots, m\}$, the boundary set B_i is the neighborhood of R_i in S, i.e., $B_i = \{u \in S \mid \exists v \in R_i : \{u, v\} \in E\}$. There is exactly one infinite empty region since S is finite. We call the corresponding boundary set the *outer boundary set*, and the others *inner boundary sets.* Naturally, each boundary set can be represented as a (potentially non-simple) cycle.

In order to distinguish inner and outer boundaries, we apply the *inner outer boundary test* by Derakhshandeh et al. [7]. They accumulate the angles of the turns while traversing the cycle of the boundary set once. An outer boundary set results in a sum of 360°, and an inner boundary set results in a value of −360°. Note that it is sufficient to count the turns by 60° modulo 5. This allows us to accumulate the sum with constant memory. However, the traversing requires $O(n)$ rounds. We accelerate the accumulation by the following result by Feldmann et al. [8]:

▶ **Theorem 2.** *Let $C = (v_0, \dots, v_{m-1})$ be a chain within the amoebot structure where $m \in \mathbb{N}$ denotes the length of the chain. Let $k \in \mathbb{N}$ be constant. Let $x_i \in \{0, \dots, k-1\}$ for all $i \in \{0, \dots, m-1\}$. Suppose that for each $i \in \{0, \dots, m-1\}$, amoebot v_i knows the value x_i. Then, the chain computes $x = \sum_{i \in \{0, \dots, m-1\}} x_i \mod k$ after $O(\log m)$ rounds.*

▶ **Corollary 3.** *A boundary set can determine whether it is an inner boundary set or the outer boundary set within $O(\log n)$ rounds w.h.p.*

Synchronization of Procedures. Sometimes, we want to execute a procedure on different subsets in parallel, e.g., we apply the inner outer boundary test on all boundary sets at once. The execution of the the same procedure may take different amounts of rounds for each subset. The amoebots of a single subset are not able to decide when all subsets have terminated. In order to synchronize the amoebot structure with respect to the procedures, the amoebots periodically establish the global circuit. The amoebots of each subset that has not yet terminated beep on that circuit. The amoebot structure may proceed to the next procedure once the global circuit was not activated.

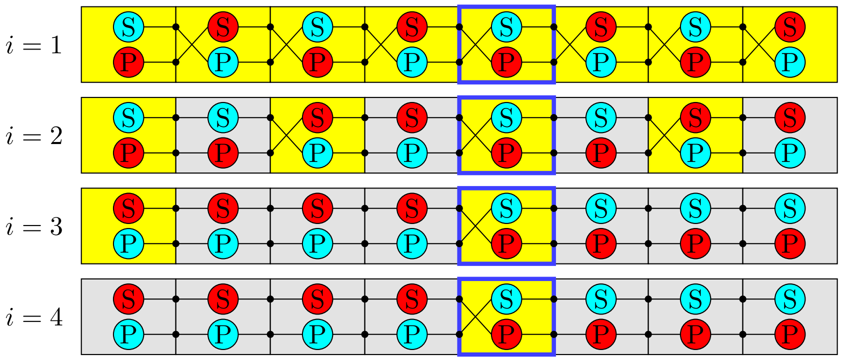


Figure 3 PASC algorithm. Each figure shows the chain at the beginning of the i-th iteration. The circles indicate the primary (P) and secondary (S) partition sets. The blue bordered amoebot denotes the reference amoebot u_r. Yellow amoebots are active, and gray amoebots are passive. All primary and secondary partition sets that are part of the primary resp. secondary circuit of u_r are depicted in red resp. cyan.

3 Computing Identifiers

In this section, we assign *identifiers* in $\mathbb{Z}$ to the amoebots. Let $(x_{k-1}, \ldots, x_0)$ denote the *two's complement representation* of $-x_{k-1} \cdot 2^{k-1} + \sum_{i=0}^{k-2} x_i \cdot 2^i$. By abuse of notation, we identify the identifiers with their two's complement representation. Each amoebot computes a two's complement representation of its identifier. In the first subsection, we compute successive identifiers along chains. In the second subsection, we compute spatial identifiers with respect to a cardinal direction. In the third subsection, we show two applications for the identifiers.

3.1 Successive Identifiers along the Chain

In this section, we compute successive identifiers along a chain with respect to an amoebot that we call the *reference amoebot*. Let $C = (u_0, \ldots, u_{m-1})$ be a chain of amoebots. Let u_r be an arbitrary reference amoebot within the chain, e.g., chosen by position (for example $r = 0$), or by a leader election. We assign identifiers id_{C,u_r} according to the following rules:

$$\mathrm{id}_{C,u_r}(u_r) = 0 \tag{1}$$

$$\mathrm{id}_{C,u_r}(u_{i+1}) = \mathrm{id}_{C,u_r}(u_i) + 1 \text{ for } 1 \leq i < m \tag{2}$$

Note that $\mathrm{id}_{C,u_r}(u_i) = i - r$.

In order to compute the identifiers, we utilize a procedure on the chain of amoebots proposed by Feldmann et al. [8] that we henceforth refer to as the *primary and secondary circuit algorithm* (PASC algorithm). Originally, the algorithm has been used as a subroutine for Theorem 2.

In the following, we explain how the PASC algorithm works (see Figure 3). So, let $C = (u_0, \ldots, u_{m-1})$ be a chain of m amoebots. If an amoebot occurs multiple (but constantly many) times, it operates each position independently of each other which is possible by using sufficiently many pins. Each amoebot is either *active* or *passive*. Initially, each amoebot is active. The algorithm iteratively transforms active amoebots into passive amoebots while keeping amoebot u_r active.

At the beginning of an iteration, the amoebots establish two circuits as follows. Each amoebot has two partition sets that we call the *primary and secondary partition set.* We connect the primary and secondary partition sets by the following two rules[2] (see Figure 3): (i) If an amoebot is active, we connect its primary partition set to the secondary partition set of its predecessor, and its secondary partition set to the primary partition set of its predecessor. (ii) If an amoebot is passive, we connect its primary partition set to the primary partition set of its predecessor, and its secondary partition set to the secondary partition set of its predecessor.

We obtain two circuits through all amoebots (see Figure 3). Each amoebot defines the circuit containing its primary partition set as its *primary circuit*, and the circuit containing its secondary partition set as its *secondary circuit.* Clearly, we obtain two disjoint circuits along the chain. We refer to [8] for a detailed construction.

After establishing these circuits, the iteration utilizes two rounds. In the first round, amoebot u_r activates its primary circuit. Each active amoebot that has received the beep on its secondary circuit beeps in the second round on its secondary circuit. These amoebots become passive amoebots in the next iteration. The algorithm terminates when the second round is silent. At this point, amoebot u_r is the remaining active amoebot. Feldmann et al. [8] have proven the following lemma:

▶ **Lemma 4.** *The PASC algorithm terminates in* $\lceil \log m \rceil$ *iterations, resp.* $O(\log m)$ *rounds.*

We obtain the identifiers from the PASC algorithm as follows. Let $k = \lceil \log m \rceil$ be the number of iterations. For $0 \leq i < k$, let r_i be the first round of the $(i+1)$-st iteration. Note that in each of these rounds, each amoebot of the chain receives a beep either on its primary circuit or its secondary circuit. Hence, in round r_i, amoebot x interprets a beep on the primary circuit as $x_i = 0$ and a beep on the secondary circuit as $x_i = 1$.

▶ **Lemma 5.** *The PASC algorithm computes* id_{C,u_r}.

3.2 Spatial Identifiers

In this section, we compute identifiers relative to the spatial positions of the amoebots with respect to a cardinal direction $d \in D_m \cup D_p$ and a reference amoebot $u_r \in S$. Let d' denote the direction obtained if we rotate d by 90° counterclockwise, e.g., $d' = N$ for $d = E$. First, consider $d \in D_p$ (see Figure 4a). Let $\mathcal{A}_d = \{\mathrm{A}(S, v, d') \mid v \in S\}$ denote a set of stripes. We first assign identifiers to $\mathcal{A}_d$. Afterwards, we extend these identifiers to the nodes in S.

Observe that $\mathcal{A}_d$ partitions S into disjoint stripes. These stripes form a chain $\mathcal{C}_d$ if we think of the stripes as nodes such that two nodes are adjacent if the corresponding stripes are neighbors (see Figure 4a). The order of the chain is given by the cardinal direction d: The successor of a stripe is the neighboring stripe in direction d. Let $\mathrm{succ}(A)$ denote the succeeding stripe of stripe A. Let $A_r = \mathrm{A}(S, u_r, d')$. We assign identifiers $\mathrm{id}_{\mathcal{C}_d,A_r}$ according to the following two rules:

$$\mathrm{id}_{\mathcal{C}_d,A_r}(A_r) = 0$$
$$\mathrm{id}_{\mathcal{C}_d,A_r}(\mathrm{succ}(A)) = \mathrm{id}_{\mathcal{C}_d,A_r}(A) + 1 \text{ for all } A \in \mathcal{A}_d$$

Finally, we define $\mathrm{id}_{d,u_r}(v) = \mathrm{id}_{\mathcal{C}_d,A_r}(\mathrm{A}(S, v, d'))$ for all nodes $v \in S$.

[2] In comparison to [8], we have adjusted the rules by mirroring the connections.

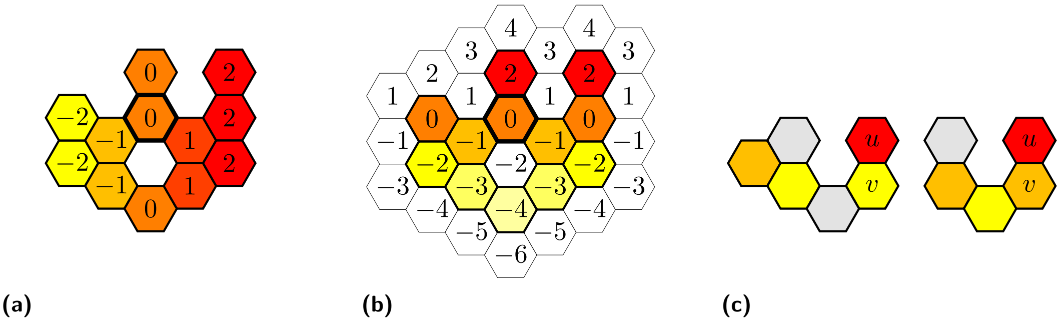


Figure 4 Spatial identifiers. (a) show the spatial identifiers with respect to $d = E$, and (b) with respect to $d = N$. Each color indicates a stripe (in S). The white hexagons indicate the neighborhood of S. The thick boundary indicates the reference amoebot u_r. (c) If we do not include the neighborhood of S, then amoebot u is unable to determine whether amoebot v belongs to the preceding stripe (orange) or the stripe preceding its preceding stripe (yellow).

In order to compute the identifiers, we transfer the concept of primary and secondary circuits from a chain of amoebots to a chain of stripes (compare with Section 3.1): From a global perspective, each stripe knows its predecessor and its successor, is either active or passive, and has a primary and secondary partition set. The primary and secondary partition sets are connected by the following rules: If a stripe is active, its primary partition set is connected to the secondary partition set of its predecessor, and its secondary partition set is connected to the primary partition set of its predecessor. If a stripe is passive, its primary partition set is connected to the primary partition set of its predecessor, and its secondary partition set is connected to the secondary partition set of its predecessor. There are no further connections. In order to keep the amoebots within a stripe synchronized, we additionally require that each amoebot has access to the primary and secondary partition set of its stripe. In the following, we explain how to construct the circuits that satisfy the aforementioned properties.

Note that the neighborhood of each amoebot only contains amoebots of the same stripe, the preceding stripe, and the succeeding stripe. Since we assume common compass orientation and chirality, each amoebot is able to determine to which stripe each neighbor belongs. We now define pin configurations that satisfy the aforementioned properties. Each pin configuration has two partition sets that we call the primary and secondary partition set. We connect the primary and secondary partition sets of an amoebot to the primary and secondary partition sets of adjacent amoebots such that the aforementioned properties are reflected locally. For example, consider two adjacent amoebots u and v such that u belongs to an active stripe and v belongs to u's preceding stripe. We connect u's primary partition set to v's secondary partition set, and u's secondary partition set to v's primary partition set. Since we assign the same identifiers to amoebots of the same stripe, we connect their primary and secondary partition sets, respectively. We obtain two pin configurations: one for amoebots of active stripes and one for amoebots of passive stripes (see Figure 5).

▶ **Lemma 6.** *The construction satisfies the aforementioned properties.*

Now, we simply apply the PASC algorithm on the chain of stripes to compute $\mathrm{id}_{\mathcal{C}_d,A_r}$, i.e., each amoebot $v \in S$ computes $\mathrm{id}_{\mathcal{C}_d,A_r}(\mathrm{A}(S,v,d'))$ that equals $\mathrm{id}_{d,u_r}(v)$ by definition.

▶ **Lemma 7.** *The PASC algorithm computes id_{d,u_r} for $d \in D_p$.*

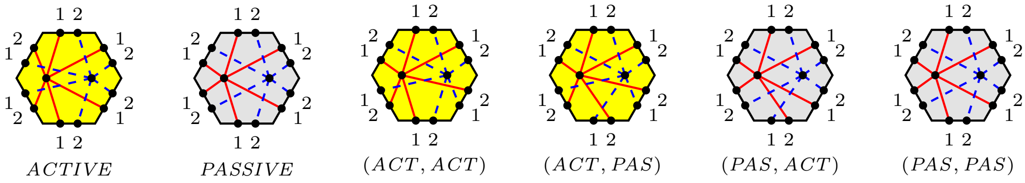


Figure 5 Utilized pin configurations. We utilize the former two pin configurations for $d = E$. We utilize the latter four pin configurations for $d = N$. The first argument denotes the state of the amoebot's stripe and the second argument denotes the state of the preceding stripe, respectively.

Next, consider $d \in D_m$. We discuss the necessary modifications in comparison to $d \in D_p$. Note that an amoebot is unable to locally determine to which stripe its neighbors belong (see Figure 4c). We therefore perform the PASC algorithm on the union of the amoebot structure and its neighborhood (see Figure 4b). Note that each neighbor of an amoebot $v \in S$ belongs either to one of the two preceding stripes or to one of the two succeeding stripe.

Each amoebot tracks when the stripes of its neighbors become passive as follows. Recall that all stripes are initially active. Suppose that each amoebot $v \in S$ knows whether (its stripe and) the stripes of its neighbors are active or passive at the beginning of an iteration of the PASC algorithm. Hence, v also knows how these stripes are interconnected. Thus, v can conclude from the signal it receives on the signals received by the stripes of its neighbors and with that whether these become passive.

There are two reasons for the tracking. First, some of the stripes are not occupied by any amoebots. The tracking allows each amoebot $v \in S$ to activate the correct circuits for each of its neighbors. Second, the connections between an amoebot and its neighbor of the stripe preceding its preceding stripe depends on the states of its stripe and its preceding stripe. We obtain four pin configurations (see Figure 5).

▶ **Lemma 8.** *The PASC algorithm computes* id_{d,u_r} *for* $d \in D_m$*.*

Note that we can generalize this technique to arbitrary directions as long as the identifiers within a neighborhood only differ by some constant.

3.3 Applications

We now consider two applications for the identifiers, namely the stripe problem and the global maxima problem.

First, consider the stripe problem. Our stripe algorithm simply executes the PASC algorithm with u as the reference amoebot, i.e., $u_r = u$. Note that this sets the identifier of u to 0, i.e., $\mathrm{id}_{d,u}(u) = 0$. By construction, $\mathrm{id}_{d,u}(v) = \mathrm{id}_{d,u}(u) = 0$ holds for all $v \in \mathrm{A}(S, u, d)$. We obtain the following theorem.

▶ **Theorem 9.** *The stripe algorithm solves the stripe computation problem in* $O(\log n)$ *rounds.*

Next, consider the global maxima problem. Recall that we compute the global maxima of a set $R \subseteq S$ (see Section 1.3). By construction, $\mathrm{argmin}_{w \in R}\, \mathrm{f}_d(R, w) = \mathrm{argmax}_{w \in R}\, \mathrm{id}_{d,u_r}(w)$ holds for any reference amoebot u_r. The idea of our global maxima algorithm is therefore to execute the PASC algorithm and to determine the highest identifier. By choosing an $u \in R$ as the reference amoebot, i.e., $u_r = u$, we ensure that the maximal identifier is non-negative. In order to determine the maximum of non-negative numbers, we apply the consensus algorithm by Feldmann et al. [8] that agrees on the highest input value. First, the amoebot structure establishes the global circuit (see Section 2). Each amoebot $v \in R$ with $\mathrm{id}(v) \geq 0$ transmits its identifier starting from the most significant bit. If a transmitting amoebot observes a beep in a round it does not beep, it stops its transmission. Only an amoebot with the highest identifier is able to transmit its identifier until the end.

However, since each amoebot can only store a constant section of its identifier, and since the PASC algorithm provides the identifiers from the least significant bit to the most significant bit, we have to partially recompute the identifiers after each bit. In order to identify the correct bit, we simply use two binary counters where the first counter indicates the current bit, and the second counter the current iteration of the PASC algorithm. These can be realized along a chain such that with the help of circuits, the incrementation, the decrementation, and the comparison only require $O(1)$ rounds. Using a chain along the outer boundary set ensures a sufficient length of the counters. We obtain the following theorem.

▶ **Theorem 10.** *The global maxima algorithm computes the global maxima in $O(\log^2 n)$ rounds w.h.p.*

4 Skeletons

This section deals with (canonical) skeletons. In the first subsection, we canonicalize and construct the skeletons. In the second and third subsection, we show two applications for skeletons by showing how to construct spanning trees and how to detect symmetries.

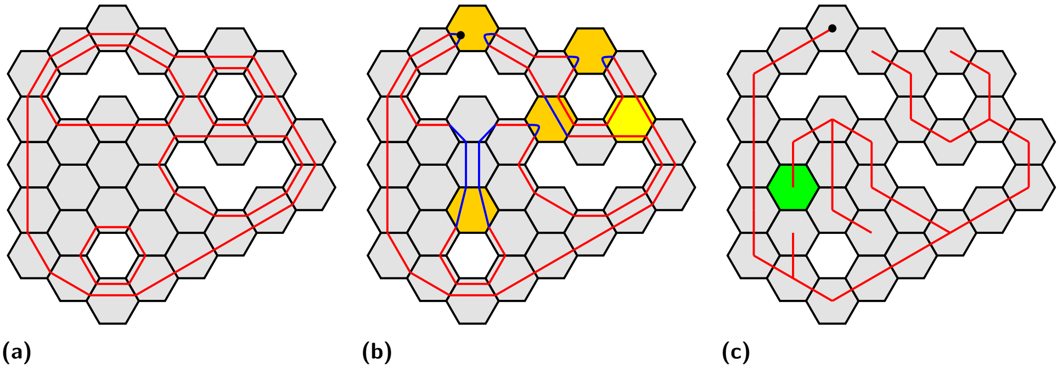
4.1 Canonicalized Construction

The general idea to construct a skeleton is to start with the cycles given by the boundary sets and to fuse these into a single cycle, i.e., a skeleton (see Figures 6 and 14). In order to fuse two boundary cycles, we have to determine a path between them. The boundary cycles are split at the respective endpoints of the path and connected along the path. Note that the fused cycle uses the path twice. The difficulty lies in finding paths between the boundary cycles and in avoiding the creation of new cycles. The construction is correct, i.e., we obtain a single cycle, iff the boundary sets and paths form a tree with the boundary sets as the nodes and the paths as the edges. In order to obtain a skeleton path, the skeleton is split at an arbitrary point, e.g., chosen by a leader election.

We now canonicalize the construction of a skeleton and a skeleton path. Recall that we define the canonical skeleton (path) with respect to a cardinal direction $d \in D_m \cup D_p$ and a sign $s \in \{+, -\}$. For the canonical skeleton, we have to define how the paths are constructed, and how the boundary cycles and paths are exactly connected. For the canonical skeleton path, we have to define the point to split the canonical skeleton. Afterwards, we show how the amoebot structure computes the canonical skeleton in a distributed fashion.

We start with the construction of the canonical skeleton. Let $\rho_s(d, x)$ denote the direction obtained if we rotate direction d by x degrees counterclockwise if the sign s is positive, and clockwise if the sign s is negative. Let $d_p = d$ if $d \in D_m$ and $d_p = \rho_s(d, 30)$ if $d \in D_p$. For each inner boundary set B, we construct a path as follows. First, we compute the global maxima of B with respect to direction d. Let B_d denote these global maxima. Then, we compute the global maximum of B_d with respect to direction $\rho_s(d, 90)$. Let u_B denote the global maximum.

▶ **Lemma 11.** *Amoebot u_B is adjacent to exactly one node in R, namely the one in direction $\rho_s(d_p, 180)$. Further, no amoebot in direction d_p of u_B is adjacent to a node in R.*



🟧 **Figure 6** (a) shows the initial situation. The red lines indicate the boundary cycles. (b) shows the $(N, +)$-skeleton. The yellow and orange amoebots indicate the global maxima of the boundary sets. The orange amoebots indicate the starting points of the paths. The blue lines indicate the paths between the boundary cycles. The node indicates the location where the cycle is split. (c) shows the spanning tree obtained from the $(N, +)$-skeleton. The node indicates the root.

The path starts at u_B and goes straight in direction d_p until it reaches an amoebot v_B of another boundary set. The existence of v_B is guaranteed by the outer boundary set. Clearly, all nodes of the path are occupied by amoebots. Note that the path may be trivial, i.e., $u_B = v_B$. There is only a single case where v_B is part of two boundary sets unequal to B. In this case, we take the boundary of v_B in direction $\rho_s(d_p, 60)$.

▶ **Lemma 12.** *The boundary sets and paths form a tree.*

It remains to define how the cycles and paths are exactly connected. We define that the cycle runs along the tree without crossing itself.

Next, consider the construction of the canonical skeleton path. We determine the splitting point u_{B_O} by applying the same procedure as for the starting points of the paths on the outer boundary set B_O. That is, we first compute the global maxima of B_O with respect to direction d, and then compute the global maximum of these global maxima with respect to direction $\rho_s(d, 90)$. If the canonical skeleton visits u_{B_O} multiple times, we pick a predefined position with respect to d_p (see Figure 11).

▶ **Lemma 13.** *The canonical skeleton (path) visits each bond at most twice. Thus, the canonical skeleton has linear complexity.*

In the following, we present our canonical skeleton algorithm that computes the canonical skeleton and the splitting point for the canonical skeleton path in parallel. Some instructions are performed on different subsets in parallel. In order to keep the amoebot structure synchronized, we apply the synchronization primitive (see Section 2). In a preprocessing step, each boundary set determines whether it is an inner or outer boundary set (see Corollary 3).

The canonical skeleton algorithm follows our construction of the canonical skeleton. In the first step, we compute the starting points of the paths and the splitting point by performing the global maxima algorithm on each boundary set B with respect to direction d, and on each resulting set B_d with respect to direction $\rho_s(d, 90)$. However, the computation of the boundary sets may interfere with each other since the boundary sets may intersect. In order to circumvent that problem, we add two additional external links and let each boundary set use the two external links closer to the corresponding empty region (see Figures 7a to 7c). Note that an amoebot is not able to determine whether two adjacent nodes belong to the

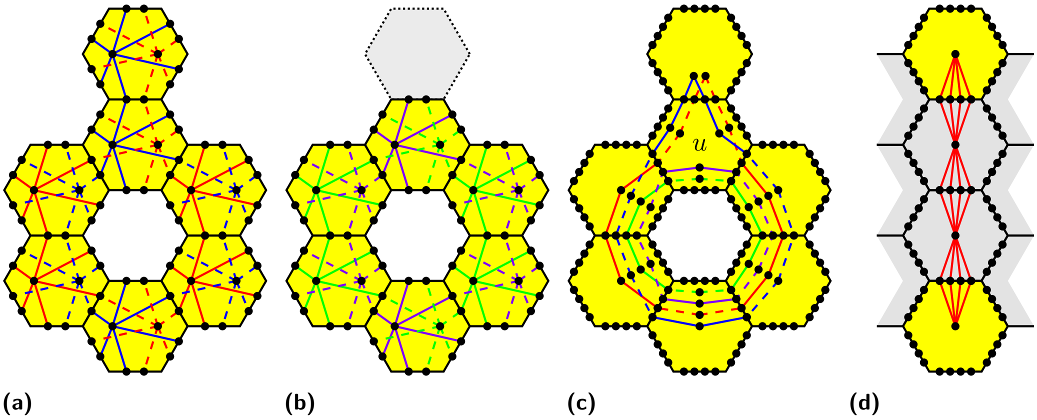


Figure 7 Computation of the global maxima of each boundary set with respect to $d = N$. (a) and (b) show the original construction for the outer and inner boundary set, respectively. The gray amoebot does not participate in the computation for the inner boundary set. (c) shows the construction for the computation in parallel. Amoebot u is unaware that two of its adjacent nodes belong to the same empty region. (d) shows the circuit utilized to identify the paths. The yellow amoebots are boundary amoebots. The gray amoebots are inner amoebots.

same empty region (see Figure 7c). Hence, it can only construct the primary and secondary circuits along the cycle. Subsequently, it handles each of its occurrences within the cycle separately. Nonetheless, the adjusted construction still satisfies the necessary properties given in Section 3.2.

The second step is the computation of the paths from the starting points straight into direction d_p. The canonical skeleton algorithm proceeds as follows. Each inner amoebot connects all pins of its neighbors in directions d_p and $\rho_s(d_p, 180)$, and each boundary amoebot connects all pins to its neighbors in direction d_p and $\rho_s(d_p, 180)$, respectively (see Figure 7d). Each starting point without a second boundary activates the circuit to its neighbor in direction d_p. Each amoebot that receives a beep is part of a path from the starting point straight into direction d_p. Finally, we obtain the following theorem:

▶ **Theorem 14.** *The canonical skeleton algorithm computes a (canonical) skeleton (path) in* $O(\log^2 n)$ *rounds w.h.p.*

4.2 Spanning Tree

We now show how a skeleton can be utilized to construct a spanning tree. We assume that we have already computed a (not necessarily canonical) skeleton (see Section 4.1). Our spanning tree algorithm consists of two phases. We first outline the goal of each phase. In the first phase, we construct a tree spanning all amoebots of the skeleton path. In the second phase, we add the remaining amoebots to the tree. Now, consider the first phase. We make use of the following lemma.

▶ **Lemma 15.** *Let* $G = (V, E)$ *be a connected graph. Let* $\pi = (v_1, \dots, v_m)$ *be a path in* G. *Let* $V' \subseteq V$ *be the set of all amoebots on the path* π. *Let* $\pi(v)$ *denote the first edge in* π *incident to* v. *Then,* $T = (V', E')$ *with* $E' = \bigcup_{v \in V' \setminus \{v_1\}} \{\pi(v)\}$ *is a tree.*

In order to determine the first occurrence of each amoebot, we apply the PASC algorithm algorithm on the path with v_0 as the reference amoebot (see Section 3.1). Each amoebot is able to determine its first occurrence by simply comparing the identifiers of all its occurrences. Each amoebot notifies the predecessor of its first occurrence.

Next, consider the second phase. For each amoebot v not included in the skeleton $S \setminus V'$, we add an edge from it to its northern neighbor w to the spanning tree. Note that v is an inner amoebot such that w has to exist. Otherwise, v would be included in the skeleton. Each amoebot $v \in S \setminus V'$ notifies its northern neighbor. We obtain the following theorem:

▶ **Theorem 16.** *Given a skeleton, the spanning tree algorithm computes a spanning tree after $O(\log n)$ rounds. Altogether, it requires $O(\log^2 n)$ rounds w.h.p.*

4.3 Symmetry Detection

We now show how to detect rotational symmetries and reflection symmetries. Due to the underlying infinite regular triangular grid graph G_Δ, there is only a limited number of possible symmetries. More precisely, an amoebot structure can only be 2-fold, 3-fold or 6-fold rotationally symmetric, and reflection symmetric to axes in a direction of $D_m \cup D_p$. Moreover, the problem is complicated by the facts that the symmetry point may be an unoccupied node of G_Δ or not a node of G_Δ at all, and that the symmetry axis may not be occupied by any amoebots.

Recall that we define a canonical skeleton by two parameters: the direction d and the sign s. Note that rotating the direction results in a rotated construction, and inverting the sign results in a reflected construction. Hence, a symmetric amoebot structure implies a symmetric construction of canonical skeletons. The idea of our symmetry detection algorithm is therefore to compare the canonical skeletons. We compare the skeletons according to the following observations (compare Figure 8).

▶ **Observation 17.** *An amoebot structure is 2-fold rotationally symmetric if the $(N, +)$-skeleton and the $(S, +)$-skeleton are symmetric. An amoebot structure is 3-fold rotationally symmetric if the $(N, +)$-skeleton and the $(ESE, +)$-skeleton are symmetric. An amoebot structure is 6-fold rotationally symmetric if it is 2-fold and 3-fold rotationally symmetric.*

Let $d \in D_m \cup D_p$ and let d' denote the direction obtained if we rotate d by $90°$ counterclockwise. An amoebot structure is reflection symmetric to an axis in direction d if the $(d', +)$-skeleton and the $(d', -)$-skeleton are symmetric. Note that due to symmetry, it is enough to only check half of $D_m \cup D_p$.

In order to compare two canonical skeletons, we map each canonical (d, s)-skeleton path to a unique bit string by having each amoebot on the path store a partial bit string of constant length encoding the direction of its successor relative to direction d and sign s. Consequently, the comparison of two canonical skeletons is reduced to the comparison of the corresponding bit strings of the two skeletons. In the following we show how such a comparison of two bit strings is possible in polylogarithmic time.

To this end, we consider the *string equality problem*: Let $A = (A_0, \dots, A_{m-1})$ and $B = (B_0, \dots, B_{m'-1})$ be two chains of amoebots with reference amoebots A_0 and B_0, holding bit strings $a = (a_0, \dots, a_{m-1})$ and $b = (b_0, \dots, b_{m'-1})$. We show how a and b can be checked for equality in time $O(\log^5 m)$ w.h.p. using probabilistic polynomial identity testing.

We first give a high-level overview of our solution: Since we can compare the length of A and B by comparing the identifiers $\mathrm{id}_{A,A_0}(A_{m-1}) = m - 1$ and $\mathrm{id}_{B,B_0}(B_{m'-1}) = m' - 1$ of the last amoebots of the chains bit by bit with the PASC algorithm (see Section 3.1) in time $O(\log m)$, we can assume $m = m'$ in the following. Let $c \in \mathbb{N}$. Chain A generates a prime $p \geq 2m$ and repeats the following procedure: A samples r uniformly at random from $[p]$ and sends the pair (p, r) to chain B. Chain A computes $f_a(r) = \sum_{i=0}^{m-1} a_i r^i \pmod p$, and chain B computes $f_b(r) = \sum_{i=0}^{m-1} b_i r^i \pmod p$ and sends the result to chain A which outputs "$a \neq b$"

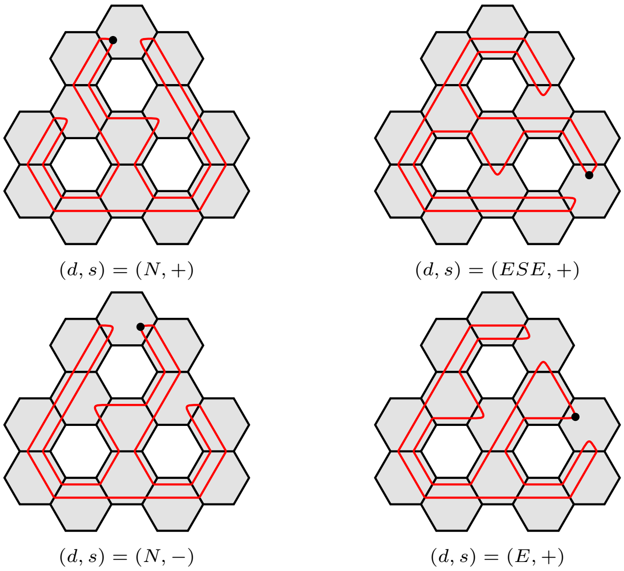


Figure 8 Symmetries of an amoebot structure. The amoebot structure is 3-fold rotational symmetric since the $(N,+)$- and the $(ESE,+)$-skeleton are symmetric, but not 2-fold or 6-fold rotational symmetric since the $(N,+)$- and the $(S,+)$-skeleton are not symmetric. Further, the amoebot structure is reflection symmetric to an axis into northern direction since the $(N,+)$- and the $(N,-)$-skeleton are symmetric, but not reflection symmetric to an axis into eastern direction since the $(E,+)$- and the $(E,-)$-skeleton are not symmetric.

if $f_a(r) \neq f_b(r)$ and repeats the procedure otherwise. After $c\lceil \log m \rceil$ repetitions, A outputs "$a = b$". Note that $a = b$ implies $f_a(r) = f_b(r)$. From the Schwartz-Zippel lemma follows that the one-sided error probability for a single repetition is $\Pr[f_a(r) = f_b(r) \mid a \neq b] \leq m/p \leq 1/2$. It follows $\Pr[A \text{ outputs "}a = b\text{"} \mid a \neq b] \leq 1/m^c$.

We now describe the algorithm in more detail: First, we describe a *block primitive* that we use to divide the chain A into blocks of length $k = O(\log m)$ where $k = 2^{\lceil \log \lambda \rceil}$, $\lambda = 2l$ and $l = \lceil \log m \rceil + 2$. Note that $k \geq \lambda$. We have $k \leq m$ for $m \geq 44 =: \eta$. From here on we assume $m \geq \eta$ (in case $m < \eta$, the chains A and B can simply compare their bit strings deterministically). Since the PASC algorithm terminates after $\lceil \log m \rceil$ iterations, we can easily determine amoebot A_λ by using the PASC algorithm 2 times and forwarding a marker after every iteration. Then we use the PASC algorithm again, with the following addition (compare Figure 13): For an amoebot let Q be the partition set on which it received a beep (either its primary or secondary partition set). An active amoebot (except A_0) splits Q into singletons. We obtain a circuit between each pair of consecutive active amoebots. Then, A_0 beeps on Q. If A_λ receives a beep, the procedure terminates, otherwise we continue with the next iteration. After termination, exactly the amoebots A_{ik} are active. Since we can directly compare the bits a_i of the amoebots between the last active amoebot and A_{m-1} with the corresponding bits b_i of chain B in time $O(\log m)$, we assume in the following w.l.o.g. that $k \mid m$ holds. This enables us to divide the amoebots of A into m/k chains $C_i = (A_{ik}, \dots, A_{(i+1)k-1})$ of length k with reference amoebot A_{ik}.

Now we describe how A generates a prime $p \geq 2m$. Chain A samples an l-bit integer $p = (p_0, \dots, p_{l-2}, 1)$ uniformly at random such that A_i stores p_i. Note that the most significant bit is fixed to 1 and therefore $p \in [2^{l-1}, 2^l[$, in particular $2m \leq p < 4m$. We check deterministically whether p is a prime by checking in parallel for all $2 \leq t < m$ whether

$t \mid p$ (note that $\lfloor \sqrt{p} \rfloor < m$): First, p and $t = \mathrm{id}_{A,A_0}(A_{ik}) = ik$ (using the PASC algorithm) are stored in the first l amoebots of every chain C_i in time $O(\log m)$. Then, all chains C_i repeat the following procedure in parallel for at most k times: If $t \geq 2$, check whether $t \mid p$ in time $O(\log^2 m)$ using binary long division with remainder. Abort the prime testing, if $t \mid p$, otherwise increment t.

We repeat the entire procedure at most $3cl^2$ times or until we have successfully sampled a prime p. The runtime for the prime generation is $O(\log^5 m)$. We now analyse the probability for the event E_{fail} that no prime is generated. Using non-asymptotic bounds on the prime-counting function, one can show that the fraction of integers in $[2^{l-1}, 2^l[$ that are prime is at least $1/(3l)$. It follows: $\Pr[E_{\text{fail}}] \leq (1 - 1/(3l))^{3cl^2} \leq 1/e^{cl} \leq 1/m^c$

We now address the probabilistic polynomial identity testing, focusing on the computation of $f_a(r)$ for $r \in [p]$. We use the previously determined division of the chain A into blocks of length $k = O(\log m)$. Assume that p, r and $e = \mathrm{id}_{A,A_0}(A_{ik}) = ik$ are stored in the first l amoebots of every chain C_i. All chains C_i repeat the following procedure in parallel for k times: Compute $s^{(i)} = a_e r^e \pmod{p}$ using modular exponentiation via the right-to-left binary method. Using binary long multiplication and division with remainder, this step is possible in time $O(\log^3 m)$. Then, increment e. Once all chains C_i have completed the j-th repetition, we compute the sum of the $s^{(i)}$ modulo p and store it in the first l amoebots of chain A using a generalization of Theorem 2. The summation is possible in time $O(\log^2 m)$. The computed sum is then added to a running total modulo p.

Finally, after k repetitions, the result $f_a(r)$ is stored in the amoebots $A_0, \ldots, A_{l-1}$. The runtime for the polynomial identity testing is $O(\log^4 m)$.

Note that the size of the outer boundary set is $\Omega(\sqrt{n})$, which is also a lower bound for the size of a canonical skeleton. Hence, we get the following result:

▶ **Theorem 18.** *The string equality problem on chains of length $O(m)$ can be solved in $O(\log^5 m)$ rounds w.h.p. Therefore, the symmetry detection problem can be solved in $O(\log^5 n)$ rounds w.h.p.*

Additionally, we can compute the amoebot occupying the symmetry point and amoebots on the symmetry axis, but due to the similarity to the applications in Section 3.2, we just sketch the algorithm. The idea is to identify some symmetric amoebots, e.g., global maxima, and to compute symmetric identifiers with these as reference amoebots (see Figure 12). We output all amoebots that receive the same identifier for each reference amoebot.

5 Conclusion and Future Work

In this paper, we have proposed polylogarithmic-time solutions for a range of problems. First, we have computed spatial identifiers in order to compute a stripe through a given amoebot and direction, and the global maxima of the given amoebot structure with respect to a direction. Using these results, we have constructed a canonical skeleton path, which provides a unique characterization of the shape of the given amoebot structure. Constructing canonical skeleton paths for different directions will then allow the amoebots to set up a spanning tree and to check symmetry properties of the given amoebot structure.

Our solutions could be useful for various applications like rapid shape transformation, energy dissemination, and structural monitoring. The details have to be worked out in future work. Beyond that, we think that exploring further applications for the spatial identifiers and the skeleton would be interesting.

References

1 John Calvin Alumbaugh, Joshua J. Daymude, Erik D. Demaine, Matthew J. Patitz, and Andréa W. Richa. Simulation of programmable matter systems using active tile-based self-assembly. In *DNA*, volume 11648 of *Lecture Notes in Computer Science*, pages 140–158. Springer, 2019.

2 Joshua J. Daymude, Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Christian Scheideler, and Andréa W. Richa. Convex hull formation for programmable matter. In *ICDCN*, pages 2:1–2:10. ACM, 2020.

3 Joshua J. Daymude, Kristian Hinnenthal, Andréa W. Richa, and Christian Scheideler. Computing by programmable particles. In *Distributed Computing by Mobile Entities*, volume 11340 of *Lecture Notes in Computer Science*, pages 615–681. Springer, 2019.

4 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: Algorithms and concurrency control. In *DISC*, volume 209 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

5 Joshua J. Daymude, Andréa W. Richa, and Jamison W. Weber. Bio-inspired energy distribution for programmable matter. In *ICDCN*, pages 86–95. ACM, 2021.

6 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot - a new model for programmable matter. In *SPAA*, pages 220–222. ACM, 2014.

7 Zahra Derakhshandeh, Robert Gmyr, Thim Strothmann, Rida A. Bazzi, Andréa W. Richa, and Christian Scheideler. Leader election and shape formation with self-organizing programmable matter. In *DNA*, volume 9211 of *Lecture Notes in Computer Science*, pages 117–132. Springer, 2015.

8 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating amoebots via reconfigurable circuits. *J. Comput. Biol.*, 29(4):317–343, 2022.

9 Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Comput.*, 33(1):69–101, 2020.

10 Ali Mashreghi and Valerie King. Broadcast and minimum spanning tree with o(m) messages in the asynchronous CONGEST model. *Distributed Comput.*, 34(4):283–299, 2021.

11 Jennifer E. Padilla, Ruojie Sha, Martin Kristiansen, Junghuei Chen, Natasha Jonoska, and Nadrian C. Seeman. A signal-passing dna-strand-exchange mechanism for active self-assembly of dna nanostructures. *Angewandte Chemie International Edition*, 54(20):5939–5942, 2015.

12 Dominic Scalise and Rebecca Schulman. Controlling matter at the molecular scale with dna circuits. *Annual Review of Biomedical Engineering*, 21(1):469–493, 2019. PMID: 31167101. doi:10.1146/annurev-bioeng-060418-052357.

13 Shalin Shah, Jasmine Wee, Tianqi Song, Luis Ceze, Karin Strauss, Yuan-Jyue Chen, and John Reif. Using strand displacing polymerase to program chemical reaction networks. *Journal of the American Chemical Society*, 142(21):9587–9593, 2020. PMID: 32364723. doi:10.1021/jacs.0c02240.

14 Tianqi Song, Abeer Eshra, Shalin Shah, Hieu Bui, Daniel Fu, Ming Yang, Reem Mokhtar, and John Reif. Fast and compact dna logic circuits based on single-stranded gates using strand-displacing polymerase. *Nature Nanotechnology*, 14(11):1075–1081, November 2019. doi:10.1038/s41565-019-0544-5.

15 Tommaso Toffoli and Norman Margolus. Programmable matter: Concepts and realization. *Int. J. High Speed Comput.*, 5(2):155–170, 1993.

A Overview

An overview of our results is given by Table 1.

Table 1 An overview of our algorithmic results.

Problem	Required pins	Runtime	Section	Theorem
Stripe	2	$O(\log n)$	Section 3.3	Theorem 9
Global maxima	2	$O(\log^2 n)$ w.h.p.	Section 3.3	Theorem 10
Canonical skeleton	4	$O(\log^2 n)$ w.h.p.	Section 4.1	Theorem 14
Spanning tree	4	$O(\log^2 n)$ w.h.p.	Section 4.2	Theorem 16
Symmetry detection	4	$O(\log^5 n)$ w.h.p.	Section 4.3	Theorem 18

B Omitted Proofs

Proof of Corollary 3. In order to accelerate the inner outer boundary test, we apply Theorem 2. Recall that each boundary set can organize itself into a cycle. We apply Theorem 1 to split the cycle into a chain. Each amoebot knows its predecessor and successor on the cycle and therefore also within the chain. Let x_i denote the angle at amoebot v_i and let $k = 5$. Finally, note that each boundary has $O(n)$ amoebots since each amoebot has at most three local boundaries. Thus, the sum of the angles can be accumulated after $O(\log n)$ rounds. ◀

Proof of Lemma 5. Since the reference amoebot u_r activates its primary circuit, it always receives a beep on its primary circuit. This implies $(u_r)_i = 0$ for $0 \leq i < k$. Thus, Equation 1 holds.

Let y be the successor of x. We have to show that Equation 2 holds, i.e., $\mathrm{id}_{C,u_r}(y) = \mathrm{id}_{C,u_r}(x) + 1$. We have two cases: (i) y never becomes passive, and (ii) y becomes passive. The first case directly implies $\mathrm{id}_{C,u_r}(x) = (1, \dots, 1) = -1$ and $\mathrm{id}_{C,u_r}(y) = (0, \dots, 0) = 0$. For the second case, let l denote the iteration where y becomes passive. Hence, y has received a beep on its primary circuit in the first $l - 2$ iterations and a beep on its secondary circuit in the $(l-1)$-st iteration, i.e., $y_{l-1} = 1$ and $y_i = 0$ for $0 \leq i < l - 1$. Since y is active, x has received a beep on its secondary circuit in the first $l - 2$ iterations and a beep on its primary circuit in the $(l-1)$-st iteration, i.e., $x_{l-1} = 0$ and $x_i = 1$ for $0 \leq i < l-1$. Since y is passive from the l-th iteration, $x_i = y_i$ holds for $l \leq i < k$. We obtain

$$\mathrm{id}_{C,u_r}(x) = (x_{k-1}, \dots, x_l, 0, 1, \dots, 1)$$
$$\mathrm{id}_{C,u_r}(y) = (y_{k-1}, \dots, y_l, 1, 0, \dots, 0) = (x_{k-1}, \dots, x_l, 0, 1, \dots, 1) + 1$$

since $2^{l-1} = \sum_{i=0}^{l-2} 2^i + 1$. ◀

Proof of Lemma 6. For the sake of analysis, we first consider an infinite amoebot structure where $S = V$. Afterwards, we transfer the results to arbitrary connected amoebot structures.

Consider a single stripe. Let *ACTIVE* denote the pin configuration used for amoebots of active stripes, and *PASSIVE* denote the pin configuration used for amoebots of passive stripes. Both pin configurations define a primary and secondary partition set. All amoebots within the stripe connect their primary and secondary partition sets, respectively (see Figure 9). We define the union of all primary resp. secondary partition sets within a stripe as the primary resp. secondary partition set of the stripe. Note that each amoebot has access to both partition sets and is able to distinguish between them.

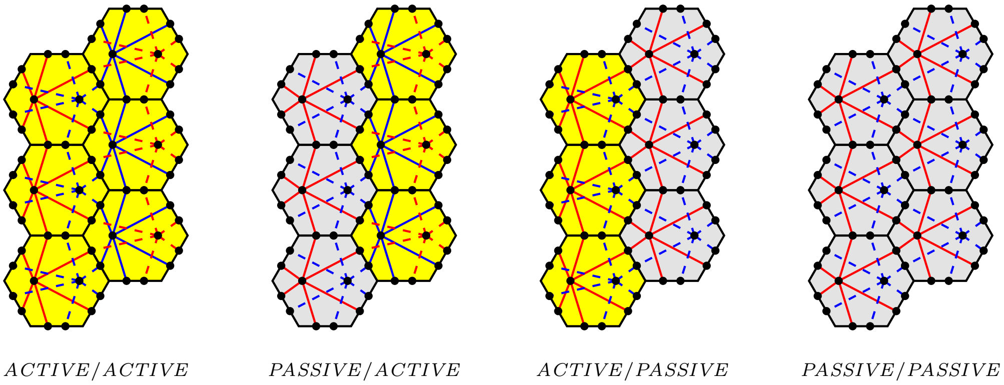


Figure 9 Sections of various stripes of an infinite amoebot structure.

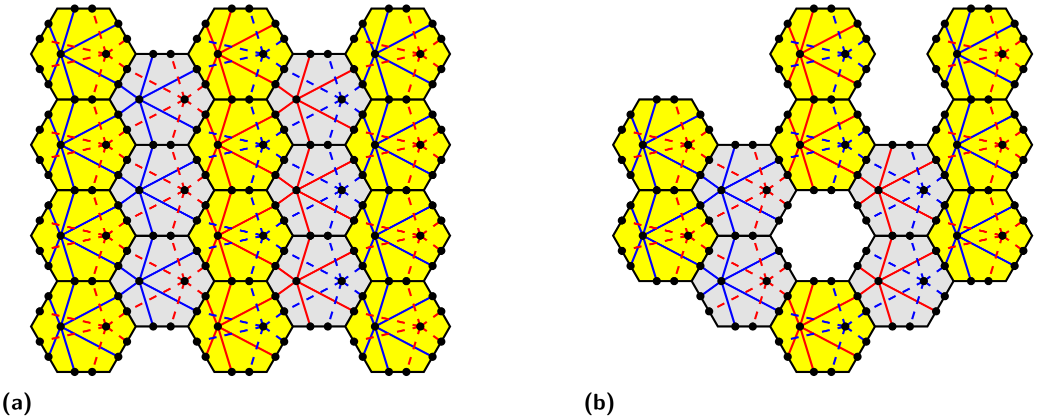


Figure 10 (a) A section of the infinite amoebot structure. (b) An arbitrary amoebot structure. The connectivity of the remaining amoebots is preserved.

Next, consider the connections to the preceding stripe. The connections within the pin configuration *ACTIVE* are selected in such a way that an active stripe connects its primary partition set exclusively to the secondary partition set of the preceding stripe, and its secondary partition set exclusively to the primary partition set of the preceding stripe (see Figure 9). Similar, the connections within the pin configuration *PASSIVE* are selected in such a way that an active stripe connects its primary partition set exclusively to the primary partition set of the preceding stripe, and its secondary partition set exclusively to the secondary partition set of the preceding stripe (see Figure 9). Note that there are no further connections.

The crucial property of our construction is that any two amoebots are connected by any arbitrary path of amoebots in the same fashion, i.e., either both primary partition sets are connected to the secondary partition set of the other amoebot, or their primary and secondary partition sets are connected, respectively (see Figure 10a). This allows us to remove amoebots without separating the circuits as long as the amoebot structure stays connected (see Figure 10b). ◀

Proof of Lemma 7. By Lemma 6, we can apply the primitive of primary and secondary circuits to the chain of stripes as long as each amoebot knows whether its stripe is active or passive. This implies that we can perform the PASC algorithm that by Lemma 5, computes $\mathrm{id}_{\mathcal{C}_d,A_r}$.

It remains to show that each amoebot knows whether its stripe is active or passive throughout the execution of the algorithm. Initially, this is trivially true since each stripe is active. A stripe becomes passive once it receives a beep on its secondary circuit. Each amoebot of the stripe can observe this beep since by construction, each amoebot has access to the secondary partition set of its stripe. Thereafter, the stripe stays passive. ◀

Proof of Lemma 8. The proof works analogously to the one of Lemma 7. ◀

Proof of Lemma 12. Let $\text{rank}(B) = \min_{w \in B} \text{f}_d(S, w)$ be the rank of an inner boundary set B, and let $\text{rank}(B_O) = -1$ be the rank of the outer boundary set B_O (compare to the definition of the global maxima in Section 1.3). The rank of an inner boundary set is lower than the rank of another inner boundary set if its global maxima are further in direction d than the global maxima of the other inner boundary set. Furthermore, the outer boundary set has a lower rank than all ranks of the inner boundary sets.

We claim that for each inner boundary set B, we construct a path from B to another boundary set B' such that $\text{rank}(B) > \text{rank}(B')$ holds. Clearly, this relationship cannot be cyclic. The lemma immediately follows since we construct a path for each inner boundary set. We prove the claim in the following.

Lemma 11 excludes the possibility of a self-loop, i.e., $B \neq B'$ holds. The claim holds by definition if B' is the outer boundary set. Suppose that B' is an inner boundary set. The claim also holds if the path from u_B to v_B is not trivial since $\text{rank}(B) = \text{f}_d(S, u_B) > \text{f}_d(S, v_B) > \text{rank}(B')$ holds.

Suppose that the path is trivial, i.e., $u_B \in B$ and $u_B \in B'$. Let $w \in R'$ be a node adjacent to u_B. Note that $\text{f}_d(S, u_B) \geq \text{f}_d(S, w)$ holds since otherwise, $R = R'$ and with that $B = B'$ would hold. We go from w into direction d_p until we reach an amoebot $x \in B'$. Note that $\text{f}_d(S, w) > \text{f}_d(S, x)$ holds since for $V \setminus R_O$, f_d is strictly monotonically decreasing if we go into direction d_p. This amoebot exists since B' is an inner boundary set. The claim holds since $\text{rank}(B) = \text{f}_d(S, u_B) \geq \text{f}_d(S, w) > \text{f}_d(S, x) > \text{rank}(B')$ holds. ◀

Proof of Lemma 11. Amoebot u_B has to be adjacent to at least one node in R. Otherwise, $u_B \notin B$ would hold. It is easy to see that if that node would lie in another direction than $\rho_s(d_p, 180)$, either u_B would not be a global maximum of B with respect to d, or u_B would not be the global maximum of B_d with respect to direction $\rho_s(d, 90)$. By the same reasoning, u_B would not be a global maximum of B with respect to d if any amoebot in direction d_p of u_B would be adjacent to a node in R. ◀

Proof of Lemma 13. The canonical skeleton (path) visits a bond either due to a boundary cycle or due to one of the paths. Each local boundary (a common unoccupied adjacent node of the endpoints) adds one visit. Each bond has at most two local boundaries. Each path adds two visits. Due to Lemma 11, a bond cannot be part of more than one path. A bond cannot be part of a boundary cycle and a path at the same time since the path would stop at either endpoints due to the unoccupied adjacent node. Altogether, each bond is visited at most twice. ◀

Proof of Theorem 14. The preprocessing step requires $O(\log n)$ rounds w.h.p. (see Section 2). The first step requires $O(\log^2 n)$ rounds for the computation of global maxima (see Section 3.3). The second requires $O(1)$ rounds. Altogether, the canonical skeleton algorithm requires $O(\log^2 n)$ rounds w.h.p. ◀

Proof of Lemma 15. In order to prove that T is a tree, we show that T is cycle-free and connected. Each edge $\pi(v) = (v', v)$ implies that v' appears before v in π. Clearly, this relationship cannot be cyclic.

We prove that T is connected by induction on the path π. The induction base holds trivially for v_1. Suppose that all nodes up to node v_i are connected within T. Consider node v_{i+1}. If it is not the first occurrence of v_{i+1} on the path, then v_{i+1} is already connected by induction hypothesis. Otherwise $\pi(v_{i+1}) = \{v_i, v_{i+1}\} \in E'$. This edge connects v_{i+1} to all nodes up to node v_i since these are connected by induction hypothesis. ◀

Proof of Theorem 16. The correctness follows from Lemma 15. The first phase requires $O(\log n)$ rounds (see Section 3.1). The second phase requires $O(1)$ rounds. Altogether, the spanning tree algorithm requires $O(\log n)$ rounds. ◀

C Omitted Figures

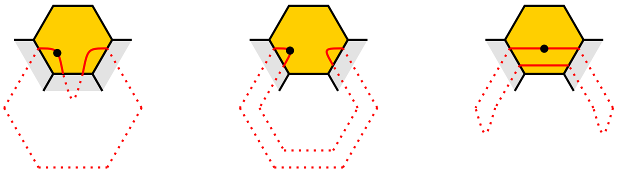


Figure 11 Predefined splitting point with respect to d_p. For $d_p = N$, the figure shows all cases where the canonical skeleton visits u_{B_O} multiple times. By similar arguments as for Lemma 11, there are no other cases. The red lines indicate the canonical skeleton. The node indicates the splitting point.

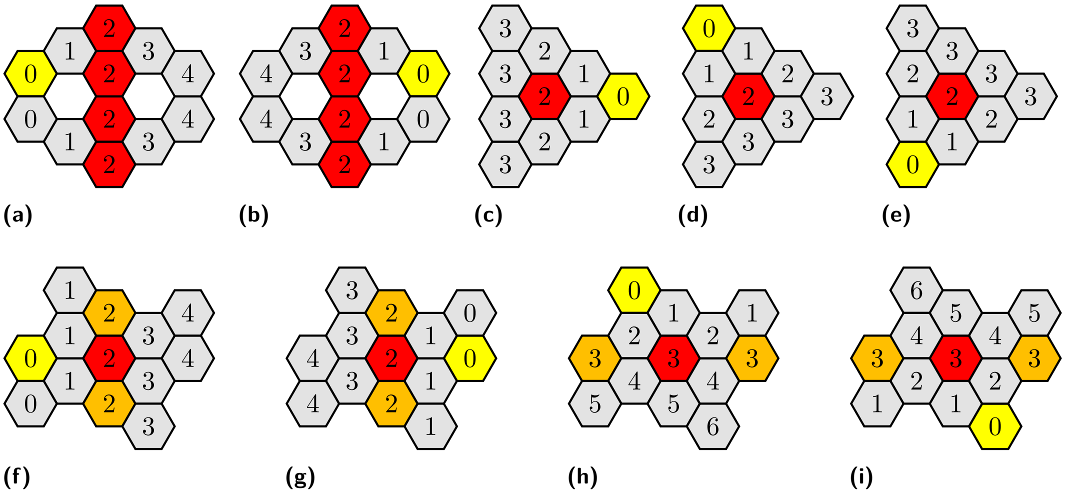


Figure 12 Symmetry point and symmetry axis. (a) and (b) show the computation of a symmetry axis. (c) to (e) show the computation of a symmetry point for a 3-fold or 6-fold rotationally symmetric amoebot structure. (f) to (i) show the computation of a symmetry point for a 2-fold rotationally symmetric amoebot structure. In this case, we have to compute two axes so that the symmetry point lies on the intersection of these. The yellow amoebot indicates the reference amoebot. The red (and orange) amoebots receive the same identifier for each reference amoebot.

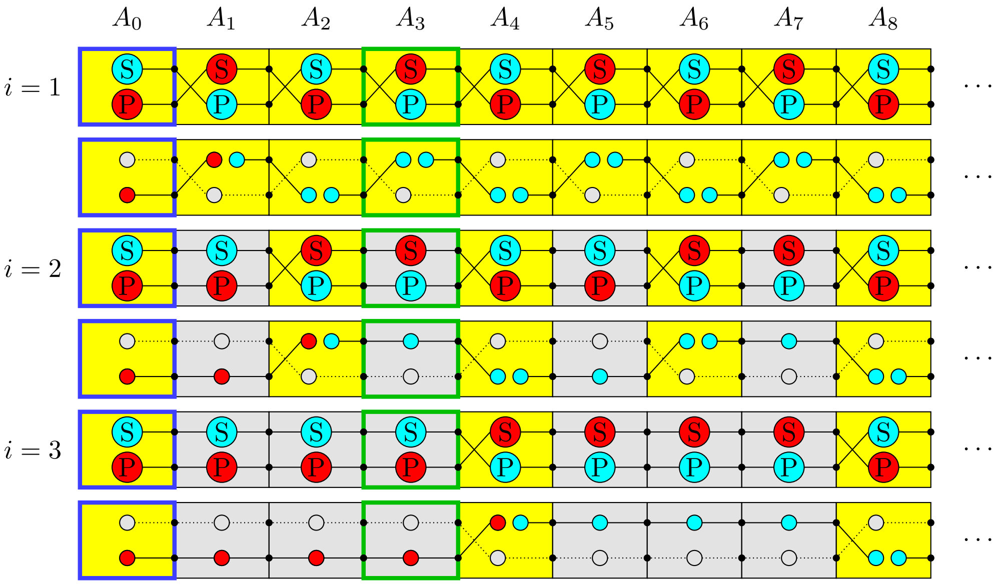


Figure 13 Block primitive. The figure shows how a chain A with reference amoebot A_0 and a marked amoebot A_λ can be divided into blocks of length $k \geq \lambda$ where $k = 2^{\lceil \log \lambda \rceil}$ using the PASC algorithm (compare Figure 3) with an additional step after each iteration. Here, we have $\lambda = 3$ and $k = 4$. The first line shows the chain at the beginning of the i-th iteration of the PASC algorithm. The circles indicate the partition sets. The blue bordered amoebot is A_0, and the green bordered amoebot is A_λ. Yellow amoebots are active, and gray amoebots are passive. The second line shows the configuration resulting from the following additional step: For an amoebot let Q be the partition set on which it received a beep (depicted in red; either its primary or secondary partition set). An active amoebot (except A_0) splits Q into singletons. We obtain a circuit between each pair of consecutive active amoebots. Then, A_0 beeps on Q. If A_λ receives a beep, the procedure terminates, otherwise continue with the next iteration. After termination, exactly the amoebots A_{ik} are active.

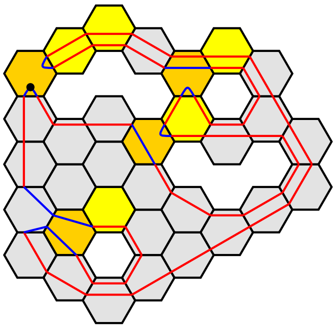


Figure 14 The figure shows $(NNW, +)$-skeleton. The yellow and orange amoebots indicate the global maxima of the boundary sets. The orange amoebots indicate the starting points of the paths. The blue lines indicate the paths between the boundary cycles. The node indicates the location where the cycle is split.

Fault-Tolerant Shape Formation in the Amoebot Model

Irina Kostitsyna
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Christian Scheideler
Department of Computer Science, Universität Paderborn, Germany

Daniel Warner
Department of Computer Science, Universität Paderborn, Germany

Abstract

The amoebot model is a distributed computing model of programmable matter. It envisions programmable matter as a collection of computational units called amoebots or particles that utilize local interactions to achieve tasks of coordination, movement and conformation. In the geometric amoebot model the particles operate on a hexagonal tessellation of the plane. Within this model, numerous problems such as leader election, shape formation or object coating have been studied. One area that has not received much attention so far, but is highly relevant for a practical implementation of programmable matter, is fault tolerance. The existing literature on that aspect allows particles to crash but assumes that crashed particles do not recover. We proposed a new model [14] in which a crash causes the memory of a particle to be reset and a crashed particle can detect that it has crashed and try to recover using its local information and communication capabilities. We present an algorithm that solves the hexagon shape formation problem in our model if a finite number of crashes occur and a designated leader particle does not fail. At the heart of our solution lies a fault-tolerant implementation of the spanning forest primitive, which, since other algorithms in the amoebot model also make use of it, is also of general interest.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Computational geometry

Keywords and phrases programmable matter, amoebot model, fault tolerance, shape formation

Digital Object Identifier 10.4230/LIPIcs.DNA.2022.9

Supplementary Material *Software (Source Code)*: `https://github.com/dwarner-git/AmoebotSim-dna28`; archived at `swh:1:dir:f6dff854d1fc85d9446086ec307437c8487bb7af`

1 Introduction

The research on *programmable matter*, first introduced in [16], deals with small simple particles that locally interact to globally solve a given task like shape formation or coordinated movement. Many future applications are conceivable, such as smart materials, autonomous monitoring and repair, and minimal invasive surgery. The amoebot model, introduced in [4], is a popular model in the context of programmable matter, envisioning particles (or *amoebots*) as computational entities on the micro or nano level. Amoebots have only finite memory and move in a way inspired by amoeba. In this work we extend the amoebot model to include the aspect of *fault tolerance* by introducing *particle crashes* [14]. Crashed particles can recover using their local information and communication capabilities and thus rejoin solving the global task at hand. In order to gain initial insights into useful strategies towards fault tolerance in our model, we examine and solve the *hexagon shape formation problem*. Our solution builds upon two primitives, a propagation primitive and a safety primitive, which both may be of interest in their own.

28th International Conference on DNA Computing and Molecular Programming (DNA 28).
Editors: Thomas E. Ouldridge and Shelley F. J. Wickham; Article No. 9; pp. 9:1–9:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Related Work

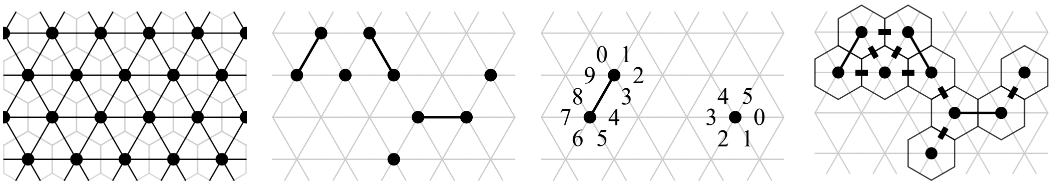
Shape formation in the amoebot model, in particular the hexagon shape formation problem, was investigated in [6, 7, 10]. The authors of [12] show how a constant number of amoebots can be used to simulate a Turing-complete entity that can move in the plane during computation and show how this can be used to form more general shapes than the previously published algorithms. The hexagon shape formation problem was recently revisited in [2] where it was shown how to define an algorithm for the problem that is correct in a concurrent setting.

In the literature on the amoebot model, the topic of fault tolerance has received little attention so far. We are only aware of the following publications: In [11] the authors present a solution to the *line recovery* problem in the amoebot model under a semi-synchronous adversarial scheduler: Initially, n particles, of which $f < n - 4$ are faulty, are positioned on a line. The algorithm forms either a single line or two lines of equal size consisting of all non-faulty particles. Faulty particles may not move or communicate with other particles. An essential difference to our work is that the faulty particles are predetermined by the initial configuration, no crashes occur during the execution of the algorithm and faulty particles cannot recover. The authors of [15] study the *Connected Line Recovery problem*: The proposed algorithm lets the non-faulty particles form a line while maintaining connectivity at all times, regardless of the initial distribution of the faulty particles. It is shown that a necessary condition for the solvability of the problem is that faulty particles can still communicate with their neighbours. Here too, however, it is assumed that faulty particles cannot recover. Daymude et al. present in [3] as an independent partial result the so-called *forest-prune-repair algorithm*, an implementation of the spanning forest primitive that copes with crash failures. Unlike in the present work, the crash failures are permanent there, i.e., particles that have crashed once cannot recover. Furthermore the authors assume that the subgraph induced by the nodes occupied by non-crashed particles is always connected - a strong assumption not necessary in our model where crashes are temporary.

2 Model

In this section, we introduce our model, which is based on the *geometric amoebot model*.

Particles. We assume that particles (*amoebots*) occupy nodes on the triangular lattice $G_\triangle = (V_\triangle, E_\triangle)$ (Figure 1a). A particle occupies either a single node (*contracted* particle) or two adjacent nodes (*expanded* particle). A node can be occupied by at most one particle (Figure 1b). Two particles occupying adjacent nodes are called *neighbours* or *connected*.



(a) A section of $G_\triangle$ (black) and its dual, the hexagonal tiling (gray).

(b) Example showing contracted and expanded particles on $G_\triangle$.

(c) Two particles with different orientations and common chirality.

(d) Adjacent particles establish bonds through the ports facing each other.

Figure 1 Illustration of some aspects of the geometric amoebot model.

Global directions. For $n \in \mathbb{N}$ let $[n] := \{ 0, 1, \dots, n-1 \}$. A *direction* is an integer in [6] that represents a vector in the Euclidean plane. We distinguish between *global* and *local* directions (see below). The global direction 0 corresponds to the vector pointing right in the triangular lattice $G_\triangle$, and the other global directions increase counter-clockwise.

Port labels. A particle has ports for edges between a node it occupies and a neighbouring node not occupied by it, i.e., a particle has ports for those edges which are incident with exactly one of the nodes it occupies. A contracted particle has six ports and an expanded particle has ten ports (Figure 1c). The ports have unique labels from the particle's own local perspective, as follows: A particle p has a fixed *orientation* $O(p) \in [6]$ that is unknown to it. Particles may have different orientations. We assume that all particles have a common (counter-clockwise) *chirality.*[1] A contracted particle has port labels in [6], an expanded particle has port labels in [10]. If p is contracted, then the edge pointing in global direction $O(p)$ gets label 0. If p is expanded, then there may be two edges pointing in global direction $O(p)$. In this case, the edge pointing in global direction $O(p)$ and pointing "away" from p (i.e., to a node adjacent to only one of the two nodes occupied by p) gets label 0. From label 0, labels increase counter-clockwise around the particle.

Local directions. In order to be able to recognise whether it is contracted or expanded, and to be able to convert port labels into local directions, a particle p has a so-called *bearing* $B(p) \in \{4, 5, 6\} \cup \{\epsilon\}$ which is updated during its movement and can be read by it during its computation. If p is contracted, then $B(p)$ is ϵ. If p is expanded, then p occupies two nodes $\{u, v\} \in E_\triangle$. Let u be the node incident to the edge with port label 0. Then, the bearing $B(p)$ is the port label of the port furthest from u. For a port label l a particle p can easily compute the local direction $d \in [6]$ of the corresponding port using only l and the bearing $B(p)$. In the case of a contracted particle, the label and corresponding local direction are equal. A local direction d corresponds to the global direction $(d + O(p)) \bmod 6$, i.e., $O(p)$ encodes p's offset for local direction 0 from global direction 0. For neighbours p and q and a direction $d_p \in [6]$ local to p let $\Delta(d_p, p, q)$ denote the corresponding direction local to q.

Memory. Each particle has a constant-size local memory. Due to the constant-size memory constraint, particles cannot have a unique id and are therefore anonymous. We assume that a particle has a variable *state* in its local memory.

Communication. Adjacent particles establish bonds through the ports facing each other (see Figure 1d). A particle p can determine for a port label l whether the associated node is empty or occupied, and if it is occupied by another particle q, then p can read q's corresponding label, read q's bearing and read from and write to q's local memory. This effectively enables communication between neighbouring particles.

Movement. Particles can move through *expansion* and *contraction*: When a particle is contracted, it can *expand* into an unoccupied neighbouring node. When a particle is expanded it can *contract* into one of the two nodes it currently occupies. Additionally two neighbouring particles can combine expansion and contraction to perform a coordinated movement: one

[1] The assumption is justified: With the hexagon shape formation problem we assume that a unique leader (SEED particle) is initially available. The SEED could, in the case of non-common chirality, impose its chirality on all other particles in the system (see also Section 4 in [13])

particle can contract out of a node while another expands into that node. This so-called *handover* helps to maintain connectivity of the particle system. We distinguish between two operations: Consider a contracted particle p occupying a node $u \in V_\triangle$ and an expanded particle q occupying two nodes $\{v, w\} \in E_\triangle$. If p and q are neighbours, i.e. $\{u, v\} \in E_\triangle$, then p may *push* q into w. Similarly, q may *pull* p into v. In both cases the result is that p is expanded and occupies the nodes $\{u, v\}$ and that q is contracted and occupies the node w.

The bearing of the particles involved in a movement is updated accordingly.

Head and tail. We assume the following implicitly for our algorithm without noting the implementation in the pseudocode: A particle p has a variable *tailDir* in its local memory, which is ϵ if p is contracted and equal to the direction from which p last expanded if p is expanded. We call the node into which p expanded *head* and the node from which p expanded *tail*. If p is contracted, the node it occupies is called the head. The variable *tailDir* is kept up-to-date during movement operations. In the event of a crash, the information stored in *tailDir* is lost, so that a particle no longer knows which node previously was its head (or tail).

Scheduling. As is common in the amoebot literature, we will assume a sequential fair scheduler for the analysis of our algorithm: In every time step $t \in \mathbb{N}$ the scheduler may either *activate* or *crash* a particle (see below). At any given time at most one particle is active (sequential activations). Furthermore the interval between any two activations of the same particle is finite (fairness property). A *round* is completed as soon as every particle has been activated at least once. An activated particle may use the operations described above: It can read its own local memory and that of its neighbours, perform a computation, update its own memory and the memory of its neighbours and perform at most one movement/handover. Due to the sequential activation model, atomicity and isolation of these actions is ensured.[2]

Model extension: Particle crashes. In our work we extend the amoebot model by introducing *particle crashes*. We assume that the adversarial scheduler may arbitrarily crash particles. A crash of a particle p has the following effects: The scheduler sets the *state* in p's local memory to CRASHED, enabling p and its neighbours to detect that it has crashed, and resets the rest of p's local memory. Note that since the bearing of a particle is not part of the local memory, it cannot be changed by the scheduler. In our algorithm `HexagonFT`, a CRASHED particle detects that it has crashed when it is activated, initialises its memory and initially switches to the additional ERROR *state*; this means in particular that a CRASHED particle is not permanently disabled, but can resume its function after its memory has been reset. A particle in *state* CRASHED or ERROR is called *faulty*. We say a faulty particle *recovers* if it becomes non-faulty.

Configuration. The configuration of a particle $p \in P$ at the beginning of time step $t \in \mathbb{N}$ is the set of nodes $V(p) \subseteq V_\triangle$ it occupies (which implicitly gives its bearing $B(p)$) and its local memory $M(p)$. The configuration $C = (V(p), M(p))_{p \in P}$ of a system P of n particles at the beginning of time step $t \in \mathbb{N}$ describes the configuration of each particle $p \in P$. The *connectivity graph* $G(C)$ is the subgraph of $G_\triangle$ induced by the nodes occupied by particles in configuration C. A configuration is called *non-faulty* if no particles are faulty. A *stable* configuration is a configuration that does not change any more with activations.

[2] Daymude et al. present a formal framework for achieving atomicity and isolation in the amoebot model without this assumption [2]. Furthermore, the authors demonstrate the correctness of their version of the `Hexagon` algorithm under any unfair asynchronous scheduler. We are confident that the analysis could also be applied to our algorithm `HexagonFT` in our extended model.

Problem description: Hexagon shape formation problem. For any two nodes $u, v \in V_\triangle$ the *distance* $\delta(u,v) \in \mathbb{N}_0$ between u and v is defined as the length of a shortest path from u to v in $G_\triangle$. For a node $v \in V_\triangle$ and $i \in \mathbb{N}_0$ let $B(v,i) := \{\, u \in V_\triangle \mid \delta(u,v) = i \,\}$. We call a set $V \subseteq V_\triangle$ a *hexagon with centre* $v \in V$ if there is a $k \in \mathbb{N}_0$ and a subset $S \subseteq B(v,k)$ such that $V = S \cup \bigcup_{i<k} B(v,i)$. We now define the *hexagon shape formation problem* **HEX**: We assume that the system of particles initially forms a single connected component of contracted particles, has a unique leader, called the SEED particle, and that all other particles are IDLE. Furthermore, since we are currently working on a manuscript in which we show that the leader election problem is solvable in our fault tolerance model, we assume in this paper that the SEED particle does not crash. The goal is to reach a stable configuration in which the set of nodes occupied by particles is a hexagon with the SEED in its centre.

3 Algorithm

3.1 Overall Description

General structure. The general structure of our fault-tolerant hexagon shape formation algorithm `HexagonFT` is given as pseudocode in Algorithm 1. All pseudocode is written from the perspective of an activated particle p. As long as no crashes occur, our algorithm `HexagonFT` basically behaves like the classical `Hexagon` algorithm. In Figure 2 we describe how the hexagon shape formation problem **HEX** is solved by `HexagonFT` in the geometric amoebot model without crashes. In the following Section 3.2, we explain the sub-algorithms of `HexagonFT` in detail, in particular how they deal with crashed particles.

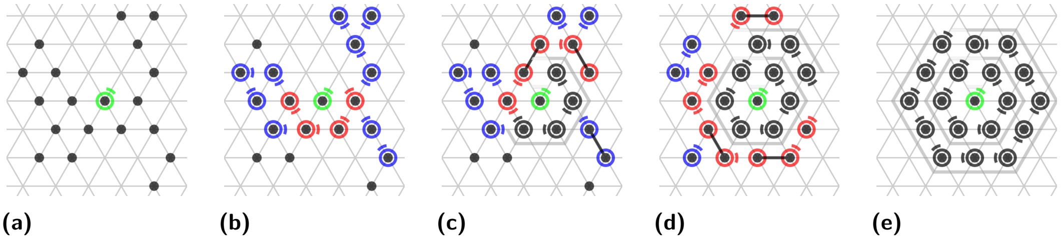


Figure 2 An example run of our hexagon shape formation algorithm `HexagonFT` with 19 particles without crashes. Particles have to assume the shape of a hexagon (but for the outer layer, which may not be completely full). The hexagon is built in a spiral ring in clockwise direction around the SEED as follows: (a) All particles except of the SEED are initially IDLE (black dots). (b) Particles adjacent to finished particles (SEED or RETIRED) become ROOT particles (sub-algorithm `tryToBecomeRoot`), and FOLLOWER particles form parent-child relationships with ROOT or FOLLOWER particles (sub-algorithm `tryToBecomeFollower`). (c)–(e) ROOT particles traverse the forming hexagon counter-clockwise (sub-algorithm `performMovement`), becoming RETIRED (sub-algorithm `tryToBecomeRetired`) when reaching the position marked by the last RETIRED particle. FOLLOWER particles follow ROOT particles via a series of handovers (sub-algorithm `performMovement`).

Variables. The local memory contains the following variables (the initial value is noted after the variable name):

1. *state*: General state. Domain: SEED, IDLE, FOLLOWER, ROOT, RETIRED, ERROR, CRASHED. We call a particle *faulty* if its *state* is CRASHED or ERROR and otherwise *non-faulty*.
2. *tailDir*: Direction of the tail of a non-faulty particle. Domain: $[6] \cup \{\epsilon\}$.

Algorithm 1 HexagonFT.

```
def HexagonFT($p$):
    if $p$ is finished:
        return ▷ Finished particles do nothing.
    if $p.state$ = CRASHED:
        $p$.initializeAfterCrash()
    if $p$ has a neighbour in $state$ CRASHED:
        return ▷ Wait until all neighbours in $state$ CRASHED are “awake”, i.e. in $state$
            ERROR, and therefore initialized.
    if $p.state$ = ERROR:
        $p$.updateFlags() ▷ Update $safeFlags$ and $safeState$.

    ▷ Try $state$ change:
    if $p$ can retire:
        $p$.tryToBecomeRetired()
        if $p.state$ = RETIRED:
            return ▷ $p$ has become retired and therefore does not do anything more.
    else if $p$ is in $state$ ERROR, IDLE or FOLLOWER:
        $p$.tryToBecomeRoot()
        if $p.state$ = ROOT:
            $p.pathToRootState$ ← VALID
        else if $p.state$ = IDLE:
            $p$.tryToBecomeFollower()
        else if $p.state$ = ERROR and $p.safeState$ = SAFE:
            $p$.tryFollowerRecoveryByPropagation()

    ▷ Propagation of $pathToRootState$:
    if $p.state$ = FOLLOWER and $p.pathToRootState$ = INVALIDATE:
        $p.pathToRootState$ ← INVALID
        $p$.propagateInvalidate()
    else if $p.state$ = ROOT or ($p.state$ = FOLLOWER and $p.pathToRootState$ = VALID):
        $p$.propagateValid()

    ▷ Movement:
    if $p$ is in $state$ FOLLOWER or ROOT:
        $p$.performMovement()
```

3. *constructionDir*, *moveDir*, *followDir*: Shape formation specific variables of a non-faulty particle. Domain: Direction in [6].
4. *pathToRootState* ← VALID: State of an ERROR or non-faulty particle as node on a path upwards to a ROOT particle. Domain: VALID, INVALID, INVALIDATE.
5. *safeState* ← UNDETERMINED, *safeFlags*$[r][d]$ ← UNDETERMINED (round $r \in [2]$, direction $d \in [6]$): The *safeState* of an ERROR particle is used to determine whether it may become a FOLLOWER. The *safeFlags* are used to determine the current value of *safeState*. Domain: UNDETERMINED $= 0$, UNSAFE $= 1$, SAFE $= 2$.
6. *hasInvalidated*$[l]$ ← FALSE: *hasInvalidated*$[l]$ of an ERROR particle is TRUE if its neighbour at port label l has been invalidated. Domain: FALSE, TRUE.

Terminology. We introduce some notions: A particle is *finished* if its *state* is SEED or RETIRED. A particle p *can retire* if it is CONTRACTED and has a finished neighbour q with *constructionDir* set to the position of p. We call q *(direct) predecessor* of p and p *(direct) successor* of q. Based on this, the terms *predecessor* and *successor* are defined in the canonical way. We say a FOLLOWER particle p *follows* a particle q, if the node in the direction of *p.followDir* relative to the head of p is occupied by q. We call q *the parent of* p and p *a child of* q. If q is expanded and the node in the direction of *p.followDir* relative to the head of p is occupied by q's tail, we call p a *tail follower* of q. *Head follower* is defined analogously. An expanded particle has a *blocking tail neighbour* if it has a tail follower or a neighbour in *state* IDLE, ERROR or CRASHED that is adjacent to its tail.

3.2 Sub-Algorithms

In this section the sub-algorithms of algorithm `HexagonFT` are presented.

3.2.1 initializeAfterCrash

As soon as a CRASHED particle is activated, its memory is initialized by sub-algorithm `initializeAfterCrash` (see Algorithm 2): the *state* is set to ERROR and the other variables to default values. To ensure that an activated particle does not read the corrupt memory of CRASHED neighbouring particles, `HexagonFT` (Line 6) also checks whether the particle has CRASHED neighbours and only continues execution if it does not.

Algorithm 2 `initializeAfterCrash`.

```
def initializeAfterCrash($p$):
    ▷ Initialization of crashed particle after crash (“wake up”):
    $p.state \leftarrow$ ERROR
    $p.pathToRootState \leftarrow$ INVALID
    $p.safeFlags[r][d] \leftarrow$ UNDETERMINED for all rounds $r \in [2]$ and all directions $d \in [6]$
    $p.safeState \leftarrow$ UNDETERMINED
    $p.hasInvalidated[l] \leftarrow$ FALSE for all port labels $l$
```

3.2.2 tryToBecomeRetired

A particle p that has a finished neighbour q with *q.constructionDir* set to its position can become RETIRED, if it can successfully determine its *constructionDir*, i.e. the position for the next particle to become RETIRED. The sub-algorithm `tryToBecomeRetired` (see Algorithm 3 and Figure 3) determines whether a particle can become RETIRED, whereby the desired hexagon shape is successively constructed around the SEED in a snake-like fashion.

3.2.3 tryToBecomeRoot

A particle in *state* ERROR, IDLE or FOLLOWER that has a finished neighbour but cannot retire could potentially become a ROOT particle. Sub-algorithm `tryToBecomeRoot` (see Algorithm 4 and Figure 3) is responsible for changing the state of a particle to ROOT, if possible. ROOT particles move counter-clockwise around the hexagonal retired structure. The auxiliary

Algorithm 3 `tryToBecomeRetired`.

```
def tryToBecomeRetired($p$):
    if $p$ has the SEED $s$ as neighbour at direction $d_p$:
        $i \leftarrow (3 + s.constructionDir - \Delta(d_p, p, s)) \bmod 6$ ▷ clockwise index of $p$ with
            respect to SEED $s$
        if $i = 5$: $i \leftarrow 4$
        $p.state \leftarrow$ RETIRED
        $p.constructionDir \leftarrow \Delta((s.constructionDir - i - 2) \bmod 6, s, p)$
    else:
        $d \leftarrow$ $p$'s label for port leading to $p$'s predecessor
        Let $q$ be the particle one label counter-clockwise of $p$'s predecessor at label
            $d_q = (d + 1) \bmod 6$.
        if $q.state =$ RETIRED:
            if $\Delta(q.constructionDir, q, p) = d_q$: ▷ exception of the rule
                $p.constructionDir \leftarrow (d + 2) \bmod 6$
            else:
                $p.constructionDir \leftarrow \Delta(q.constructionDir, q, p)$
            $p.state \leftarrow$ RETIRED
```

function `computeMoveLabel` tries to determine the label in which direction the potential ROOT particle should move next (and returns ϵ if the label could not be determined due to a faulty particle). The auxiliary function `updateMoveDir` tries to update the *moveDir* of the potential ROOT particle and returns TRUE on success.

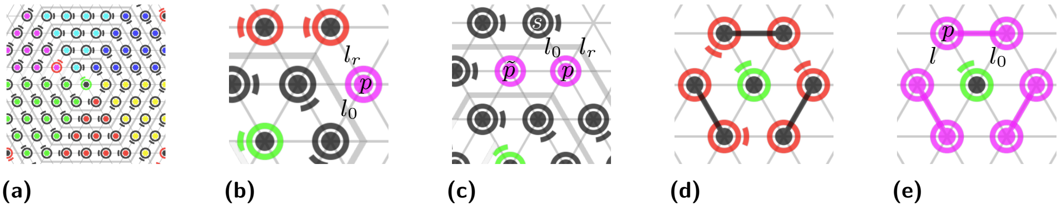


Figure 3 `tryToBecomeRetired`: (a) Particles of the same colour have the same *constructionDir*. A particle beyond the first layer can determine its *constructionDir* from a specific neighbour from the previous layer. `tryToBecomeRoot`: (b) p has a finished neighbour at label l_0, and the particle at label l_r is non-finished and non-faulty. p will become a ROOT with its *moveDir* set to the direction of l_r. (c) Special case: p has successfully determined its *moveDir*. Since p is inside the retired structure, it may not become a ROOT, which is ensured by Lines 5–8. (d)–(e) Lines 14–18 prevent that crashing a cycle of expanded ROOT particles leads to a deadlock.

3.2.4 tryToBecomeFollower

According to the spanning forest primitive an IDLE particle that has a ROOT or FOLLOWER as neighbour, follows it by becoming a FOLLOWER and setting its variable *followDir* appropriately, thereby becoming part of the spanning tree. This is realised by sub-algorithm `tryToBecomeFollower`, which is given as pseudocode in Algorithm 5.

Algorithm 4 `tryToBecomeRoot`.

```
def tryToBecomeRoot($p$):
    if $p$ is contracted:
        $l_r \leftarrow p$.computeMoveLabel()
        if $l_r \neq \epsilon$:
            Let $s$ be the finished particle at label $l_s = (l_r + 1) \bmod 6$.
            $d \leftarrow \Delta(s.constructionDir, s, p)$
            ▷ Check on the basis of $s$ whether $p$ is outside the retired structure:
            if $s$ is the SEED or $(d - l_s) \bmod 6 \leq 2$:
                $p$.updateMoveDir()
                $p.state \leftarrow$ ROOT
    else if $p$ is expanded:
        if $p$.updateMoveDir():
            $p.state \leftarrow$ ROOT
        else if $p$ has a finished neighbour at label $l_0$ with constructionDir pointing at $p$:
            Let $l$ be the in clockwise direction next label after $l_0$ that does not point to
                the same node as $l_0$.
            Update $p.tailDir$ such that $l$ is a head label of $p$.
            $p.moveDir \leftarrow p$'s direction of $l$
            $p.state \leftarrow$ ROOT

def computeMoveLabel($p$):
    if $p$ has a finished neighbour at some label $l_0$:
        Starting at $l_0$ let $l$ be the in clockwise direction first label at whose port there is no
            finished particle.
        if there is no faulty particle at label $l$ :
            return $l$
    return $\epsilon$

def updateMoveDir($p$):
    $l \leftarrow p$.computeMoveLabel()
    if $l \neq \epsilon$:
        Update $p.tailDir$ such that $l$ is a head label of $p$.
        $p.moveDir \leftarrow p$'s direction of $l$
        return true
    return false
```

Algorithm 5 `tryToBecomeFollower`.

```
def tryToBecomeFollower($p$):
    if $p$ has a neighbour $q$ in direction $d$ in state ROOT or FOLLOWER:
        $p.state \leftarrow$ FOLLOWER
        $p.followDir \leftarrow d$
```

3.2.5 tryFollowerRecoveryByPropagation

We now discuss the sub-algorithm `tryFollowerRecoveryByPropagation` (see Algorithm 6), which tries to repair a crashed particle p that could potentially be a follower. Our fault-tolerant solution builds upon two primitives, a *safety primitive* and a *validation primitive*, each of which solves a significant problem: Firstly, we have to make sure that p can actually be a follower, i.e., we have to exclude the possibility that p could be retired. If this is not ensured, various problematic consequences are possible: among other things, the particles may become disconnected (see Figure 4), it is not guaranteed that a hexagon will be built, and it is even possible that the algorithm will no longer terminate. We show that if a particle is in *safeState* safe, it can in principle become a follower. Secondly, we need to ensure that p choosing a follower *parent candidate* (a follower neighbour not following p) does not lead to disconnection of the particles (see Figure 5). In order to avoid disconnection, we use a validation mechanism that determines for a faulty particle which of the follower parent candidates it can attach to without closing a cycle. We will see that these are the previously invalidated follower parent candidates in *pathToRootState* valid.

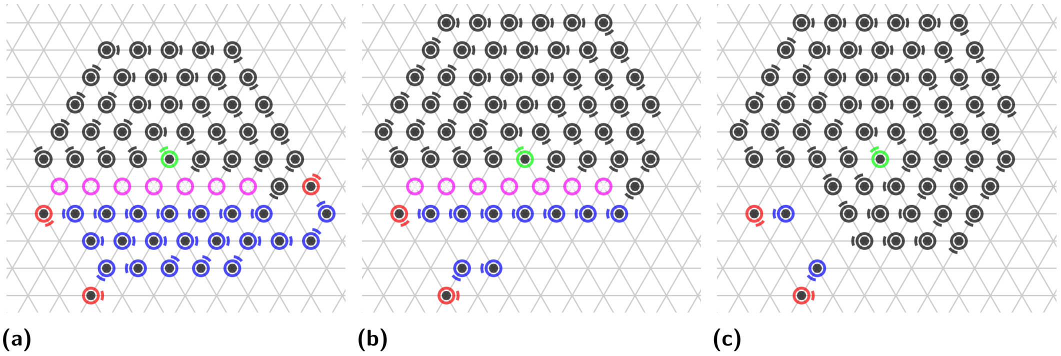


Figure 4 The figure illustrates problems that can arise if it has not been ruled out that a particle to be repaired could be part of the retired structure. (a) Crashed particles have become followers that pass through the structure of originally retired particles. (b) Some followers followed their root and thus split the retired structure. The particles are disconnected. (c) Final stable configuration.

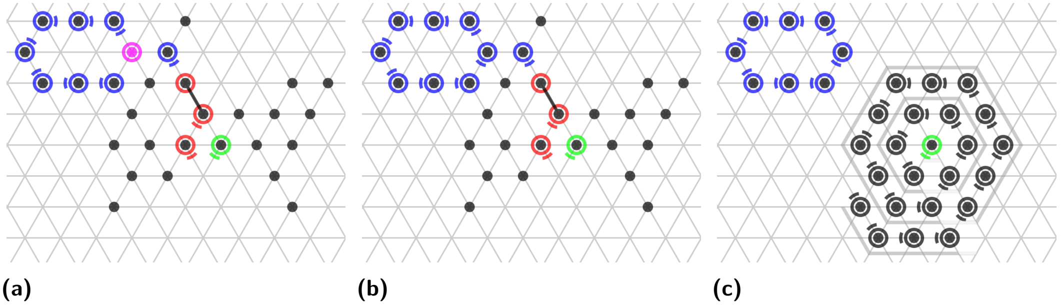


Figure 5 The figure shows that if a crashed particle attaches itself to an arbitrary follower pointing away from it, this can lead to irreversible disconnection of the particles. (a) A particle has crashed. (b) The crashed particle attaches itself to the wrong of the two possible follower candidates, closing a cycle. (c) Final stable configuration in which the particles are disconnected.

Here, we explain how the validation primitive solves the second problem (compare Figure 6). The safety primitive for solving the first problem is explained in Section 3.2.6.

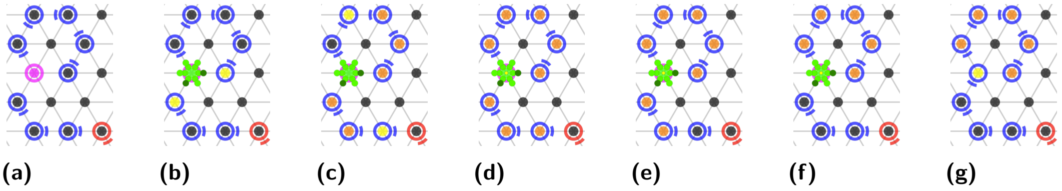


Figure 6 Illustration of the validation primitive. (a) A faulty particle needs to ensure that there is a path to a ROOT before following a particle. (b) The faulty particle sends INVALIDATE tokens to possible parent candidates. (c) INVALIDATE is propagated upwards, causing particles on the path to become INVALID. (d)–(e) An INVALIDATE that reaches an ERROR particle is stored by it, an INVALIDATE that reaches a ROOT is consumed by it. The ROOT generates a VALID token which is propagated downwards along the INVALID particles. (f)–(g) If the VALID token reaches the crashed SAFE particle, it may connect to the FOLLOWER. The INVALIDATE token previously stored by the crashed particle will then again be propagated upwards.

Algorithm 6 `tryFollowerRecoveryByPropagation`.

```
def tryFollowerRecoveryByPropagation($p$):
    if $p$ has a neighbour $q$ at label $l$ with $q.state$ = ROOT or ($q.state$ = FOLLOWER and _
        $q.pathToRootState$ = VALID and $p.hasInvalidated[l]$ = TRUE):
        ▷ Become a FOLLOWER following a ROOT or a previously invalidated VALID
            FOLLOWER parent candidate:
        $p.state \leftarrow$ FOLLOWER
        Update $p.tailDir$ such that $l$ is a head label of $p$.
        $p.followDir \leftarrow$ $p$'s direction of $l$
    else:
        ▷ Check whether new FOLLOWER parent candidates have emerged in the
            neighbourhood in the meantime:
        if $p$ has a FOLLOWER parent candidate $q$ at label $l$ with _
            $p.hasInvalidated[l]$ = FALSE:
            $q.pathToRootState \leftarrow$ INVALIDATE
            $p.hasInvalidated[l] \leftarrow$ TRUE
```

An ERROR particle p in *safeState* SAFE can in principle become a FOLLOWER (see Section 3.2.6). In order to determine an admissible FOLLOWER parent candidate, p employs the following validation mechanism: p invalidates FOLLOWER parent candidates by setting their *pathToRootState* to INVALIDATE and keeps track of which neighbours it has already invalidated in its *hasInvalidated* array, so that each neighbour is invalidated at most once. Subsequently the *pathToRootState* INVALIDATE is propagated “upwards” by followers in the direction of a tree root, such that the particles on the path upwards become INVALID (safety). A special case is when a FOLLOWER in *pathToRootState* INVALIDATE points to an ERROR particle. In this case, the ERROR particle's *pathToRootState* is set to INVALIDATE, but the ERROR particle only continues to propagate if it is repaired to a FOLLOWER. INVALIDATE is finally consumed by a tree root after which the *pathToRootState* VALID is propagated downwards to the leaves (liveness). Note that here it is crucial for safety that an ascending INVALIDATE always has priority over a descending VALID. The propagation of this validation mechanism is realised by the two functions `propagateInvalidate` and `propagateValid` (see Algorithm 7).

In summary, an ERROR particle p may become a FOLLOWER of a neighbour q if $p.safeState$ = SAFE and q is a ROOT or a FOLLOWER parent candidate in *pathToRootState* VALID which was previously invalidated by p.

Algorithm 7 `propagateInvalidate` and `propagateValid`.

```
def propagateInvalidate($p$):
    if $p$ has a FOLLOWER or ERROR parent $q$:
        $q.pathToRootState \leftarrow$ INVALIDATE

def propagateValid($p$):
    for each FOLLOWER child $q$ of $p$ with $q.pathToRootState =$ INVALID:
        $q.pathToRootState \leftarrow$ VALID
```

3.2.6 updateFlags

We now present the sub-algorithm `updateFlags` (see Algorithm 8) which implements the *safety primitive* mentioned in the previous Section 3.2.5. The safety primitive basically ensures that particles inside the retired structure (the hexagon built so far) cannot become FOLLOWER particles. The primitive is explained in Figure 7.

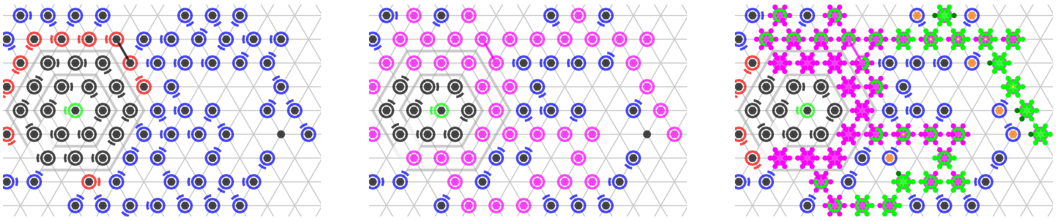

(a) (b) (c)

Figure 7 Illustration of the safety primitive. Crashed particles initially are UNDETERMINED and will become either SAFE or UNSAFE. Crashed particles that are connected to a finished particle via one or two line segments in $G_\triangle$ become UNSAFE, otherwise SAFE by the propagation of *safeFlags*. An UNSAFE particle cannot become a FOLLOWER.

Algorithm 8 `updateFlags`.

```
def updateFlags($p$):
    if $p$ has a finished neighbour: ▷ Initialize $p.safeState$ based on $p$'s neighbours:
        $p.safeState \leftarrow$ UNSAFE
    else:
        $p.safeState \leftarrow$ SAFE

    for each round $r \in [2]$: ▷ Update $p.safeFlags$ and $p.safeState$ for two rounds:
        for each direction $d \in [6]$:
            $S \leftarrow \{\, q.safeFlags[r][d_q] \mid q$ is an ERROR neighbour of $p$ in direction $d \,\}$
            $p.safeFlags[r][d] \leftarrow \min(\{p.safeState\} \cup S)$

        $p.safeState \leftarrow \min \{\, p.safeFlags[r][d] \mid d \in [6] \,\}$
```

3.2.7 performMovement

FOLLOWER and ROOT particles move according to the rules in sub-algorithm `performMovement` (see Algorithm 9), which are essentially the same as the movement rules for the classical hexagon shape formation algorithm `Hexagon`, with the following differences: The definition of a *blocking tail neighbour* includes faulty particles. This ensures that the contraction of a particle does not disconnect faulty particles from the rest of the particle system. To ensure that INVALIDATE does not propagate downwards, a particle cannot push a particle that is in *pathToRootState* INVALIDATE. To maintain property F2 for segments (Lemma 4), a pushing FOLLOWER takes over the *pathToRootState* of the pushed particle. A ROOT particle can only move if its *moveDir* was successfully updated by sub-algorithm `updateMoveDir`.

Algorithm 9 `performMovement`.

```
def performMovement($p$):
    if $p$ is an expanded FOLLOWER or ROOT that has no blocking tail neighbour:
        $p$ contracts into its head.
    else if $p$ is a contracted tail FOLLOWER of a FOLLOWER or ROOT particle $q$:
        if $q.pathToRootState \neq$ INVALIDATE: ▷ Ensure that INVALIDATE does not
                propagate downwards.
            $p$ pushes $q$ and updates $p.followDir$ accordingly.
            $p.pathToRootState \leftarrow q.pathToRootState$
    else if $p$ is a contracted ROOT and $p$.updateMoveDir():
        if $p$ has no neighbour in direction $p.moveDir$:
            $p$ expands in direction $p.moveDir$.
        else if $p$ has the tail of an expanded ROOT in direction $p.moveDir$:
            $p$ pushes in direction $p.moveDir$.
```

3.3 Analysis

In this section we show that our algorithm `HexagonFT` solves the hexagon shape formation problem **HEX**, if a finite number of crashes occur. Due to space constraints, most proofs are given in the appendix in Appendix A.1. We assume w.l.o.g. that the SEED is positioned on $(0,0) \in V_\triangle$ and has orientation 0. Our algorithm constructs the hexagon in a spiral ring around the SEED. A particle that retires sets the direction *constructionDir* for the next particle to become RETIRED. The *constructionDir* for a particle beyond the first layer is determined by the *constructionDir* of a specific neighbouring particle in the previous layer (compare algorithm `tryToBecomeRetired` and Figure 3a). The following lemma provides the basis for the correctness of this procedure:

▶ **Lemma 1.** *For $v \in V_\triangle$ and $d \in [6]$, let $n(v,d) \in V_\triangle$ denote the neighbour of v in direction d. The sequence of nodes v_i defined inductively by $v_0 = (0,0) \in V_\triangle$, $v_k = n(v_{k-1}, d(v_{k-1}))$, $d(v_k) = (0,4,3,2,1,0,0,5)$ for $0 \leq k \leq 7$, $d_k = d(n(v_{k-1}, (d(v_{k-1}) - 1) \bmod 6))$ for $k > 7$ is well-defined, and it holds: For any $k \geq 0$ the nodes $v_0, \dots, v_k$ form a* spiral ring *around the centre $(0,0)$ such that the set $\{v_j | 0 \leq j \leq k\}$ forms a hexagon with centre $(0,0)$.*

▶ **Definition 1.** *For a configuration C let $k \in \mathbb{N}_0$ such that for all $j \in [k]$ the node v_j of the spiral ring is occupied by a finished or faulty particle, and node v_{k+1} is empty or occupied by a non-finished or non-faulty particle. We call the set of particles that occupy the nodes v_j for $j \in [k]$ the* retired structure*. A particle inside (outside) the retired structure is called an* interior *(*exterior*) particle. An exterior particle adjacent to an interior particle is called a* boundary *particle.*

▶ **Definition 2.** *Based on a configuration C, we define a directed graph $A(C)$ as follows: The node set of $A(C)$ is the set of particles. If p is a* FOLLOWER *following a particle q that is a* FOLLOWER*, a* ROOT *or a faulty exterior particle, then $A(C)$ contains a directed edge from p to q. If p is a faulty exterior particle and q a neighbour of p that is a* FOLLOWER *parent candidate of p, a* ROOT *or a faulty exterior particle, then $A(C)$ contains a directed edge from p to q. A simple path or simple circuit $p_1, \ldots, p_k$, $k > 1$ in $A(C)$ from a faulty exterior particle p_1 to a faulty exterior particle p_k, where all p_j for $1 < j < k$ are* FOLLOWER *particles, is called a* segment*. We call a particle a* tree root *if it is a* FOLLOWER *following an interior particle, a* ROOT *or a faulty boundary particle.*

3.3.1 Safety

▶ **Lemma 2** (Retired structure).

R1 *At any time, all finished particles are interior particles.*

R2 *At any time, a finished particle occupying a node v_j of the spiral ring has its* constructionDir *set in the direction of the node v_{j+1}.*

R3 *An interior particle will always remain an interior particle.*

▶ **Lemma 3** (Roots). *At any time, the following* root property *holds: The head of a* ROOT *particle is adjacent to an interior particle.*

▶ **Lemma 4** (Followers). *At any time, the following properties hold:*

F1 *For a* FOLLOWER *particle p, there is a simple path in $A(C)$ from p to a tree root.*

F2 *If $p_1, \ldots, p_k$ is a segment and l the label of p_1 for the port leading to p_2, then one of the two following statements holds:*

a. p_1.state $=$ *CRASHED or (p_1.state $=$ ERROR and p_1.hasInvalidated$[l]$ $=$ FALSE) or*

b. p_1.state $=$ *ERROR and p.hasInvalidated$[l]$ $=$ TRUE, and one of the three following statements holds:*

i. *There exists $1 < j < k$ such that p_i.pathToRootState $=$ INVALID for all $1 < i < j$ and p_j.pathToRootState $=$ INVALIDATE.*

ii. *p_i.pathToRootState $=$ INVALID for all $1 < i < k$ and (p_k crashed after p_1 or p_k.pathToRootState $=$ INVALIDATE)*

iii. *There exists $1 < j < k$ such that p_i.pathToRootState $=$ INVALID for all $1 < i < j$ and p_j.pathToRootState $=$ VALID, and there is a simple path in $A(C)$ from p_j to a tree root, on which all faulty particles crashed after p_1.*

▶ **Lemma 5** (Connectivity). *At any time, all particles are connected, i.e., $G(C)$ is one connected component.*

▶ **Lemma 6.** *Every connected component of* IDLE *particles is connected to at least one non-*IDLE *particle.*

3.3.2 Liveness: Recovery

▶ **Lemma 7** (Recovery). *If a finite number of crashes occur during the execution of algorithm* `HexagonFT` *and m particles are faulty after the last crash, then a non-faulty configuration is reached within $\mathcal{O}(mn)$ rounds after the last crash.*

Proof. We show that within $\mathcal{O}(n)$ rounds at least one faulty particle recovers. Assume that at the beginning of round t all faulty particles are in *state* ERROR. Let C be the configuration at the beginning of round t. We consider the following cases at the beginning of round t, assuming that there is still at least one faulty particle:

Case 1: There is a faulty interior particle: Starting at the SEED, let p be the first faulty interior particle in the spiral ring around the SEED. Clearly, p will become RETIRED within one round.

Case 2: There is no faulty interior particle but a faulty boundary particle: Let v be the first node in the spiral ring around the SEED occupied by a faulty particle p and u the predecessor node of v in the spiral ring. When p is activated in round t, node u is either empty or occupied by a non-faulty particle. If p is contracted and u occupied by a finished particle, p retires. Otherwise, p becomes a ROOT.

Case 3: There is no faulty interior or boundary particle but a faulty exterior non-boundary particle q that has a ROOT or a FOLLOWER as neighbour: We only consider the case that q has a FOLLOWER q' as neighbour. The proof for the case that q has a ROOT as neighbour is analogous. By F1 there is a simple path in $A(C)$ from q' to a tree root. If there is no faulty particle on that path, then let $p = q$ and $p' = q'$, otherwise, let p be the faulty particle on that path nearest to the tree root and p' its successor occupying some node u adjacent to p. Let l be a port label of p pointing towards u. Let $k = 4$. Suppose no faulty particle recovers within kn rounds. By Lemma 5, p is part of a connected component F of faulty particles. The component F does not change within kn rounds since no particles crash and no faulty particles recover during this period. At the beginning of round t no particle in F has a interior or boundary particle as neighbour. This also does not change within kn rounds, as we assume that no faulty particles recover during this period. It follows that *p.safeState* is SAFE at the beginning of round $t + t_1$ for some $t_1 \leq 2n$. Note that from round $t + t_1$ onwards, node u must be occupied by a FOLLOWER parent candidate for p that lies on a path in $A(C)$ of non-faulty particles up to a tree root. We can make the following observations: Since no faulty particle recovers within kn rounds, no new FOLLOWER parent candidates emerge in the neighbourhood of faulty particles. The *pathToRootState* INVALIDATE is propagated upwards by FOLLOWER particles. Therefore, after at most n rounds there are no FOLLOWER particles in *pathToRootState* INVALIDATE. It follows, since then the *pathToRootState* VALID propagates downwards, that after at most n further rounds the node u is occupied by a FOLLOWER parent candidate p'' for p in *pathToRootState* VALID. Since *p.hasInvalidated*$[l]$ = TRUE, it follows that p becomes a FOLLOWER after a total of at most kn rounds, i.e. recovers, a contradiction. Therefore, at least one faulty particle recovers within kn rounds.

Case 4: There is no faulty interior or boundary particle and no faulty exterior non-boundary particle that has a ROOT or a FOLLOWER as neighbour but a faulty particle that has an IDLE neighbour: By Lemma 5, the graph $G(C)$ is connected. Let P be any simple path in $G(C)$ from the SEED particle to some faulty particle that has an IDLE neighbour. Let r be the first faulty particle on P, i.e. the one closest to the SEED on the path P, and let q be its predecessor. Clearly, q is an IDLE particle and will become non-IDLE after $O(n)$ rounds. If r has not yet recovered by this time, the statement now follows with one of the previous cases. ◀

For the upper bound given in Lemma 7, there is a matching (algorithm-depended) lower bound: If all n particles lie on a line and all particles except the SEED are CRASHED, then the algorithm `HexagonFT` needs $\Omega(n^2)$ rounds until a non-faulty configuration is reached.

3.3.3 Termination

We now state our main result:

▶ **Theorem 1.** *If a finite number of crashes occur, then the algorithm* `HexagonFT` *solves the hexagon shape formation problem* **HEX** *in worst-case $\mathcal{O}(n^2)$ work (total number of moves executed by all particles). From the time when no more crashes occur and the configuration is non-faulty, the algorithm needs $\mathcal{O}(n)$ rounds until termination.*

4 Conclusion and Future Work

We have shown that our algorithm `HexagonFT` solves the hexagon shape formation **HEX** in our new model. `HexagonFT` can easily be transferred to other shapes (e.g. square, triangle, line), and the fault-tolerant implementation of the spanning forest primitive can be used for algorithms that employ it. We have assumed that there is a unique SEED particle in the initial configuration that does not fail. We show that this assumption is justified in a manuscript in progress in which we prove that the leader election problem can be solved in our fault tolerance model. Our result is based on the assumption of a finite number of particle crashes. We have extended our simulation environment *AmoebotSim* with the fault tolerance model presented here and implemented our fault-tolerant hexagon shape formation algorithm in it. In our simulations we observed that `HexagonFT` terminates successfully even if particle crashes occur continuously, as long as the rate of crashes is not too high. It could be interesting to investigate under what assumptions the algorithm terminates, if crashes occur continuously. Although the upper bound given in Lemma 7 has a matching algorithm-dependent lower bound, determining a problem-dependent lower bound remains an open problem. It might be interesting to transfer our ideas to other problems in the amoebot model, for example: coating problems [9, 5, 8], bridging problems [1] or general shape formation problems. In view of [11, 15], the question arises, how to deal with particles that permanently crash, i.e. cannot recover. Further research could investigate model variants, e.g. hybrid models in which failures can be both temporary and permanent, models in which the system is already not well-initialised to begin with, and models in the context of self-stabilization. Furthermore, we propose to investigate how a framework could look like that makes it possible to transform any algorithm A without error states into an equivalent algorithm A' with error states that is robust w.r.t. an adversary as specified above.

References

1 Marta Andrés Arroyo, Sarah Cannon, Joshua J Daymude, Dana Randall, and Andréa W Richa. A stochastic approach to shortcut bridging in programmable matter. *Natural Computing*, 17(4):723–741, 2018.

2 Joshua J Daymude, Andréa W Richa, and Christian Scheideler. The canonical amoebot model: Algorithms and concurrency control. In *35th International Symposium on Distributed Computing*, 2021.

3 Joshua J Daymude, Andréa W Richa, and Jamison W Weber. Bio-inspired energy distribution for programmable matter. In *International Conference on Distributed Computing and Networking 2021*, pages 86–95, 2021.

4 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot–a new model for programmable matter. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pages 220–222, 2014.

5 Zahra Derakhshandeh, Robert Gmyr, Alexandra Porter, Andréa W Richa, Christian Scheideler, and Thim Strothmann. On the runtime of universal coating for programmable matter. In *International Conference on DNA-Based Computers*, pages 148–164. Springer, 2016.

6 Zahra Derakhshandeh, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, pages 1–2, 2015.

7 Zahra Derakhshandeh, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 289–299, 2016.

8 Zahra Derakhshandeh, Robert Gmyr, Andréa W Richa, Christian Scheideler, and Thim Strothmann. Universal coating for programmable matter. *Theoretical Computer Science*, 671:56–68, 2017.

9 Zahra Derakhshandeh, Robert Gmyr, Andréa W Richa, Christian Scheideler, Thim Strothmann, and Shimrit Tzur-David. Infinite object coating in the amoebot model. *arXiv preprint*, 2014. arXiv:1411.2356.

10 Giuseppe A Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 33(1):69–101, 2020.

11 Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Giovanni Viglietta. Line recovery by programmable particles. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–10, 2018.

12 Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Mobile ram and shape formation by programmable particles. In *European Conference on Parallel Processing*, pages 343–358. Springer, 2020.

13 Yuval Emek, Shay Kutten, Ron Lavi, and William K Moses Jr. Deterministic leader election in programmable matter. *arXiv preprint*, 2019. arXiv:1905.00580.

14 Irina Kostitsyna, Christian Scheideler, and Daniel Warner. Brief Announcement: Fault-Tolerant Shape Formation in the Amoebot Model. In James Aspnes and Othon Michail, editors, *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*, volume 221 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:3, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAND.2022.23.

15 Nooshin Nokhanji and Nicola Santoro. Line reconfiguration by programmable particles maintaining connectivity. In *International Conference on the Theory and Practice of Natural Computing*, pages 157–169. Springer, 2020.

16 Tommaso Toffoli and Norman Margolus. Programmable matter: concepts and realization. *Physica. D, Nonlinear phenomena*, 47(1-2):263–272, 1991.

A Appendix

A.1 Omitted Proofs

In this section we give all the proofs that have been omitted from the main part due to space constraints in a rigorous form.

Proof of Lemma 1. Define $v_{0,0} = (0,0) \in V_\triangle$, $d_{0,0} = 0$, $d_{1,k} = (4,3,2,1,0,0)_{1 \leq k \leq 6}$ and $d_{2,1} = 5$. Inductively define $\pi(l,k) = (l-1, 6(l-1))$ for $l \geq 1$, $k = 1$, $\pi(l,k) = (l, k-1)$ for $l \geq 1$, $1 < k \leq 6l$, $d_{l,1} = d_{l-1,1}$ for $l > 2$, $d_{l,jl+i} = d_{l-1,j(l-1)+(i-1)}$ for $l \geq 2$, $0 \leq j \leq 5$, $2 \leq i \leq l$, $d_{l,jl+1} = d_{l-1,j(l-1)}$ for $l \geq 2$, $1 \leq j \leq 5$, $v_{l,k} = n(v_{\pi(l,k)}, d_{\pi(l,k)})$ for $l \geq 1$, $1 \leq k \leq 6l$, $\sigma(v_{l,k}) = (l,k)$ for $l \geq 1$, $1 \leq k \leq 6l$ and $\rho(l,k) = n(v_{\pi(l,k)}, (d_{\pi(l,k)} - 1) \bmod 6)$ for $l \geq 2$, $1 \leq k \leq 6l$, $(l,k) \neq (2,1)$. By induction the following properties follow: $\rho(l,1) = v_{l-1,1}$ for $l > 2$, $\rho(l, jl+i) = v_{l-1,j(l-1)+(i-1)}$ for $l \geq 2$, $0 \leq j \leq 5$, $2 \leq i \leq l$, $\rho(l, jl+1) = v_{l-1,j(l-1)}$ for $l \geq 2$, $1 \leq j \leq 5$ and $d_{l,k} = (5^{l-1}, 4^l, 3^l, 2^l, 1^l, 0^{l+1})_{1 \leq k \leq 6l}$ for $l \geq 1$. This implies $d_{l,k} = d_{\sigma(\rho(l,k))}$ for $l \geq 2$, $1 \leq k \leq 6l$, $(l,k) \neq (2,1)$ and that the nodes $v_{l,1}, \ldots, v_{l,6l}$ for $l \geq 1$ form a single ring of radius l around the centre $(0,0) \in V_\triangle$ with corners $v_{l,jl}$ for

$1 \leq j \leq 6$. The nodes in the defined order therefore form a spiral ring around the centre $(0,0)$, in particular for any $l \geq 1$ and $1 \leq k \leq 6l$ the set of all predecessors of $v_{l,k}$ forms a hexagon with centre $(0,0)$. ◀

▶ **Lemma 2** (Retired structure).

R1 *At any time, all finished particles are interior particles.*

R2 *At any time, a finished particle occupying a node v_j of the spiral ring has its* constructionDir *set in the direction of the node v_{j+1}.*

R3 *An interior particle will always remain an interior particle.*

Proof. In an initial configuration the properties hold trivially. Assume the properties hold for a configuration C. Consider that **a particle p becomes retired (Line 13):** Assume that `tryToBecomeRetired` will change p's *state* to RETIRED. Let $\tilde{p}$ be the finished direct predecessor of p. By R1, $\tilde{p}$ is an interior particle occupying some node v_j of the spiral ring. By R2, p occupies node v_{j+1} of the spiral ring. The structure of the spiral ring is precisely characterised by Lemma 1 and implemented accordingly in `tryToBecomeRetired`: The particles in the first layer of the hexagon are neighbours of the SEED. If p has the SEED as neighbour, then *p.constructionDir* is uniquely determined by the *constructionDir* of the SEED and the position of p relative to the SEED (Lines 3–6). If p is a particle beyond the first layer of the hexagon, then *p.constructionDir* is, with one exception (Line 12), determined by the previous layer of the hexagon (Line 14, Figure 3a; the exception is the first particle in the second layer; the particle is highlighted with a red border). In any case *p.constructionDir* is set correctly in the direction of the node v_{j+2}. Therefore, all properties still hold after the state change. Now we show that **an interior particle will always remain an interior particle:** Suppose the contrary. Let t_1 be the first time when an interior particle p gets activated and subsequently is no interior particle. Since finished particles do nothing and CRASHED particles become initialized by `initializeAfterCrash`, we can assume that p is in *state* ERROR. Note that all predecessor nodes of the node occupied by p in the spiral ring around the SEED are occupied by finished or faulty particles. At the end of the activation, p is no interior particle; we distinguish the following *state* changes (Line 11):

Case 1: p becomes a ROOT (Line 17): Assume that `tryToBecomeRoot` will change p's *state* to ROOT. Note that p cannot be the successor of the SEED. Since $l_r \neq \epsilon$, p cannot be entirely surrounded by faulty or finished particles. Let $\tilde{p}$ be the direct predecessor of p in the spiral ring.

Case a: $\tilde{p}$ is not entirely surrounded by faulty or finished particles: Then, starting at any finished neighbour of p, all in clockwise direction traversed neighbouring nodes of p up to and including $\tilde{p}$ are occupied by finished or faulty particles. Since p cannot retire, $\tilde{p}$ must be faulty. But from this follows $l_r = \epsilon$, a contradiction.

Case b: $\tilde{p}$ is entirely surrounded by faulty or finished particles (Figure 3c): By an analogous argument as in the previous case, the finished particle s (Line 5) must be the last interior particle in the spiral ring. Since $(d - l_s) \bmod 6 \leq 2$, p is one or two labels counter-clockwise of *s.constructionDir* w.r.t. s, i.e., p is outside of the retired structure, a contradiction.

Case 2: p becomes a FOLLOWER (Line 23): Note that the *state* change in `tryFollowerRecoveryByPropagation` only takes place if *p.safeState* = SAFE holds. So we must have *p.safeState* = SAFE after the call of *p*.`updateFlags`. By u denote the node occupied by p and by w the node occupied by the SEED. One of the following two statements holds:

Case a: There exists a line segment L in $G_\triangle$ from u to w such that all nodes (except u) on L are predecessors of u in the spiral ring around the SEED.

Case b: There exists a node v and line segments L_1, L_2 in $G_\triangle$ from u to v, and from v to w, respectively, such that all nodes (except u) on L_1 and L_2 are predecessors of u in the spiral ring around the SEED.

We only consider the latter case; in the former case, the proof is analogous. Let $p = p_1, \ldots, p_j$ and $p_j, \ldots, p_k = \text{SEED}$ be the interior particles occupying the nodes of the line segments L_1 and L_2, respectively. Note that p_{i+1} is a predecessor of p_i in the spiral ring for all $1 \leq i < k$. To simplify the notation, we assume in the following that local and global directions correspond. By d_1 and d_2 denote the direction of L_1 and L_2, respectively. We argue in the following by means of backward analysis: At time t_1 particle p_2 cannot be finished or in *state* CRASHED, since otherwise $p.\mathit{safeState} \neq$ SAFE after the call of p.`updateFlags`. Therefore, p_2 must be in *state* ERROR with $p_2.\mathit{safeFlags}[1][d_1] =$ SAFE. Let t_2 be the time before t_1 when p_2 was last activated. At time t_2 particle p_3 cannot be finished or in *state* CRASHED, since otherwise $p_2.\mathit{safeFlags}[1][d_1] \neq$ SAFE. Therefore, p_3 must be in *state* ERROR with $p_3.\mathit{safeFlags}[1][d_1] =$ SAFE. By repeated application of the argument, it follows that at some time t_{j-1}, p_j must be in *state* ERROR with $p_j.\mathit{safeFlags}[1][d_1] =$ SAFE, and therefore $p_j.\mathit{safeFlags}[0][d_2] =$ SAFE. Further repeated application of the argument finally yields that at some time t_{k-2}, p_{k-1} must be in *state* ERROR with $p_{k-1}.\mathit{safeFlags}[0][d_2] =$ SAFE, contradicting that p_{k-1} has the finished neighbour $p_k =$ SEED. ◀

▶ **Lemma 3** (Roots). *At any time, the following* root property *holds: The head of a* ROOT *particle is adjacent to an interior particle.*

Proof. In an initial configuration the root property holds trivially. Assume the root property holds for a configuration C. By C' denote the successor configuration of C. One can easily check that a particle p becomes a ROOT by `tryToBecomeRoot` (`HexagonFT`: Line 17) only if it has a finished neighbour q, and that `tryToBecomeRoot` ensures that the head of p is adjacent to q. By R1, q is an interior particle. Therefore, the root property holds for p in C'. Now let p be a ROOT in C. By R3, an interior particle will always remain an interior particle. Therefore, the root property for p could only be violated in C' by movement of p (`HexagonFT`: Line 32):

Case 1: p is an expanded FOLLOWER or ROOT that has no blocking tail neighbour: Since p contracts into its head, the root property holds for p in C'.

Case 2: p is a contracted tail FOLLOWER of a FOLLOWER or ROOT particle q: Since p pushes q into its head, the root property holds for q in C'.

Case 3: p is a contracted ROOT and `updateMoveDir`(p) succeeds: In both subcases: After the expansion, the position of p's head has changed. Due to the choice of label by `computeMoveLabel` and making it a head label, p still has a finished (and therefore interior) particle adjacent to its head in C', so the root property holds for p in C'. ◀

▶ **Lemma 4** (Followers). *At any time, the following properties hold:*

F1 *For a* FOLLOWER *particle* p, *there is a simple path in* $A(C)$ *from* p *to a tree root.*

F2 *If* $p_1, \ldots, p_k$ *is a segment and* l *the label of* p_1 *for the port leading to* p_2, *then one of the two following statements holds:*

- **a.** p_1.state $=$ *CRASHED or* (p_1.state $=$ *ERROR and* p_1.hasInvalidated$[l] =$ *FALSE*) *or*
- **b.** p_1.state $=$ *ERROR and* p.hasInvalidated$[l] =$ *TRUE, and one of the three following statements holds:*
  - **i.** *There exists* $1 < j < k$ *such that* p_i.pathToRootState $=$ *INVALID for all* $1 < i < j$ *and* p_j.pathToRootState $=$ *INVALIDATE.*

ii. p_i.pathToRootState $=$ INVALID *for all* $1 < i < k$ *and (*p_k *crashed after* p_1 *or* p_k.pathToRootState $=$ INVALIDATE*)*

iii. *There exists* $1 < j < k$ *such that* p_i.pathToRootState $=$ INVALID *for all* $1 < i < j$ *and* p_j.pathToRootState $=$ VALID*, and there is a simple path in* $A(C)$ *from* p_j *to a tree root, on which all faulty particles crashed after* p_1*.*

Proof. In an initial configuration the properties holds trivially. Assume the properties holds for a configuration C. We show in the following that the properties remain true for the successor configuration C' of C resulting from a crash or a configuration change by the algorithm. **Particle crash:** Clearly, F1 remains true. For F2 assume that an exterior particle p crashes and let $S = (p_1, \ldots, p_k)$ be a segment in $A(C')$. We consider the following cases:

Case 1: p is not a particle on segment S: Clearly, F2 holds for S in C'.

Case 2: $p = p_1$: F2a holds for S in C'.

Case 3: $p = p_k$ and $p_1 \neq p_k$: It is sufficient to show that in the case of F2b one of the three properties holds for S in C'.

Case a: p_i*.pathToRootState* $=$ INVALID for all $1 < i < k$: F2(b)i holds for S in C'.

Case b: There exists $1 < j < k$ such that p_i*.pathToRootState* $=$ INVALID for all $1 < i < j$ and p_j*.pathToRootState* $\neq$ INVALID: If p_j*.pathToRootState* $=$ INVALIDATE, then F2(b)i holds for S in C'. Assume p_j*.pathToRootState* $=$ VALID. By F1, there is a simple path P in $A(C)$ from p_j to a tree root.

Case: There is no faulty particle on P: F2(b)iii holds for S in C'.

Case: There is a faulty particle on P: Let q be the first faulty particle on P. The path from p_1 to q in $A(C)$ is a segment for which F2(b)iii holds in C. Together with the fact that p_k crashed after p_1, it follows that there is a simple path in $A(C')$ from p_j to a tree root, on which all particles crashed after p_1. Therefore, F2(b)iii holds for S in C'.

Initialization of crashed particle: Assume a CRASHED particle p of a segment calls `initializeAfterCrash` (`HexagonFT`, Line 5). After the initialization it holds *p.state* $=$ ERROR and *p.hasInvalidated*$[l]$ $=$ FALSE for all port labels l. It is immediately apparent that all properties still hold. **State change (`HexagonFT`, Lines 11–21):**

Case 1: p becomes a ROOT or RETIRED: Consider F1: For a FOLLOWER q, there is a simple path in $A(C)$ to a tree root. If a particle p on the path becomes a ROOT or RETIRED, two cases are possible: the path is shortened or split into two paths, in both cases F1 still applies. For F2 consider that a particle p of a segment becomes a ROOT or RETIRED. This results in the segment no longer being a segment, and thus F2 trivially holds.

Case 2: p becomes a FOLLOWER by `tryToBecomeFollower`: It is obvious that F1 and F2 will still hold after p has become a FOLLOWER.

State change (`HexagonFT`, Lines 22–23): *p tries to become a* FOLLOWER *by* `tryFollowerRecoveryByPropagation`*:*

Case: if-branch: Let C' be the successor configuration of C after p became a FOLLOWER following q.

Case: q is not on a segment in $A(C)$: By F1, there is a simple path $P_q = (q = q_2, \ldots, q_r)$, $r > 1$ in $A(C)$ from q to a tree root q_r. By the definition of a segment, all particles q_i for $2 \leq i \leq r$ must be non-faulty. Therefore, $P_p = (p = q_1, q = q_2, \ldots, q_r)$ is a simple path from p to a tree root in $A(C')$; in particular, F1 holds for p in C'. *Consider F1*: Let $\tilde{p} \neq p$ be any FOLLOWER particle. By F1, there is a simple path $P_{\tilde{p}}$ in $A(C)$ from $\tilde{p}$ to a tree root. We only need to consider the case that p lies on $P_{\tilde{p}}$. Connecting the subpath of $P_{\tilde{p}}$ from $\tilde{p}$ to p with the path P_p results in a path in $A(C')$ from $\tilde{p}$ to a tree root. *Consider F2*: Let $p_1, \ldots, p_k$ be a segment in $A(C')$ and observe that it is also a segment in $A(C)$. Particle p becoming a FOLLOWER could only have an effect on the validity of

F2(b)iii: By F2(b)iii, there is a simple path P_{p_j} in $A(C)$ from p_j to a tree root, on which all faulty particles crashed after p_1. We only need to consider the case that p lies on P_{p_j}. Connecting the subpath of P_{p_j} from p_j to p with the path P_p results in a path in $A(C')$ from p_j to a tree root, on which all faulty particles crashed after p_1.

Case: q is on a segment $p = q_1, q = q_2, \ldots, q_k$, $k > 1$ in $A(C)$: Note that q is a FOLLOWER parent candidate of p with $q.pathToRootState$ = VALID. By F2(b)iii, there is a simple path $P_q = (q = q_2, \ldots, q_r)$, $r \geq k$ in $A(C)$ from q to a tree root q_r, on which all faulty particles crashed after p. Therefore, $P_p = (p = q_1, q = q_2, \ldots, q_r)$ is a simple path from p to a tree root in $A(C')$; in particular, F1 holds for p in C'. *Consider F1*: Let $\tilde{p} \neq p$ be any FOLLOWER particle. By F1, there is a simple path $P_{\tilde{p}}$ in $A(C)$ from $\tilde{p}$ to a tree root. We only need to consider the case that p lies on $P_{\tilde{p}}$. Connecting the subpath of $P_{\tilde{p}}$ from $\tilde{p}$ to p with the path P_p results in a path in $A(C')$ from $\tilde{p}$ to a tree root.

Consider F2: Let $S' = (p_1, \ldots, p_{k'})$ be a segment in $A(C')$.

Case 1: p is not a particle on segment S': Particle p becoming a FOLLOWER could only have an effect on the validity of F2(b)iii: By F2(b)iii, there is a simple path P_{p_j} in $A(C)$ from p_j to a tree root, on which all faulty particles crashed after p_1. We only need to consider the case that p lies on P_{p_j}. Connecting the subpath of P_{p_j} from p_j to p with the path P_p results in a path in $A(C')$ from p_j to a tree root, on which all faulty particles crashed after p_1.

Case 2: p is a particle on segment S': We have $p = p_{k''}$ for some $1 < k'' < k'$. Note that $S = (p_1, \ldots, p_{k''})$ is a segment in $A(C)$. It is sufficient to show that in the case of F2b one of the three properties holds for S' in C'.

Case a: F2(b)i holds for S in $A(C)$: Clearly, F2(b)i holds for S' in $A(C')$.

Case b: F2(b)ii holds for S in $A(C)$: If $p_{k''}.pathToRootState$ = INVALIDATE, then F2(b)ii clearly holds for S' in $A(C')$. If $p_{k''}.pathToRootState \neq$ INVALIDATE and $p_{k''}$ crashed after p_1, then F2(b)iii holds for S' in $A(C')$.

Case c: F2(b)iii holds for S in $A(C)$: Clearly, F2(b)iii holds for S' in $A(C')$.

Case: else-branch: It is obvious that the properties will still hold in C'.

Propagation of *pathToRootState* (`HexagonFT`, Lines 25–30): Clearly, F1 remains true. Furthermore, one can easily check that F2 remains true. **Movement (`HexagonFT`, Line 34):** Clearly, F1 remains true. Only the first else-branch of `performMovement` may have an effect on F2. Let p be a contracted tail FOLLOWER of a FOLLOWER or ROOT particle q. Consider Line 7 of Algorithm 9: After p has pushed q, it occupies the node with its head, which q previously had occupied with its tail. The paths to a tree root that previously ran over the tail of q now run over the head of p. Therefore and since propagation has already taken place, p takes on the *pathToRootState* of q. Together with the fact that the branch is only executed if $q.pathToRootState \neq$ INVALIDATE, which ensures that INVALIDATE does not propagate downwards, this guarantees that F2 still holds after the movement. ◀

▶ **Lemma 5** (Connectivity). *At any time, all particles are connected, i.e., $G(C)$ is one connected component.*

Proof. In an initial configuration all particles are connected. Assume the property holds for a configuration C. The property could only be violated by a contraction of a particle, which only occurs in `performMovement` if p is an expanded FOLLOWER or ROOT that has no blocking tail neighbour (Line 3 of Algorithm 9). Let q be a neighbour of p before p contracts into its head and C' the configuration after p contracted. Since p has no blocking tail neighbour, we only need to consider the following cases:

Case 1: q is finished or a ROOT By 2 and the definition of a boundary particle, all finished and boundary particles are connected in C'. By F1, p is connected to a boundary particle in C'. Since q is finished or, by 3, a boundary particle, in C', p and q remain connected.

Case 2: q is a FOLLOWER of some particle other than p Similar to the previous case with the additional observation that, by F1, q is connected to a boundary particle.

Case 3: q is a head follower of p or an IDLE, ERROR or CRASHED particle adjacent to p's head Since p contracts into its head, p and q remain connected. ◀

▶ **Lemma 6.** *Every connected component of* IDLE *particles is connected to at least one non-*IDLE *particle.*

Proof. Consider a connected component H of IDLE particles. If H were not connected to a non-IDLE particle it would follow with 5 that all particles IDLE, contradicting the fact that there is always at least one non-idle particle, namely the SEED. ◀

The following properties are used in the proof of Theorem 1. As in its proof, we assume in the following that there is a time t^* from which on the configuration is legal.

▶ **Lemma 8.** *Eventually, every* IDLE *particle becomes a* FOLLOWER, ROOT *or* RETIRED *particle.*

Proof. Consider a configuration at time $t \geq t^*$. If there is an IDLE particle, then by Lemma 6 there exists an IDLE particle p that is connected to a non-IDLE particle. When p is activated, p becomes a FOLLOWER, ROOT or RETIRED particle. ◀

▶ **Lemma 9.** *Eventually, every expanded particle contracts.*

Proof. Consider a configuration at time $t \geq t^*$. By Lemma 8 we can assume that all particles are non-IDLE. Furthermore, we can assume that all FOLLOWER particles are in *pathToRootState* VALID. Let p be an expanded particle. If p has no tail FOLLOWER, then p can contract. If p has a contracted tail FOLLOWER or, p is a ROOT and has a contracted ROOT with *moveDir* set to p's tail, then p can contract by a handover. If p has an expanded tail FOLLOWER q, then we can apply this argument recursively for a finite number of steps to conclude that q eventually contracts and therefore also p. ◀

▶ **Theorem 1.** *If a finite number of crashes occur, then the algorithm* `HexagonFT` *solves the hexagon shape formation problem* **HEX** *in worst-case $\mathcal{O}(n^2)$ work (total number of moves executed by all particles). From the time when no more crashes occur and the configuration is non-faulty, the algorithm needs $\mathcal{O}(n)$ rounds until termination.*

Proof. By Lemma 7, there is a time from which on the configuration is non-faulty. From the safety properties in Section 3.3.1 the following statements can be derived:

1. Eventually, every IDLE particle becomes a FOLLOWER, ROOT or RETIRED particle. (Lemma 8)
2. Eventually, every expanded particle contracts. (Lemma 9): This implies in particular that as long as ROOT particles move, FOLLOWER particles will eventually follow.
3. A FOLLOWER that has a finished neighbour becomes a ROOT or RETIRED.
4. A ROOT will eventually become RETIRED or expand in direction *moveDir*.
5. All particles eventually become RETIRED.

The proof of these statements is essentially the same as for the analysis of the "classical" hexagon algorithm (cf. e.g. [7, 8]). Since all particles eventually become RETIRED and RETIRED particles do nothing, we eventually reach a stable configuration. Together with Lemma 2 this shows that `HexagonFT` solves **HEX**. ◀