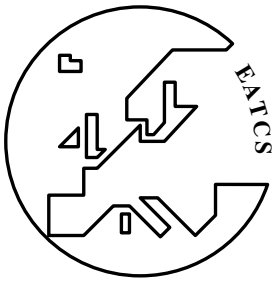
ISSN 0252–9742

Bulletin

of the

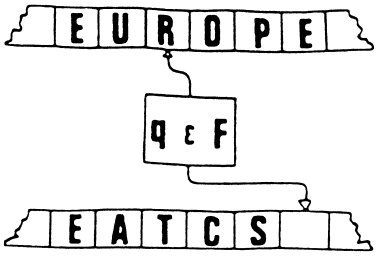
European Association for Theoretical Computer Science

EATCS



Number 140 June 2023

Council of the European Association for Theoretical Computer Science



President:	Artur Czumaj	United Kingdom
Vice Presidents:	Anca Muscholl	France
	Giuseppe F. Italiano	Italy
Treasurer:	Jean-Francois Raskin	Belgium
Bulletin Editor:	Stefan Schmid	Germany

Ivona Bezakova	USA	Anca Muscholl	France
Tiziana Calamoneri	Italy	Luke Ong	UK
Thomas Colcombet	France	Tal Rabin	USA
Artur Czumaj	UK	Eva Rotenberg	Denmark
Javier Esparza	Germany	Maria Serna	Spain
Fabrizio Grandoni	Switzerland	Alexandra Silva	USA
Thore Husfeldt	Sweden, Denmark	Jiri Sgall	Czech Republic
Giuseppe F. Italiano	Italy	Ola Svensson	Switzerland
Fabian Kuhn	Germany	Jukka Suomela	Finland
Slawomir Lasota	Poland	Till Tantau	Germany
Elvira Mayordomo	Spain	Sophie Tison	France
Emanuela Merelli	Italy		

Past Presidents:

Maurice Nivat	(1972–1977)	Mike Paterson	(1977–1979)
Arto Salomaa	(1979–1985)	Grzegorz Rozenberg	(1985–1994)
Wilfred Brauer	(1994–1997)	Josep Díaz	(1997–2002)
Mogens Nielsen	(2002–2006)	Giorgio Ausiello	(2006–2009)
Burkhard Monien	(2009–2012)	Luca Aceto	(2012–2016)
Paul Spirakis	(2016–2020)		

Secretary Office:	Dmitry Chistikov	UK
	Efi Chita	Greece

EATCS Council Members

email addresses

Ivona Bezakova . . . . . ib@cs.rit.edu
Tiziana Calamoneri . . . . . calamo@di.uniroma1.it
Thomas Colcombet . . . . . Thomas.Colcombet@irif.fr
Artur Czumaj . . . . . A.Czumaj@warwick.ac.uk
Javier Esparza . . . . . esparza@in.tum.de
Fabrizio Grandoni . . . . . fabrizio@idsia.ch
Thore Husfeldt . . . . . thore@itu.dk
Giuseppe F. Italiano . . . . . giuseppe.italiano@uniroma2.it
Fabian Kuhn . . . . . kuhn@cs.uni-freiburg.de
Slawomir Lasota . . . . . sl@mimuw.edu.pl
Elvira Mayordomo . . . . . elvira@unizar.es
Emanuela Merelli . . . . . emanuela.merelli@unicam.it
Anca Muscholl . . . . . anca@labri.fr
Luke Ong . . . . . luke.Ong@cs.ox.a.uk
Tal Rabin . . . . . chair.sigact@sigact.acm.org
Jean-Francois Raskin . . . . . jraskin@ulb.ac.be
Eva Rotenberg . . . . . eva@rotenberg.dk
Maria Serna . . . . . mjserna@cs.upc.edu
Stefan Schmid . . . . . stefan.schmid@tu-berlin.de
Alexandra Silva . . . . . alexandra.silva@cornell.edu
Jiri Sgall . . . . . sgall@iuuk.mff.cuni.cz
Ola Svensson . . . . . Ola.Svensson@epfl.ch
Jukka Suomela . . . . . jukka.suomela@aalto.fi
Till Tantau . . . . . tantau@tcs.uni-luebeck.de
Sophie Tison . . . . . sophie.tison@lifl.fr

Bulletin Editor: Stefan Schmid, Berlin, Germany
Cartoons: DADARA, Amsterdam, The Netherlands

The bulletin is entirely typeset by pdfTEX and ConTEXt in TXfonts.

All contributions are to be sent electronically to

bulletin@eatcs.org

and must be prepared in LaTeX 2ε using the class `beatcs.cls` (a version of the standard LaTeX 2ε `article` class). All sources, including figures, and a reference PDF version must be bundled in a ZIP file.

Pictures are accepted in EPS, JPG, PNG, TIFF, MOV or, preferably, in PDF. Photographic reports from conferences must be arranged in ZIP files layed out according to the format described at the Bulletin's web site. Please, consult `http://www.eatcs.org/bulletin/howToSubmit.html`.

We regret we are unfortunately not able to accept submissions in other formats, or indeed submission not *strictly* adhering to the page and font layout set out in `beatcs.cls`. We shall also not be able to include contributions not typeset at camera-ready quality.

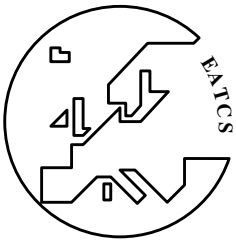
The details can be found at `http://www.eatcs.org/bulletin`, including class files, their documentation, and guidelines to deal with things such as pictures and overfull boxes. When in doubt, email `bulletin@eatcs.org`.

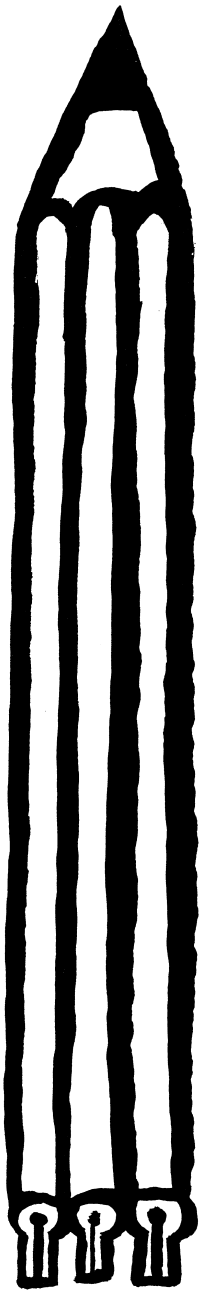
Deadlines for submissions of reports are January, May and September 15th, respectively for the February, June and October issues. Editorial decisions about submitted technical contributions will normally be made in 6/8 weeks. Accepted papers will appear in print as soon as possible thereafter.

The Editor welcomes proposals for surveys, tutorials, and thematic issues of the Bulletin dedicated to currently hot topics, as well as suggestions for new regular sections.

Table of Contents

EATCS Matters





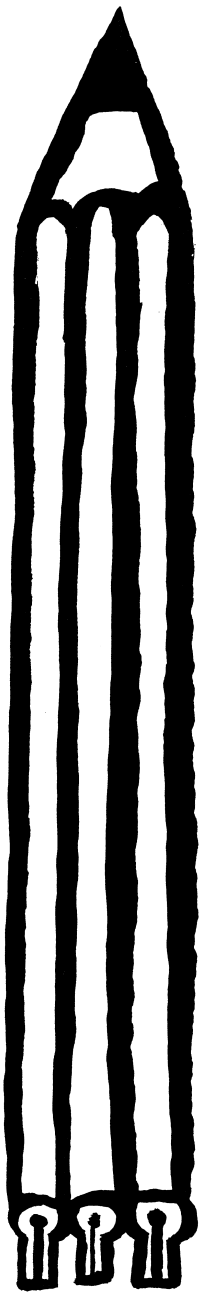
Dear EATCS members,

As usual, the June issue of the Bulletin will be available just before ICALP, the flagship conference of the EATCS and an important meeting of the theoretical computer science community world-wide. The 50th ***EATCS International Colloquium on Automata, Languages, and Programming (ICALP 2023)****, will be held in Paderborn, July 10-14, 2023 (https://icalp2023.cs.upb.de/). I am looking forward to the conference run again in a fully in-person mode. The conference chair Sevag Gharibian, supported by his colleagues in Paderborn, promises us an exciting scientific event. I look forward to the conference and to celebrating the 50th ICALP with you and I hope to see many of you joining us in these celebrations during the ICALP 2023 conference in Paderborn!*

In the scientific part of the ICALP program, the Programme Committee chairs Uriel Feige (track A) and Kousha Etessami (track B) and their PCs have done fantastic job selecting an impressive collection of papers, 103 accepted papers in track A and 29 in track B out of almost 450 submissions (~350 for track A and 97 for track B). The acceptance rate was about 29.5 percent. The programme of ICALP 2023 will highlight research across many areas within theoretical computer science. I invite you to attend and/or watch the talks even outside your own research field.

The ***best paper awards at ICALP 2023*** *will go to the following three papers:*

- *Track A: Tsun-Ming Cheung, Hamed Hatami, Pooya Hatami, and Kaave*



Hosseini, "Online learning and disambiguations of partial concept classes;"

- *Track A: Miguel Bosch Calvo, Fabrizio Grandoni, and Afrouz Jabal Ameli, "A 4/3 approximation for 2-vertex-connectivity;"*
- *Track B: Marvin Künnemann, Filip Mazowiecki, Lia Schütze, Henry Sinclair-Banks, and Karol Węgrzycki, "Coverability in VASS revisited: Improving Rackoff's bound to obtain conditional optimality."*

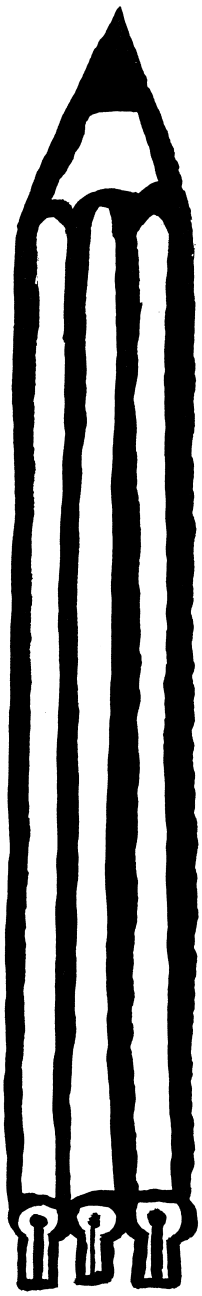
The ***best student paper award*** *(for a paper that is solely authored by a student) will go to the following two papers:*

- *Track A: Manuel Cáceres. "Minimum chain cover in almost linear time;"*
- *Track B: Ruiwen Dong. "The Identity Problem in $\mathbb{Z} \wr \mathbb{Z}$ is decidable."*

Congratulations to the authors of the award-receiving papers!

In addition to regular research talks, ICALP 2023 will feature five invited talks delivered by

- *Anna Karlin (University of Washington, USA),*
- *Rasmus Kyng (ETH Zurich, Switzerland),*
- *Rupak Majumdar (Max Planck Institute for Software Systems, Germany)*
- *Thomas Vidick (California Institute of Technology, USA, and Weizmann Institute of Science, Israel), and*



- *James Worrell (University of Oxford, UK).*

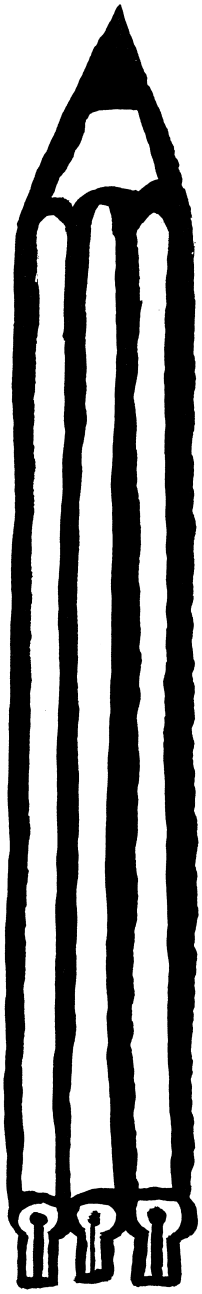
Further two special keynote presentations will be given to celebrate and commemorate the 50th ICALP:

- *Thomas Henzinger (Institute of Science and Technology Austria, Austria) and*
- *Kurt Mehlhorn (Max-Planck-Institut für Informatik, Germany).*

These activities will be complemented by a Colloquium in honor of Friedhelm Meyer auf der Heide taking place at the end of ICALP.

Apart from the invited and contributed talks, ICALP 2023 will feature special presentations of the receipts of the EATCS Award 2023, the EATCS Presburger Award, and of the Alonzo Church Award:

- *The EATCS annually honors a respected scientist from our community with the* ***EATCS Distinguished Achievements Award****, to acknowledge extensive and widely recognized contributions to theoretical computer science over a life long scientific career (http://eatcs.org/index.php/eatcs-award). The recipient of the* ***EATCS Award 2023*** *is* ***Amos Fiat*** *(Tel Aviv University), for fundamental work within many areas of theoretical computer science and in particular for work in cryptography, on-line algorithms, and algorithmic game theory.*
- *The* ***EATCS Presburger Award*** *is awarded by the EATCS to a young scientist for outstanding contributions in theoretical computer science,*



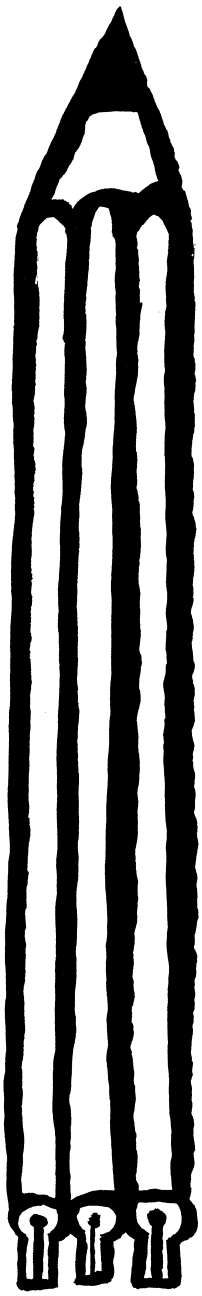
documented by a published paper or a series of published papers (https://eatcs.org/index.php/presburger). The recipients of the ***Presburger Award 2023*** *are jointly* ***Aaron Bernstein*** *and* ***Thatchaphol Saranurak****.*

- *The* ***Alonzo Church Award*** *for Outstanding Contributions to Logic and Computation is a major prize in cooperation of the EATCS with ACM Special Interest Group on Logic and Computation and the European Association for Computer Science Logic (https://eatcs.org/index.php/church-award). The recipients of the* ***2023 Alonzo Church Award*** *are jointly Lars Birkedal, Aleš Bizjak, Derek Dreyer, Jacques-Henri Jourdan, Ralf Jung, Robbert Krebbers, Filip Sieczkowski, Kasper Svendsen, David Swasey, and Aaron Turon, for the design and implementation of Iris, a higher-order concurrent separation logic framework.*

Moreover, during the conference, we will honor the recipients of the 2022 EATCS Distinguished Dissertation Award and the new group of EATCS Fellows.

The EATCS established the ***EATCS Distinguished Dissertation Award*** *to promote and recognize outstanding dissertations in the field of Theoretical Computer Science (https://eatcs.org/index.php/dissertation-award). The three recipients of the* ***2022 EATCS Distinguished Dissertation Award*** *for the PhD dissertation in the field of Theoretical Computer Science that have been successfully defended in 2022 are*

- *Kuikui Liu, University of Washington (advisor: Shayan Oveis Gharan),*



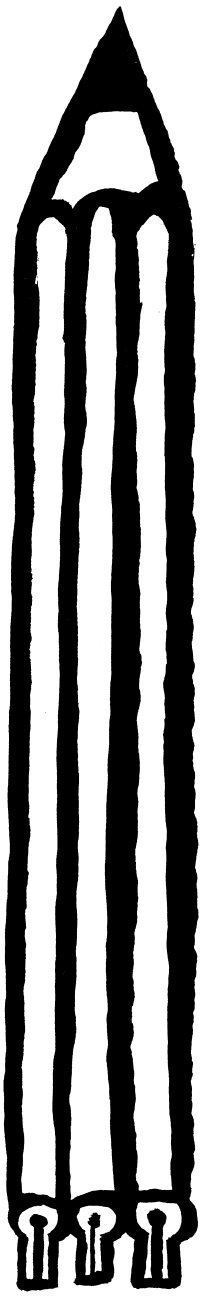
- *Alex Lombardi, MIT (advisor: Vinod Vaikuntanathan), and*
- *Lijie Chen, MIT (advisor: Ryan Williams).*

*The **EATCS Fellows** Program is established by the EATCS to recognize outstanding EATCS Members for their scientific achievements in the field of Theoretical Computer Science. The new group of **EATCS Fellows (class 2023)** consists of*

- *Michael A. Bender (Stony Brook University, USA),*
- *Leslie Ann Goldberg (University of Oxford, UK), and*
- *Claire Mathieu (CNRS, France).*

In addition to the standard conference program, ICALP 2023 will also have ten satellite workshops co-located with the main conference, taking place on Monday before the main event:

- *Combinatorial Reconfiguration*
- *Graph Width Parameters: from Structure to Algorithms (GWP 2023)*
- *Algorithmic Aspects of Temporal Graphs VI*
- *Adjoint Homomorphism Counting Workshop (ad hoc)*
- *Congestion Games*
- *Workshop On Reachability, Recurrences, and Loops '23 (WORReLL'23)*
- *Workshop on Recent Trends in Online Algorithms*



- *Quantum Computing with Qiskit, and why Classical Algorithms still matter!*
- *Algebraic Complexity Theory*
- *Computer Science for CONTINUOUS Data*

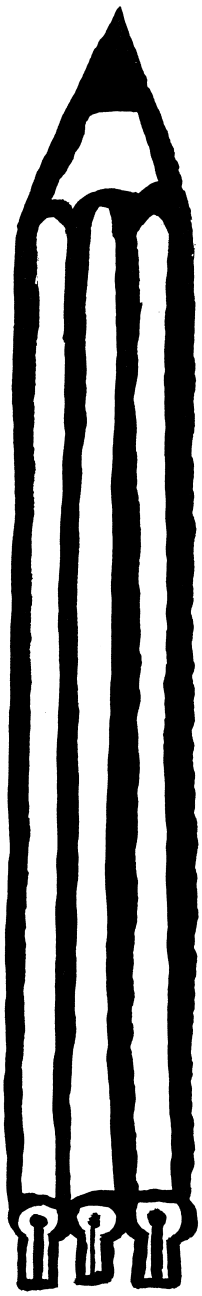
As usual, a more detailed report on the ICALP 2023 conference will be published in the October 2023 issue of the Bulletin.

In the recent months we have seen announcements of numerous further awards given to the members of theoretical computer science. While more details about many of these awards can be found on the pages of this Bulletin and elsewhere, let me list some highlights here.

*The **Gödel Prize** for outstanding papers theoretical computer science is sponsored jointly by the EATCS and the ACM SIGACT (https://eatcs.org/index.php/goedel-prize). The **2023 Gödel Prize**, to be presented at STOC 2023, has been awarded to the following two papers:*

- *Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Raj Tiwary, and Ronald de Wolf. "Exponential lower bounds for polytopes in combinatorial optimization," STOC 2012: 95-106, and Journal of the ACM, 62(2), 17:1-17:23 (2015);*
- *Thomas Rothvoss. "The matching polytope has exponential extension complexity," STOC 2014: 263-272, and Journal of the ACM, 64(6), 1-19 (2017).*

*The **Edsger W. Dijkstra Prize in Distributed Computing** is awarded for outstanding papers on the principles of distributed computing*



(https://eatcs.org/index.php/dijkstra-prize), and is sponsored jointly by the EATCS Symposium on Distributed Computing (DISC) and the ACM Symposium on Principles of Distributed Computing (PODC).
The ***2023 Dijkstra Prize*** *has been awarded for introducing Information-Theoretic Secure Multiparty Computations and showing how to achieve maximal resilience to malicious adversaries while providing unconditional security to the following papers:*

- *Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson, "Completeness theorem for non-cryptographic fault-tolerant distributed computation," STOC 1988: 1-10;*
- *David Chaum, Claude Crépeau, and Ivan Damgård, "Multiparty unconditionally secure protocols," STOC 1988, 11-19;*
- *Tal Rabin and Michael Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," STOC 1989, 73-85.*

EATCS also sponsors the ***Best ETAPS Paper Award 2023*** *for the best theory paper at ETAPS, which this year was awarded to the following paper:*

- *Pascal Baumann, Flavio D'Alessandro, Moses Ganardi, Oscar Ibarra, Ian McQuillan, Lia Schütze, and Georg Zetzsche, "Unboundedness problems for machines with reversal-bounded counters."*

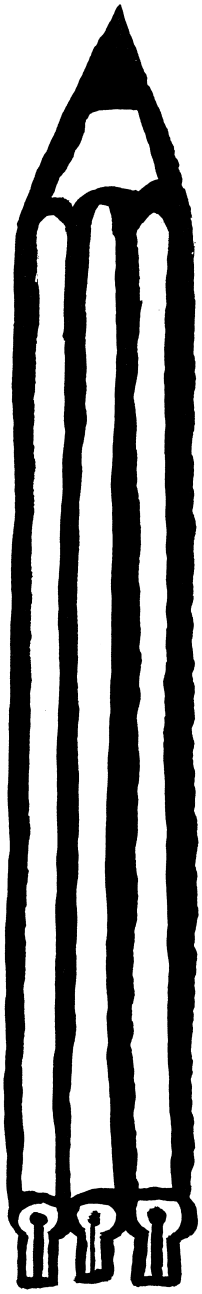
Congratulations to the award winners and new EATCS Fellows!

On behalf of the EATCS, I also heartily thank the members of the awards, dissertation and fellow committees for their work in the selection of this stellar set of award recipients and fellows. It will be a great honor to celebrate the work of these colleagues during ICALP 2023. (More details about the EATCS Award 2023, the EATCS Presburger Award 2023, 2023 Alonzo Church Award, the 2022 EATCS Distinguished Dissertation Award, the EATCS Fellows, and the Gödel Prize 2023 are presented on the later pages of this issue of the Bulletin.)

Also, please allow me to remind you about three other EATCS affiliated conferences that will be taking place later this year.

- ***MFCS 2023:*** *the 48th International Symposium on Mathematical Foundations of Computer Science, will be held in Bordeaux, France, August 28-September 1, 2023 (https://mfcs2023.labri.fr/).*
- ***ESA 2023:*** *the 31st Annual European Symposium on Algorithms, will be held in Amsterdam, The Netherlands, September 4-6, 2023 (https://algo-conference.org/2023/esa/).*
- ***DISC 2023:*** *the 36th International Symposium on Distributed Computing, will be held in L'Aquila, Italy, October 9-13, 2023 (http://www.disc-conference.org/wp/disc2023/).*

As usual, let me close this letter by reminding you that you are always most welcome to send me your comments, criticisms and suggestions for improving the impact of the EATCS on the Theoretical

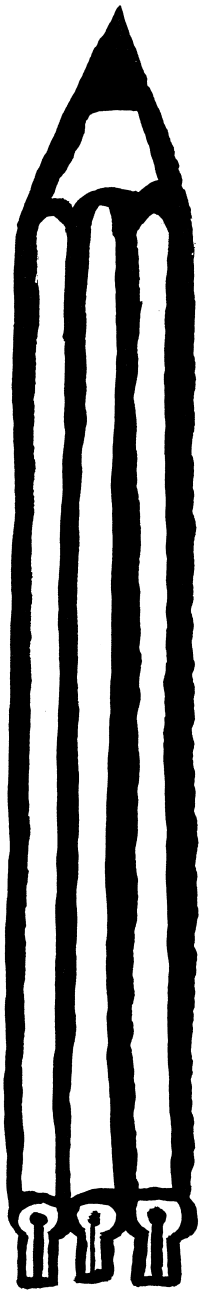


Computer Science community at president@eatcs.org. We will consider all your suggestions and criticisms carefully.

I look forward to seeing many of you at ICALP 2023 and to discussing ways of improving the impact of the EATCS within the theoretical computer science community at the general assembly.

Artur Czumaj
University of Warwick, UK
President of EATCS
president@eatcs.org

June 2023



Dear EATCS community,

the Summer issue of the EATCS Bulletin is out! And I thank all editors and contributors, and in particular Efi Chita from the EATCS Secretary Office, for all their help in making this a very interesting and special issue!

I invite you to read the interview with Shweta Agrawal, our "person behind the papers" of this issue, who shares her career experiences and advice, and also tells us about her move back from the USA to India, realizing her childhood dream. In the algorithms column, Stefan Walzer reviews and compiles interesting lessons from studying the power of choices in hashing-based data structures. In particular, he explores the space efficiency of data structures, if the additional power afforded by more than 2 choices is outweighed by the additional costs, and whether it may be beneficial to try less power. In the complexity column, Michal Koucky, raises the question whether the time has come to let the automata and grammars courses go and replace the content by other theoretical foundations of computer science; he also discusses hurdles which one might enounter when trying to modify the course.

In the logics column, Kenichi Morita surveys how computing is effectively performed in a reversible world: physical reversibility is one of the fundamental laws of nature, so can computing machines be realized utilizing a reversible law as well and how? In the distributed computing column, Michel Raynal revisits the concepts

of mutual exclusion and consensus and argues that consensus is to logical objects what mutual exclusion is to physical objects. In the formal languages theory column, Joel D. Day, Vijay Ganesh, and Florin Manea present an overview of recent results on how formal languages and their properties can be expressed via theories over strings. In the educations column, Valentina Dagiene reports on the Bebras challenge and how it inspires informatics education.

The Bulletin further includes a report of the scientific colloquium in honor of former EATCS President Burkhard Monien on the occasion of his 80th birthday, as well as conference reports and updates from EATCS Japan chapter.

Enjoy the new Bulletin!

Stefan Schmid, Berlin
June 2023

The EATCS Award 2023

Amos Fiat

The EATCS Award 2023 is awarded to

Amos Fiat

The 2023 EATCS Award is given Amos Fiat for his fundamental work within many areas of theoretical computer science and in particular for work in cryptography, on-line algorithms, and algorithmic game theory.

In cryptography, the Fiat-Shamir heuristic for protocol design has had enormous influence. It is basic, simple and elegant. In particular, it gives a way to transform any interactive identification scheme into a non-interactive signature scheme.

A world of massive communication with ever changing group membership calls for novel solutions. Fiat and Naor created the notion of broadcast encryption and proposed an implementation which comes with both theoretical guarantees and excellent practical performance.

Among other influential contributions in cryptography are the Feige–Fiat–Shamir identification scheme and his work with Chaum and Naor on electronic money.

In the area of on-line algorithm Fiat has many excellent contributions including foundational work on competitive paging algorithms and the k-server problem.

His most recent research in algorithmic game theory led to several fundamental results in algorithmic mechanic design, auctions and pricing, and the study of market equilibria.

The EATCS Award Committee 2023

- Johan Håstad (chair)
- Thomas Henzinger

- Valerie King

The award will be presented at ICALP 2023, in Paderborn.

The EATCS annually honors a respected scientist from our community with this prestigious EATCS Distinguished Achievements Award, to acknowledge extensive and widely recognized contributions to theoretical computer science over a life long scientific career (see http://eatcs.org/index.php/eatcs-award for more information, including the list of previous recipients)

The following is the list of the previous recipients of the EATCS Awards:

2022 Patrick Cousot
2021 Toniann (Toni) Pitassi
2020 Mihalis Yannakakis
2019 Thomas Henzinger
2018 Noam Nisan
2017 Éva Tardos
2016 Dexter Kozen
2015 Christos Papadimitriou
2014 Gordon Plotkin
2013 Martin Dyer
2012 Moshe Y. Vardi
2011 Boris (Boaz) Trakhtenbrot
2010 Kurt Mehlhorn
2009 Gérard Huet
2008 Leslie G. Valiant
2007 Dana S. Scott
2006 Mike Paterson
2005 Robin Milner
2004 Arto Salomaa
2003 Grzegorz Rozenberg
2002 Maurice Nivat
2001 Corrado Böhm
2000 Richard Karp

The Presburger Award 2023

Laudatio

The 2023 Presburger Committee has unanimously selected

Aaron Bernstein
and
Thatchaphol Saranurak

The 2023 Presburger Award Committee has chosen Aaron Bernstein and Thatchaphol Saranurak as joint recipients of the 2023 EATCS Presburger Award for Young Scientists. These two remarkable individuals have emerged as leaders in the field of fast graph algorithms, focusing their research on core problems such as shortest paths, connectivity, and matching. These problems have long been central to computer science, driving advancements in education, research, and real-world applications.

Aaron Bernstein has spearheaded a new wave of major breakthroughs in fast graph algorithms, with numerous achievements authored by him or inspired by his original ideas. Several techniques that by now are part of the toolbox for the design of graph algorithms are largely due to Aaron. These include edge-degree constrained subgraphs and hopsets for undirected graphs, and low diameter decompositions for directed graphs. These techniques, as used by Aaron and others, have made a profound impact on fundamental graph problems such as shortest paths and maximum matching, in multiple settings, including sequential, dynamic, distributed, online, parallel, sublinear, and streaming. A notable recent example of his groundbreaking work is a near linear time algorithm for the single source shortest path problem in directed graphs, allowing for edges of negative weights.

Thatchaphol Saranurak has made remarkable strides in the design of efficient graph algorithms, often solving long standing open problems. In his work, Thatchaphol developed techniques that are of general interest and of high impact.

A notable such technique is expander decomposition. Thatchapol introduced and applied it successfully in new settings such as dynamic graph algorithms and directed graphs. Moreover, Thatchaphol designed new efficient constructions of expander decompositions, in sequential settings as well as in distributed ones. Thatchaphol's many results span a wide range of graph problems, including nearly best possible algorithms for classical problems such as vertex connectivity and all pairs max flow, and groundbreaking work on dynamic algorithms for problems such as spanning forests and matching.

The Presburger Committee 2023

- Mikołaj Bojańczyk (chair)
- Uriel Feige
- Tal Malkin

EATCS Distinguished Dissertation Award for 2022

EATCS is proud to announce that, after examining the nominations received from our research community, the EATCS Distinguished Dissertation Award Committee 2022, consisting of Susanne Albers, Nikhil Bansal (chair), Elvira Mayordomo, Jaroslav Nešetřil, Damian Niwiński, Vladimiro Sassone, Alexandra Silva and David Woodruff, has selected the following three theses as recipients of the EATCS Distinguished Dissertation Award for 2022:

Kuikui Liu: "Spectral Independence: A New Tool to Analyze Markov Chains" (University of Washington; supervisor: Shayan Oveis Gharan).

Alex Lombardi: "Provable Instantiations of Correlation Intractability and the Fiat-Shamir Heuristic" (Electrical Engineering and Computer Science (EECS) at MIT; supervisor: Vinod Vaikuntanathan)

Lijie Chen: "Better Hardness via Algorithms, and New Forms of Hardness versus Randomness" (MIT Department of Electrical Engineering and Computer Science; supervisor: Ryan Williams)

The award certificate will be presented to in the award ceremony of ICALP 2023, to take place in Paderborn, Germany, in July 10-14, 2023.

The EATCS Distinguished Dissertation Award Committee 2022 consisted of

- Susanne Albers
- Nikhil Bansal (chair)
- Elvira Mayordomo
- Jaroslav Nesetril
- Damian Niwiński
- Vladimiro Sassone
- Alexandra Silva
- David Woodruff

The EATCS Distinguished Dissertation Award has been established to promote and recognize outstanding dissertations in the field of Theoretical Computer Science. Any PhD dissertation in the field of Theoretical Computer Science successfully defended in 2022 has been eligible. The dissertations were evaluated on the basis of originality and potential impact on their respective fields and on Theoretical Computer Science. Each of the selected dissertations will receive a prize of 1000 Euro. The award receiving dissertations will be published on the EATCS web site, where all the EATCS Distinguished Dissertations will be collected.

The list of the previous recipients of the EATCS Distinguished Dissertation Award is available at https://eatcs.org/index.php/dissertation-award.

EATCS-Fellows 2023

The EATCS has recognized three of its members for their outstanding contributions to theoretical computer science by naming them as recipients of an EATCS fellowship.

The EATCS Fellows for 2023 are:

Michael A. Bender, for fundamental contributions in bringing theoretical computer science techniques to practical problems and systems.

Leslie Ann Goldberg, for fundamental contributions to many areas of theoretical computer science, primarily focusing on randomized algorithms and their limitations .

Claire Mathieu, for fundamental contributions to solving theoretical and applied problems in approximation algorithms, online algorithms, and auction theory.

The aforementioned members of the EATCS were selected by the EATCS Fellow Selection Committee, after examining the nominations received from our research community.

The EATCS Fellow Selection Committee consisted of

- Christel Baier
- Mikołaj Bojanczyk
- Mariangiola Dezani
- Josep Diaz
- Giuseppe F. Italiano (chair)

The EATCS Fellows Program was established by the association in 2014 to recognize outstanding EATCS members for their scientific achievements in the

field of Theoretical Computer Science. The Fellow status is conferred by the EATCS Fellows-Selection Committee upon a person having a track record of intellectual and organizational leadership within the EATCS community. Fellows are expected to be "model citizens" of the TCS community, helping to develop the standing of TCS beyond the frontiers of the community.

The EATCS is very proud to have the above-mentioned members of the association among its fellows.

The list of EATCS Fellows is available at http://www.eatcs.org/index.php/eatcs-fellows.

2023 Alonzo Church Award for Outstanding Contributions to Logic and Computation

The awardee papers are:

- Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, Derek Dreyer: "Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning". POPL 2015.
- Ralf Jung, Robbert Krebbers, Lars Birkedal, Derek Dreyer: "Higher-order ghost state". ICFP 2016.
- Robbert Krebbers, Ralf Jung, Aleš Bizjak, Jacques-Henri Jourdan, Derek Dreyer, Lars Birkedal: "The Essence of Higher-Order Concurrent Separation Logic". ESOP 2017.
- Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, Derek Dreyer: "Iris from the ground up: A modular foundation for higher-order concurrent separation logic". J. Funct. Program. 28 (2018).

for the design and implementation of Iris, a higher-order concurrent separation logic framework.

The award was established in 2015 by SIGLOG, EATCS, EACSL and the Kurt Gödel society.

The 2023 Alonzo Church Award Committee:

- Thomas Colcombet,
- Mariangiola Dezani,
- Marcelo Fiore,
- Radha Jagadeesan and
- Igor Walukiewicz.

The list of the previous recipients of the Alonzo Church Award for Outstanding Contributions to Logic and Computation is available at `https://www.eatcs.org/index.php/church-award`.

2023 GÖDEL PRIZE

The 2023 Gödel Prize is awarded to the following papers

- Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Raj Tiwary and Ronald de Wolf: Exponential Lower Bounds for Polytopes in Combinatorial Optimization. STOC 2012: 95-106. J. ACM, 62(2), 17:1-17:23 (2015)
- Thomas Rothvoss: The matching polytope has exponential extension complexity. STOC 2014: 263-272. J. ACM, 64(6),1-19 (2017)

Linear Programming and polyhedral methods form the backbone of combinatorial optimization. Associating a polytope to a discrete optimization problem, and characterizing its structure, has long furnished important insights in combinatorics and algorithm design.

A basic question is whether the polytopes for classical problems such as traveling salesman and matching admit a small description. Resolving a long-standing question, Fiorini, Massar, Pokutta, Tiwary and de Wolf made ingenious use of techniques from communication complexity (following a framework pioneered by Yannakakis) to show that any extended formulation for the TSP polytope has exponential size.

Building on these techniques, Rothvoss showed that even for the perfect matching problem, which is polynomially time solvable, any extended formulation of its polytope must have exponential size. This shows that Edmonds's characterization of the matching polytope from the 1960s is essentially optimal.

Award Committee:

- Nikhil Bansal (University of Michigan)
- Irit Dinur (Weizmann Institute)
- Anca Muscholl (University of Bordeaux)
- Tim Roughgarden (Columbia University)
- Ronitt Rubinfeld, Chair (Massachusetts Institute of Technology)
- Luca Trevisan (Bocconi University)

The list of the previous recipients of the Gödel Prize is available at `https://eatcs.org/index.php/goedel-prize`.

IPEC Nerode Prize 2023

The EATCS-IPEC Nerode Award Committee consisting of Fedor V. Fomin (chair), Thore Husfeldt, and Sang-il Oum, has selected the following paper as the recipient of the EATCS-IPEC Nerode Prize 2023:

"Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time" by Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. ACM Trans. Algorithms 18(2): 17:1-17:31 (2022). Conference version: FOCS 2011. *Appraisal*

The paper solved a major open problem in parameterized complexity, with numerous consequences and bounds improved. It deals with so-called graph connectivity problems, where the goal is to find in a given graph a structure that, in addition to other constraints, is connected. This is a broad class of fundamental problems, including Steiner Tree, Hamiltonian Cycle, Connected Vertex Cover, and Feedback Vertex Set. The paper focuses on these problems parameterized by the width of the given tree decomposition of the input graph.

The paper introduces a new generic technique called cut-and-count. The technique brings drastic improvements in the running times of many state-of-the-art dynamic programming algorithms on graphs of bounded treewidth. By now, cut-and-count has become one of the classic techniques in parameterized complexity. The main insight of the Nerode Award paper–dynamic programming does not need to track all the partial solutions—paved the road to many other important results in graph algorithms. For this reason, the Nerode Prize 2023 Committee unanimously decided that this breakthrough paper by Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, and Wojtaszczyk deserves to win the Nerode prize.

Institutional Sponsors

CTI, Computer Technology Institute & Press "Diophantus"
Patras, Greece

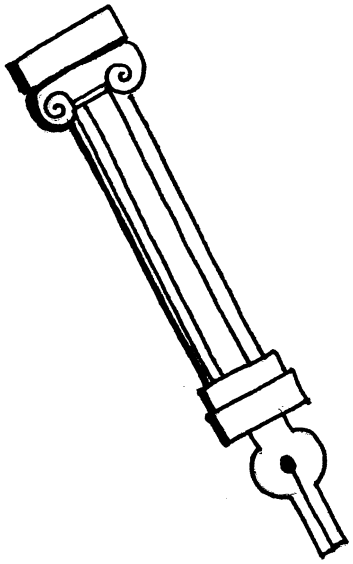
CWI, Centum Wiskunde & Informatica
Amsterdam, The Netherlands

MADALGO, Center for Massive Data Algorithmics
Aarhus, Denmark

Microsoft Research Cambridge
Cambridge, United Kingdom

Springer-Verlag
Heidelberg, Germany

EATCS
Columns



The Interview Column

BY

Chen Avin and Stefan Schmid

Ben Gurion University, Israel and TU Berlin, Germany
{chenavin,schmiste}@gmail.com

KNOW THE PERSON BEHIND THE PAPERS

Today: Shweta Agrawal

Bio: *Shweta Agrawal is an associate professor at the Computer Science and Engineering department, at the Indian Institute of Technology, Madras. She earned her PhD at the University of Texas at Austin, and did her postdoctoral work at the University of California, Los Angeles. Her area of research is cryptography and broadly theoretical CS, with a focus on post quantum cryptography. She has received several awards and honours such as the national Swarnajayanti award, the ACM India award for Outstanding Contributions to Computing by a Woman, a best paper award at Eurocrypt, best reviewer awards for Asiacrypt and CCS, invited speaker and program co-chair at the flagship conference Asiacrypt.*

EATCS: We ask all interviewees to share a photo with us. Can you please tell us a little bit more about the photo you shared?

SA: This picture is with my pet Tara, during one of our morning walks. This is a daily event so the picture is not special in the sense of being rare. But it shows how ordinary can be so fun!

EATCS: Can you please tell us something about you that probably most of the readers of your papers don't know?

SA: I love all kinds of art – music, painting, poetry, literature, sculpture, ceramics and anything else. I pursued painting (oil on canvas) quite seriously for several years and had the beginnings of a career there (via some initial exhibitions of my work) before I ended up dedicating most of my time to academia. I think I love math because I see it also as a kind of art. I see beautiful connections between cryptography and abstract expressionism – both are playing with the boundaries between structure and randomness, form and formlessness! I also love to hike, and find myself in the Himalayas every summer.

EATCS: Is there a paper which influenced you particularly, and which you recommend other community members to read?

SA: When I was in graduate school, the first paper on fully homomorphic encryption by Craig Gentry [2] came out. This paper had a very significant impact on me.

It was such a beautiful, simple to state problem, and the solution was so creative. A series of subsequent works (also very cool) improved this significantly, but the first paper was always special to me. It was so surprising – it broadened for me

the landscape of what is possible. I have always loved the apparent paradoxes in cryptography and this paper was a wonderful example.

EATCS: Is there a paper of your own you like to recommend the readers to study? What is the story behind this paper?

SA: I think the paper on obfuscation from Eurocrypt 2019 [1] is my paper that I like the most. I loved the problem – the balancing between algebraic structure and computational hardness was so delicate and beautiful! In this paper, I made new conjectures about hard lattice problems and this was a very mixed experience. On one hand, I love exploring such questions, on the other, it also gave me many anxious days and sleepless nights. I sat on this paper for a whole year before finally posting it online – I was so nervous about the conjectures. I used to joke that I love it on odd dates and hate it on even. Finally, a casual conversation with a colleague (Daniele Micciancio) is what helped me make it public – he remarked something to the effect that if my conjectures survive (algorithmic attacks), it's great, if they are broken, then I'll still be in good company. He was referring to the fact that several conjectures made by eminent researchers in the space of obfuscation had been broken. I decided it was worth putting out there for people to look at.

EATCS: When (or where) is your most productive working time (or place)?

SA: Early hours of the morning are the best for me. Place does not matter too much – after Covid, I've become relatively robust to this aspect and can work from anywhere. That said, quiet walks in green places where I am not actively thinking about any problem, but just letting "the cooking pot" simmer in my head, are the most productive. For me, most good ideas come when the mind is relaxed, typically when not working actively. I am fortunate to live in a very beautiful area with gorgeous, seemingly timeless banyan trees all around. While summer is very hot where I live, there is also a wonderful, stark beauty to it. The bright yellow of the sun filtering through the thick green foliage of the banyans is magical. I suspect that every good idea that I have ever had was somehow born here.

EATCS: What do you do when you get stuck with a research problem? How do you deal with failures?

SA: I try to have clarity on why the techniques I am trying are not sufficient for the problem I am working on. By identifying a fundamental roadblock which is not allowing me to proceed, I feel that I have understood the root cause of the issue and this helps me to get closure. At a philosophical level, I try to not be too obsessed with the outcome of the effort – at a deeper level, one is in this field for the beauty of it, and grasping too much at particular desired results ruins that. So stepping back and focusing energy on something else (inside or outside science)

helps. Having dealt with a large number of failures over the years also helps – one is able to internalize that life really does go on and there are many other questions to study.

EATCS: Is there a nice anecdote from your career you would like to share with our readers?

SA: Perhaps the most significant thing I can share is from the time when I started the process of moving back to India after my postdoc, which (together with my PhD) was in the US. I had always wanted to contribute to science in India – my growing up years in India had been in an environment of enormous struggle and strife, set in a canvas of equally enormous kindness and generosity. In this environment, I had somehow managed to get the best of opportunities that any girl could have, anywhere in the world. It was my deepest desire growing up, to somehow give back to where I came from, in whatever small way I could.

While I had always been certain I wanted to do this, I had a bad case of cold feet as the time started approaching. During my PhD and postdoc, I had worked in groups that were among the best in the world, and had always been surrounded by super smart, highly trained and extremely motivated people. The momentum of the group had always been a big factor in my own progress, and I was very nervous about whether I could do any meaningful research without such a support structure – back then there was almost no one working in my area in India. To add to it, people that I held in very high regard made comments about how this move was essentially professional suicide, and this shook my already shaky confidence even further.

Yet, I made the move and things worked out. Being without the support of the group structure forced me to become more independent and get clarity on my own research agenda. I believe my work has become more authentically my own and hence deeper, and it has been very satisfying to be able to live my childhood dream. Looking back, I'm glad I did not give in to the worries and fears of that time!

EATCS: Do you have any advice for young researchers? In what should they invest time, what should they avoid?

SA: I think the main "advice" I have (this makes me feel old!) is to be unapologetic about being yourself and to work on questions that you love. Invest time in taking courses, learning new things, attending talks and asking lots of questions. When trekking up a mountain, one enjoys the beautiful views along the way. Research is the same – the tedium of the climbing should not get in the way of the enjoyment. If one dedicates one's effort to something unshakeable, like beauty or service or anything broader than oneself, then it is easier to keep going when frustrations and failures come (as they inevitably will).

One should avoid comparing oneself with others, or falling into negative self talk and such other habit patterns. We feel best and do best by being ourselves, not anyone else – no single colour is the most important in a painting. I remember thinking for one painting (by Cezanne, I believe), how a single streak of a particular green (viridian), which was not present anywhere else in that painting, was foundational to its beauty. One must ruthlessly negate the desire to be like anyone else.

EATCS: What are the most important features you look for when searching for graduate students?

SA: I try to separate ability and "spark" from training. Often students come in who haven't had the opportunity to get very rigorous training yet – this is something that can be fixed, given time. What is more innate is their ability, enthusiasm, motivation and earnestness. I've been very lucky in the students I have had so far. I also think that as an advisor, one can play a big role in shaping innate qualities and helping to develop weaker aspects. So I try to keep a very open mind.

EATCS: Do you see a main challenge or opportunity for theoretical computer scientists for the near future?

SA: Science is constantly evolving and each era brings wonderful new questions. Being a cryptographer, one area I am very excited about is quantum algorithms and their effect on cryptography. We understand so little about computational hardness in cryptography even in the classical regime and asking questions about hardness in the realm of quantum is really intriguing. I am excited to see what quantum algorithms can do to solve problems that we consider difficult classically.

EATCS: Can you recommend some source of information that you enjoy (e.g., a specific blog, podcast, youtube channel, book, show, ...)?

SA: I enjoy reading, particularly Indian philosophy, history and popular science. A favourite in the popular science category is "The fabric of the cosmos" by Brian Greene. I love poetry deeply – a favourite here is Seamus Heaney's "Clearances". Another book I recently enjoyed a lot was "Consolations" by David Whyte. A great source of inspiration for me is painting – I am blown away by the works of Jackson Pollock, M.F Hussain, Hans Hoffman and Picasso. The pinnacle of brilliance and beauty for me are the verses of some of the old Indian philosophy texts like "Katha Upanishad" and "Yoga Vashishta".

Please complete the following sentences?

- *Being a researcher* is a great job – so much freedom and space!
- *My first research discovery* gave me the confidence that I could do this.
- Having good intentions *is key to being a happy academic.*
- *Theoretical computer science in 100 years from now* will be just as rich and beautiful!

References

[1] S. Agrawal. Indistinguishability obfuscation without multilinear maps: New techniques for bootstrapping and instantiation. In *Eurocrypt*, 2019.

[2] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

The Logic in Computer Science Column

BY

Yuri Gurevich

Computer Science & Engineering
University of Michigan, Ann Arbor, Michigan, USA
gurevich@umich.edu

BEATCS no 140

Making Reversible Computing Machines in a Reversible Cellular Space

Kenichi Morita
Hiroshima University, Higashi-Hiroshima 739-8527, Japan
Currently Professor Emeritus of Hiroshima University
km@hiroshima-u.ac.jp

Abstract

Reversible computing is a study that investigates the problem of how computing is effectively performed in a reversible world. Since physical reversibility is one of the fundamental microscopic laws of nature, it is important to clarify how computing machines are realized utilizing a reversible law directly. In this survey/tutorial paper, we investigate this problem using a reversible cellular automaton as a reversible environment, and search for a new way of constructing reversible Turing machines (RTMs), a model of a reversible computer, in it. That is to find a good pathway from a reversible microscopic law to reversible computers. When doing so, it is convenient to assume several conceptual levels on the pathway, by which the problem is decomposed into subproblems. In the middle level on the pathway we use a reversible logic element with 1-bit memory (RLEM), rather than a reversible logic gate, as a logical primitive. By these methods, we see that RTMs can be implemented systematically even in a space that obeys a very simple reversible microscopic law.

1 Introduction

A reversible computing machine is a system having a "backward deterministic" property. That is to say, every computational state of the machine has at most one predecessor state. Though its definition is thus simple, it has a close relation to the physical reversibility, one of the fundamental microscopic laws in physics [1, 8]. Therefore, it is important to know how reversible machines are realized by utilizing reversible microscopic phenomena.

So far many kinds of reversible computing models have been proposed and studied. One method of showing that a reversible computing model has a sufficient computing power is to simulate an irreversible version of the model by a

reversible one, i.e., "reversifying" [4] the irreversible system. By this method, computational/logical universality of various reversible models have been shown. Reversifying techniques have been applied, for example, to Turing machines [1], logic elements and circuits [3, 26], two-counter machines [12], two-way finite automata [7], two-way multihead finite automata [14], cellular automata [25], and so on. Once universality of a reversible computing model is established by this method, universality of another reversible model can be shown by simulating the former by the latter. For example, it has been shown that a universal reversible logic gate and its circuits are simulated by a simple reversible 2D cellular automaton [11], and that a reversible Turing machine can be simulated by a reversible 1D cellular automaton [19]. In this way, it turned out that, in many computing models, computing powers do not decrease even if the reversibility constraint is added.

Besides the study of individual reversible computing machines, it is also important to investigate how these machines can be efficiently realized in a space that obeys a simple reversible law. In other words, it is to investigate the problems of how simple reversible primitive operations can be that support universal computation, and how reversible macroscopic systems can be realized from a reversible microscopic law. When we try to implement reversible machines in such a simple environment, it is convenient to consider several implementation levels ranging from a microscopic level to a macroscopic one as shown in Fig. 1. By this, the problem is decomposed into several simpler subproblems. In the bottom level, i.e., Level 1, there is a simple microscopic reversible law of evolution, which corresponds to a microscopic physical law. In Level 2, various phenomena that emerge from the reversible microscopic law can be observed. In Level 3, we implement suitable reversible logic elements using the observed phenomena. In Level 4, combining the reversible logic elements, functional modules for reversible computers are composed. In the top level, i.e., Level 5, a reversible computing machine is systematically constructed by assembling the reversible functional modules.

Whether we can successfully find a pathway from a reversible microscopic law to reversible computers firstly depends on the choice of the reversible microscopic law in Level 1. In this paper, we use a reversible cellular automaton for it as a thought experiment. It is a reversible elementary square partitioned cellular automaton (ESPCA) with a hexadecimal identification number "01caef"' [17], which is denoted by P_0 for short in this paper. It is described by only six local transition rules, and thus very simple. Though the reversible cellular automaton used here is an artificial model, and its physical realizability in the nano-scale level is not known at present, they will give new vistas in reversible computing. In particular, we shall see that even from very simple local rules, useful phenomena that can be used for composing reversible machines are found in Level 2.

It also depends on the choice of reversible logic elements in Level 3. Here, we

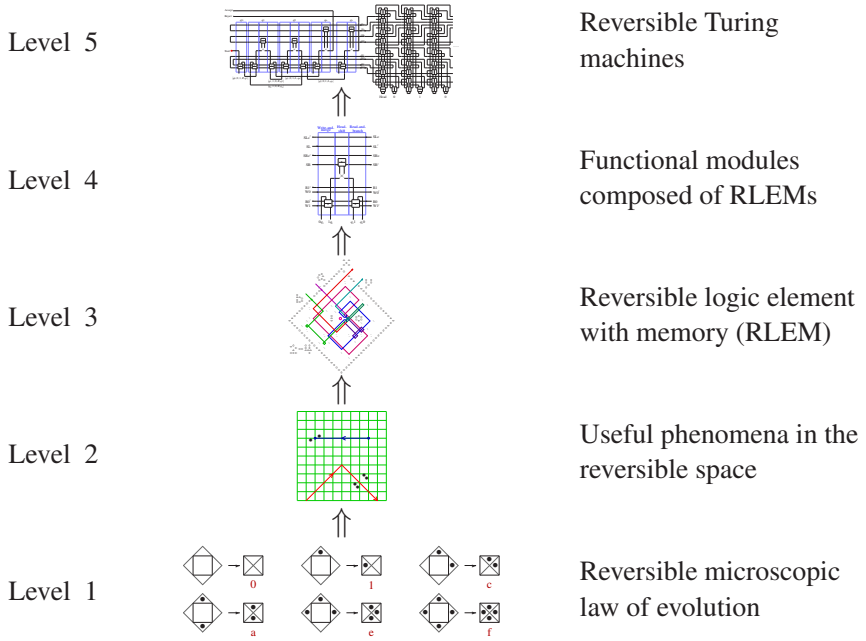


Figure 1: A pathway from a reversible microscopic law to reversible computers

use a reversible logic element with 1-bit memory (RLEM) rather than a reversible logic gate. We shall see that RLEMs can be implemented using a small number of phenomena found in Level 2. In addition, construction of reversible computing machines in the upper levels is greatly simplified.

In Levels 4 and 5, a reversible Turing machine (RTM), an abstract model of a reversible computer, is constructed. To do so, in Level 4, functional modules are composed out of RLEMs. Then, in Level 5, RTMs are systematically constructed by assembling the modules.

The contents of the following sections are as follows. In Sect. 2 reversible Turing machines are defined, and computational universality results of their restricted subclasses are surveyed. In Sect. 3 reversible logic elements with memory (RLEM) are given, and a construction method of RTMs using a particular RLEM called a rotary element (RE) is explained. In Sect. 4 a very simple 2D cellular automaton called an elementary square partitioned cellular automaton (ESPCA) is introduced, and useful phenomena in the particular reversible ESPCA P_0 are explored. Using them, an RE is implemented in P_0, and then RTMs are realized as configurations of P_0. Sect. 5 gives concluding remarks.

2 Reversible Turing Machine (RTM)

A reversible Turing machine (RTM) is a standard model in the theory of reversible computing. Lecerf [9] first investigated RTMs, and showed unsolvability of the halting problems and some related problems. Bennett [1] studied them from the viewpoint of thermodynamics of computing, and showed that any irreversible TM can be converted into an equivalent RTM.

2.1 Definitions and examples

A one-tape Turing machine (TM) consists of a finite control, a read-write head, and a two-way infinite tape divided into squares in which symbols are written. Formal definition of a TM is as follows.

Definition 2.1. A *one-tape Turing machine (TM)* is defined by

$$T = (Q, S, q_0, F, s_0, \delta),$$

where Q is a non-empty finite set of states, S is a non-empty finite set of tape symbols, q_0 is an *initial state* ($q_0 \in Q$), F is a set of *final states* ($F \subseteq Q$), and s_0 is a special *blank symbol* ($s_0 \in S$). Here, δ is a move relation, which is a subset of $(Q \times S \times S \times \{L, N, R\} \times Q)$. The symbols "$L$", "$N$", and "$R$" are *shift directions* of the head, which stand for "left-shift", "no-shift", and "right-shift", respectively. Each element of δ is a *quintuple* of the form $[p, s, s', d, q]$, which is called a *rule* of T. It means if T reads the symbol s in the state p, then write s', shift the head to the direction d, and go to the state q. We assume each state $q_f \in F$ is a *halting state*, i.e., there is no quintuple of the form $[q_f, s, s', d, q]$ in δ. In this paper, we assume T is deterministic. Hence, for any pair of distinct quintuples $[p_1, s_1, t_1, d_1, q_1]$ and $[p_2, s_2, t_2, d_2, q_2]$ in δ, the relation $(p_1 = p_2) \Rightarrow (s_1 \neq s_2)$ holds.

Reversibility of a TM is defined as below.

Definition 2.2. Let $T = (Q, S, q_0, F, s_0, \delta)$ be a TM. We call T a *reversible TM* (RTM), if the following holds for any pair of distinct quintuples $[p_1, s_1, t_1, d_1, q_1]$ and $[p_2, s_2, t_2, d_2, q_2]$ in δ.

$$(q_1 = q_2) \Rightarrow (d_1 = d_2 \wedge t_1 \neq t_2)$$

It means that for any pair of distinct rules, if the next states are the same, then the shift directions are the same, and the written symbols are different. The above is called the *reversibility condition* for TMs.

Note that, in [1], RTMs are defined in a quadruple form, where read-write rules and head-shift rules are separated. This formulation is useful when composing an "inverse" RTM that undoes the computation performed by a given RTM. However, here, we employ the quintuple formulation, since the number of rules for defining an RTM in the quintuple form is about a half of that for defining an RTM in the quadruple form. See Sect. 5.1.3 of [15] for a conversion method between these two forms.

An instantaneous description (ID) of a TM is an expression to describe its computational configuration.

Definition 2.3. Let $T = (Q, S, q_0, F, s_0, \delta)$ be a one-tape TM. We assume $Q \cap S = \emptyset$. An *instantaneous description* (ID) of T is a string of the form $\alpha q \beta$ where $q \in Q$ and $\alpha, \beta \in S^*$. Let λ denote the empty string. The ID $\alpha q \beta$ describes the *computational configuration* of T such that the content of the tape is $\alpha\beta$ (the remaining part of the tape contains only blank symbols), and T is reading the leftmost symbol of β (if $\beta \neq \lambda$) or s_0 (if $\beta = \lambda$) in the state q. An ID $\alpha q \beta$ is called a *standard form ID* if $\alpha \in (S - \{s_0\})S^* \cup \{\lambda\}$, and $\beta \in S^*(S - \{s_0\}) \cup \{\lambda\}$. Namely, a standard form ID is obtained from a general ID by removing superfluous blank symbols from the left and the right ends. An ID $\alpha q_0 \beta$ is called an *initial ID*. An ID $\alpha q \beta$ is called a *final ID* if $q \in F$.

The transition relation among standard form IDs of T is denoted by $\vdash_T$. Let $\alpha q \beta$ and $\alpha' q' \beta'$ be two standard form IDs. If $\alpha' q' \beta'$ is obtained from $\alpha q \beta$ by applying a rule in δ of T, then we write $\alpha q \beta \vdash_T \alpha' q' \beta'$, and say that T goes to the computational configuration $\alpha' q' \beta'$ from $\alpha q \beta$ in one step. For example, if $[q, s, s', R, q'] \in \delta$, $\alpha \in (S - \{s_0\})S^*$, and $\beta \in S^*(S - \{s_0\})$, then $\alpha q s \beta \vdash_T \alpha s' q' \beta$. Though the relation $\vdash_T$ is conceptually straightforward one, its formal definition is slightly complex, since only standard form IDs are considered, and thus we have to deal with many cases. Hence, its definition is omitted here (see Sect. 5.1.1.3 of [15] for its precise definition).

The reflexive and transitive closure of $\vdash_T$ is denoted by $\vdash_T^*$. The transitive closure is denoted by $\vdash_T^+$. The relation of n-step transition is denoted by $\vdash_T^n$. Let γ be a standard form ID of T. We say γ is a *halting ID*, if there is no ID γ' such that $\gamma \vdash_T \gamma'$. Let $\alpha_i, \beta_i \in S^*$, and $p_i \in Q$ $(n \in \mathbb{N}, i = 0, 1, \ldots, n)$. We say $\alpha_0 p_0 \beta_0 \vdash_T \alpha_1 p_1 \beta_1 \vdash_T \cdots \vdash_T \alpha_n p_n \beta_n$ (or $\alpha_0 p_0 \beta_0 \vdash_T^* \alpha_n p_n \beta_n$) is a *complete computing process* of T starting from $\alpha_0 p_0 \beta_0$, if $\alpha_0 p_0 \beta_0$ is an initial ID (i.e., $p_0 = q_0$), and $\alpha_n p_n \beta_n$ is a halting ID.

We give two examples of RTMs. In the following sections, they are constructed using reversible logic element with memory (RLEM), and then implemented in a simple reversible cellular automaton.

Example 2.1. An RTM T_{parity} defined below is a very simple example.

$$T_{\text{parity}} = (Q_{\text{parity}}, \{0,1\}, q_0, \{q_{\text{a}}\}, 0, \delta_{\text{parity}})$$

Here, $Q_{\text{parity}} = \{q_0, q_1, q_2, q_{\text{a}}, q_{\text{r}}\}$, and δ_{parity} are given below.

$$\begin{aligned}\delta_{\text{parity}} = \{\ & [q_0,0,1,R,q_1],\ [q_1,0,1,L,q_{\text{a}}],\ [q_1,1,0,R,q_2],\\ & [q_2,0,1,L,q_{\text{r}}],\ [q_2,1,0,R,q_1]\ \}\end{aligned}$$

It is easy to see that T_{parity} is reversible. Consider the pair of rules $[q_0,0,1,R,q_1]$ and $[q_2,1,0,R,q_1]$. The next states in these rules are the same (i.e., q_1). We can see the shift directions in them are the same (i.e., R), and the written symbols are different (i.e., 1 and 0). Thus the pair satisfies the reversibility condition in Definition 2.2. No other pair of distinct rules have the same next state. Therefore T_{parity} is reversible. Complete computing processes starting from the IDs q_0011 and q_00111 are as follows.

$$\begin{aligned}&q_0011 \ \vdash_{T_{\text{parity}}} 1q_111 \ \vdash_{T_{\text{parity}}} 10q_21 \ \vdash_{T_{\text{parity}}} 100q_1 \ \vdash_{T_{\text{parity}}} 10q_{\text{a}}01\\ &q_00111 \vdash_{T_{\text{parity}}} 1q_1111 \vdash_{T_{\text{parity}}} 10q_211 \vdash_{T_{\text{parity}}} 100q_11 \vdash_{T_{\text{parity}}} 1000q_2 \vdash_{T_{\text{parity}}} 100q_{\text{r}}01\end{aligned}$$

For a given string 01^n, the RTM T_{parity} tests whether n is even or not. If it is even, T_{parity} halts in the final (accepting) state q_{a}. Otherwise it halts in q_{r}. All the read symbols are complemented.

Example 2.2. An RTM T_{power} is defined by

$$T_{\text{power}} = (Q_{\text{power}}, \{0,1\}, q_0, \{q_{\text{a}}\}, 0, \delta_{\text{power}}).$$

Here, $Q_{\text{power}} = \{q_0, q_1, \ldots, q_7, q_{\text{a}}, q_{\text{r}}\}$, and δ_{power} are given below.

$$\begin{aligned}\delta_{\text{power}} = \{\ & [q_0,0,0,R,q_1],\ [q_1,0,0,R,q_2],\ [q_2,0,0,L,q_6],\ [q_2,1,0,R,q_3],\\ & [q_3,0,1,L,q_4],\ [q_3,1,1,R,q_3],\ [q_4,0,0,L,q_7],\ [q_4,1,0,L,q_5],\\ & [q_5,0,1,R,q_2],\ [q_5,1,1,L,q_5],\ [q_6,0,0,L,q_{\text{r}}],\ [q_6,1,1,R,q_1],\\ & [q_7,0,0,L,q_{\text{a}}],\ [q_7,1,1,L,q_{\text{r}}]\ \}\end{aligned}$$

It is again easy to see that T_{power} satisfies the reversibility condition. Complete computing processes starting from $q_0001111$ and $q_000111111$ are as follows.

$$\begin{aligned}&q_0001111 \ \vdash^{31}_{T_{\text{power}}} 110\, q_{\text{a}}1001\\ &q_000111111 \vdash^{43}_{T_{\text{power}}} 111\, q_{\text{r}}\, 01011\end{aligned}$$

For a given string 001^n, the RTM T_{power} tests whether n is a power of 2. If it is so, T_{power} halts in the final state q_{a}. Otherwise it halts in q_{r}. It uses a straightforward algorithm that repeatedly divides the unary number n by 2, and checks the remainder at each division. But, note that, T_{power} is carefully designed so that it satisfies the reversibility condition.

2.2 Computational universality of RTMs

Bennett [1] showed that any one-tape irreversible TM can be converted into an equivalent three-tape RTM. Hence, the class of three-tape RTMs is computationally universal.

Theorem 2.1. *For any (irreversible) one-tape TM, we can construct a reversible three-tape RTM that simulates the former and leaves no garbage information on its tape.*

Assume some class of RTMs is known to be computationally universal. If any RTM in this class is simulated by an RTM in another class of RTMs, then the latter class of RTMs is also computationally universal. In this way, computational universality of various subclasses of RTMs can be shown. In particular, it is possible to show the following.

(1) For any RTM with k two-way infinite tapes, we can construct an RTM with k one-way infinite (i.e., rightward infinite) tapes that simulates the former ($k = 1, 2, \ldots$).

(2) For any RTM with k rightward infinite tapes, we can construct an RTM with only one rightward infinite tape that simulates the former ($k = 2, 3, \ldots$).

(3) For any k-symbol RTM with one rightward infinite tape, we can construct a two-symbol RTM with one rightward infinite tape that simulates the former ($k = 3, 4, \ldots$).

In the case of irreversible TMs, it is relatively easy to show the results corresponding to the above (see, e.g., [6] for (1), [5] for (2), and [24] for (3)). However, in the case of RTMs, the simulating TMs should be carefully constructed so that they satisfy the reversibility condition. In addition, the notion of simulation should also be defined properly. These details are found in Sect. 5.3 of [15].

By above, we obtain the following.

Theorem 2.2. *The class of two-symbol RTMs with a rightward infinite tape is computationally universal.*

In the following sections, only two-symbol RTMs with a rightward infinite tape are constructed by reversible logic elements, and then implemented in a simple reversible cellular automaton.

It has been shown that a one-tape many-state RTM can be simulated by a one-tape three-state RTM having many symbols (see Sect. 5.3.5 of [15]). Therefore we have the following.

Theorem 2.3. *The class of three-state RTMs with a rightward infinite tape is computationally universal.*

In the case of irreversible TMs, it is known that a many-state TM can be simulated by a two-state TM [24]. Hence the class of two-state TMs is universal. However, it is unknown whether the class of two-state RTMs is universal.

A *universal Turing machine* (UTM) is one that can simulate any TM. Let UTM(m,n) denote an m-state n-symbol UTM. It is known that various kinds of UTMs with very small m and n exist. For example, Rogozhin [23] gave UTM(4,6), and Neary and Woods [22] gave UTM(6,4), which simulate 2-tag systems and bi-tag systems, respectively. These UTMs have the smallest value of $m \times n$ among the ones so far found.

It is, of course, possible to have a *universal reversible Turing machine* (URTM) by reversifying a UTM using the method of Bennett (Theorem 2.1) and then converting it into a one-tape RTM. However, if we do so, m and n become very large. In the case of URTM(m,n) with m states and n symbols, a method of simulating cyclic tag systems [2] was used to have ones with small m and n (Sect. 7.3 of [15]). Among them, URTM(10,8) has the smallest value of $m \times n$.

3 Reversible Logic Element with Memory (RLEM)

A *reversible logic element with memory* (RLEM) [13] is a kind of a reversible finite automaton having output symbols as well as input symbols, which is also called a reversible sequential machine of Mealy type. In the following, we use RLEMs rather than reversible logic gates for composing RTMs.

Definition 3.1. A *sequential machine* (SM) M is defined by $M = (Q, \Sigma, \Gamma, \delta)$, where Q is a finite set of states, Σ and Γ are finite sets of input and output symbols, and $\delta : Q \times \Sigma \to Q \times \Gamma$ is a move function (see Fig. 2 **(a)**). If δ is injective, it is called a *reversible sequential machine* (RSM).

To use an SM as a logic element, we interpret it as the one with decoded input/output ports (Fig. 2 **(b)**), i.e., for each input symbol, there is a unique input port to which a signal (or a particle) is given. It is also the case for the output symbols. Therefore, signals should not be given to two or more input ports at the same time.

An RLEM is an RSM that satisfies $|\Sigma| = |\Gamma|$. When connecting many RLEMs to form an RLEM-circuit, each output port of an RLEM can be connected to at most one input port of another (or may be the same) RLEM. Furthermore, two or more output ports should not be connected to one input port. Therefore, neither branching (i.e., fan-out of an output) nor merging of signal lines is permitted. See Sect. 3.5.1 of [15] for the precise definition of an RLEM-circuit.

Among RLEMs, two-state RLEMs are particularly important, since they are simple yet powerful (see Sect. 3.4). In the following, we use a specific RLEM, a rotary element (RE), to compose RTMs. This is because the operation of RE is intuitively easy to understand, and RTMs can be constructed by it very simply.

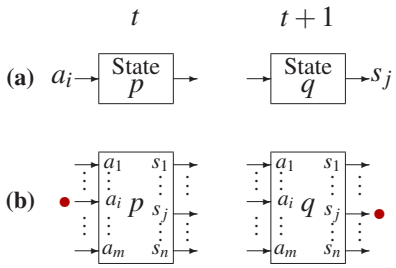


Figure 2: **(a)** A sequential machine with $\delta(p, a_i) = (q, s_j)$, and **(b)** an interpretation of it as a module having decoded input ports and output ports

3.1 Rotary element (RE), a typical RLEM

A *rotary element* (RE) [13] is a two-state RLEM that has four input ports and four output ports, and is depicted as in Fig. 3. Intuitively, an RE has a rotatable bar inside, and an incoming signal is controlled by the bar. It takes either of the two states, state V or state H, depending on the direction of the bar. If the direction of a coming signal is parallel to the bar, the signal goes straight ahead, and the state does not change (Fig. 4 **(a)**). If the direction of a coming signal is orthogonal to the bar, the signal turns right, and the state changes (Fig. 4 **(b)**).

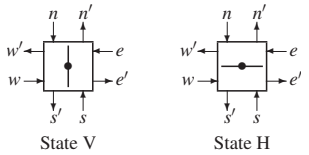


Figure 3: Two states of a rotary element (RE)

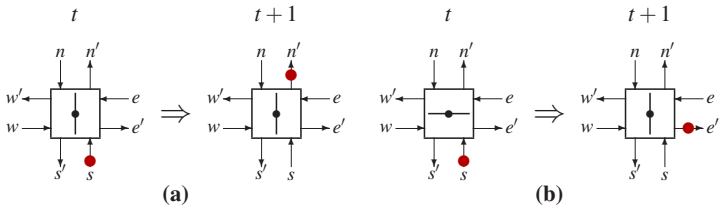


Figure 4: Operations of an RE. **(a)** Parallel case, and **(b)** orthogonal case

3.2 Constructing reversible sequential machines using REs

We can construct any RSM using only REs. To do so, we introduce a circuit module called an RE-column. RSMs are composed of it systematically.

3.2.1 RE-column, a module for building RSMs

An *RE-column* of degree n is shown in Fig. 5, which has $n+1$ REs. We assume, in a resting state, it is in the state **(a)** or **(b)** of Fig. 5, where all the REs except the bottom are in the state V. It has $2n$ input ports $a_1, \ldots, a_n, b_1, \ldots, b_n$, and $2n$ output ports $s_1, \ldots, s_n, t_1, \ldots, t_n$. If a signal is given to one of the input ports, the module will take a state other than those of **(a)** and **(b)**. However, as we shall see, the module will become again the state **(a)** or **(b)** when the signal goes out from it. Therefore, an RE-column behaves as if it is a two-state RSM. That is to say, the states **(a)** and **(b)** are macroscopic states 0 and 1 of the RE-column.

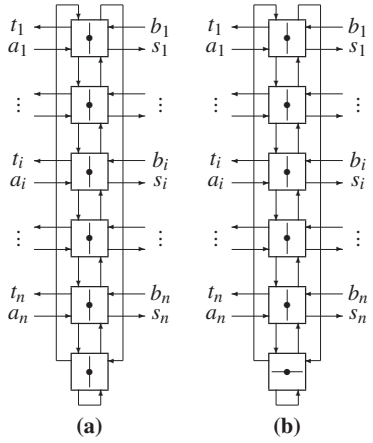


Figure 5: RE-column of degree n. **(a)** State 0, and **(b)** state 1

Table 1: The move function of an RE-column of degree n. Here, $i \in \{1, \ldots, n\}$

	Input	
Present state	a_i	b_i
0	$0\, s_i$	$1\, s_i$
1	$0\, t_i$	$1\, t_i$

The move function of the RE-column as a two-state RSM is shown in Table 1. In the following, we examine how the circuit works for the four cases of state-input pairs: $(0, a_i)$, $(1, a_i)$, $(0, b_i)$, and $(1, b_i)$, where $i \in \{1, \ldots, n\}$.

First, consider the case where the state is 0 (Fig. 5 **(a)**) and a signal is given to a_i. By the signal from a_i, the i-th RE changes its state from V to H. Then the signal moves downward through the $(n - i + 1)$ REs. At the bottom of the column the signal makes a U-turn, and goes upward through the $(n - i + 1)$ REs. At the i-th RE, the signal turns right and changes the RE's state from H to V. Finally the signal goes out from the port s_i. In this case the RE-column keeps the state 0.

Second, consider the case where the state is 1 (Fig. 5 **(b)**) and a signal is given to a_i. As in the first case, the signal sets the i-th RE to the state H, and then moves downward. At the bottom RE, the signal makes a right-turn, and changes the state of the RE to V. The signal goes upward along the left vertical line, and reaches the north input of the top RE. It moves downward through the $(i - 1)$ REs, and makes a right-turn at the i-th RE, restoring the RE's state to V. Finally the signal goes out from the port t_i. In this case the RE-column changes the state from 1 to 0.

Third, consider the case where the state is 0 and a signal is given to b_i. The signal sets the i-th RE to the state H, and moves upward through the $(i - 1)$ REs. Then the signal goes downward along the right vertical line, and reaches the east input of the bottom RE. It changes the state of the bottom RE to H, and moves upward through the $(n - i)$ REs. The signal makes a right-turn at the i-th RE, and restores its state to V. Finally the signal goes out from the port s_i. In this case the RE-column changes the state from 0 to 1.

Fourth, consider the case where the state is 1 and a signal is given to b_i. As in the third case, the signal sets the i-th RE to the state H, and reaches the east input of the bottom RE. The signal goes out from the west output of the bottom RE without changing its state. It moves upward along the left vertical line, and reaches the north input of the top RE. It makes a right-turn at the i-th RE, restoring the RE's state to V. Finally the signal goes out from the port t_i. In this case the RE-column keeps the state 1.

3.2.2 Composing RSMs using RE-columns

We can systematically compose any RSM out of RE-columns. The composing method is explained by the following example of an RSM M_0.

$$M_0 = (\{q_1, q_2, q_3\}, \{c_1, c_2, c_3\}, \{d_1, d_2, d_3\}, \delta_0)$$

The move function δ_0 is given in Table 2.

Fig. 6 shows the circuit that simulates M_0. It consists of three RE-columns of degree 3. The j-th RE-column corresponds to the j-th state q_j of M_0. The i-th row except the bottom row corresponds to the input symbol c_i and the output symbol d_i. If the state of M_0 is q_j, then the state of the j-th RE-column is set to 1, while the other RE-columns are set to 0. Fig. 6 shows that M_0 is in the state q_3.

For example, assume an input signal is given to the port c_2. Since the first two RE-columns are in the state 0, the signal goes rightward through these RE-columns without changing their states. At the third RE-column, the signal changes the RE-column's state from 1 to 0, and then comes out from the west output port of the second RE, which is labeled by q_3c_2. This port is connected to the east input port of the third RE of the second RE-column labeled by q_2d_3. By this, the state of the second RE-column changes from 0 to 1. The signal appears from the east output port of the third RE in the second RE-column. Since the third RE-column is now in the state 0, the signal finally goes out from the port d_3. By above, the operation $\delta_0(q_3, c_2) = (q_2, d_3)$ is simulated. Other cases are similar to this case.

Table 2: The move function δ_0 of an RSM M_0

	Input		
Present state	c_1	c_2	c_3
q_1	$q_2\, d_2$	$q_3\, d_1$	$q_1\, d_2$
q_2	$q_3\, d_2$	$q_2\, d_1$	$q_1\, d_3$
q_3	$q_3\, d_3$	$q_2\, d_3$	$q_1\, d_1$

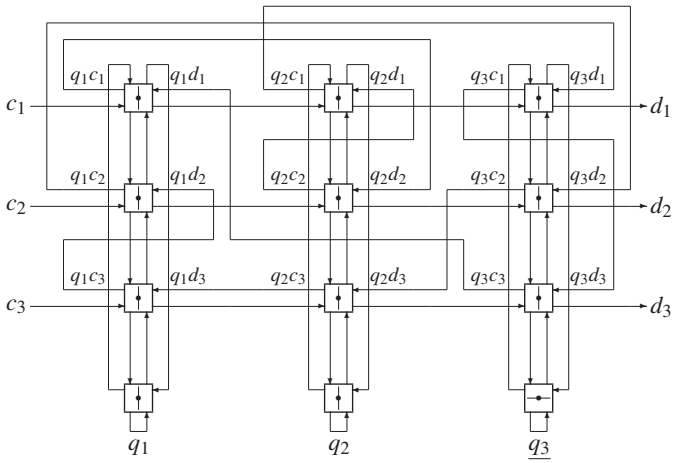


Figure 6: An RSM M_0 composed only of RE [15]

Generally, for any given RSM $M = (\{q_1, \ldots, q_n\}, \{c_1, \ldots, c_l\}, \{d_1, \ldots, d_m\}, \delta)$, we can construct a circuit composed of RE-columns that simulates M in the following way. First prepare n RE-columns of degree $r = \max\{l, m\}$, and connect the s_i output of the j-th RE-column to the a_i input of the $(j + 1)$-st RE-column $(i \in \{1, \ldots, r\}, j \in \{1, \ldots, n - 1\})$. Also connect c_i to a_i of the first RE-column

($i \in \{1, \ldots, l\}$), and s_i of the n-th RE-column to d_i ($i \in \{1, \ldots, m\}$). For all q_h, q_j, c_i, and d_k, if $\delta(q_h, c_i) = (q_j, d_k)$, then connect the west output of the RE at (i, h) to the east input of the RE at (k, j). By this, M is correctly simulated.

3.3 Constructing reversible Turing machines using REs

Using only REs, we can compose any two-symbol RTM with a rightward infinite tape. A composing method was first given in [13]. Then it was revised in [15], and it is further revised here. An RTM is constructed by assembling two kinds of functional modules. They are a tape cell module and a state module. Note that the tape cell module uses an RE-column as a submodule.

3.3.1 Tape cell module

A *tape cell module* is a circuit shown in Fig. 7. It simulates one tape square of an RTM. Connecting infinite number of copies of it, a tape unit is obtained as shown in the right part of Fig. 12.

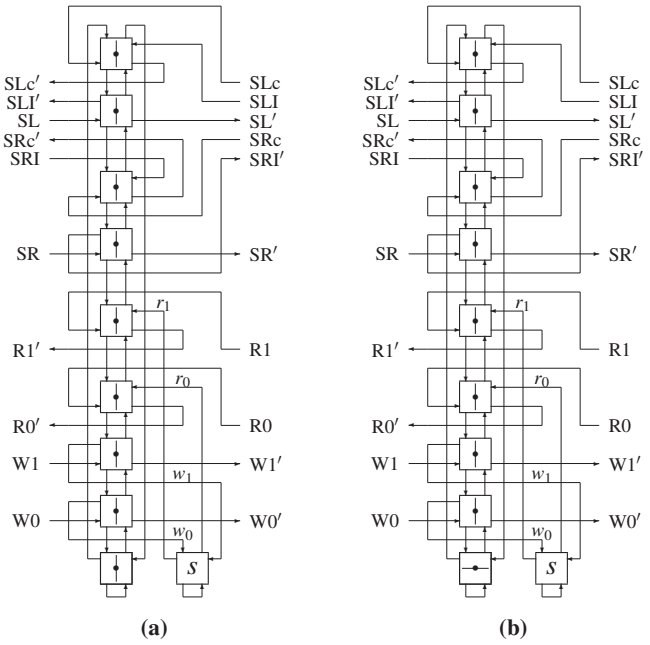


Figure 7: Tape cell module for two-symbol RTMs. The state **(a)** shows that the head is not on this cell, and **(b)** shows that the head is on this cell

The tape cell keeps the information whether the head of the RTM is on this cell or not in its left part, which is an RE-column of degree 8 (Fig. 5). If the RE-column is in the state 0 (i.e., its bottom RE is in the state V), then the head is not here (Fig. 7 **(a)**). If it is in the state 1 (i.e., its bottom RE is in the state H), then the head is here (Fig. 7 **(b)**).

A tape symbol $s \in \{0, 1\}$ is stored in the RE indicated by s in Fig. 7, where 0 and 1 are represented by the states V and H, respectively. The right part of the tape cell is, in fact, a one-bit memory (Fig. 8) having the move function given in Table 3. It has two input ports w_0 and w_1, and two output ports r_0 and r_1. Assume the present state is s ($\in \{0, 1\}$). If a signal is given to w_t ($t \in \{0, 1\}$), then the new symbol t is written in it, and the old symbol s is read-out from the output port r_s. Thus, a write operation always accompanies a read operation.

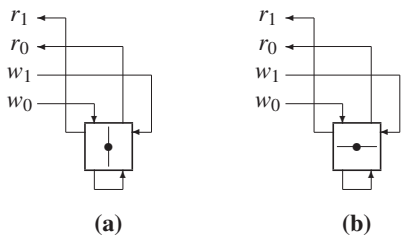


Figure 8: One-bit memory for a tape cell module. **(a)** State 0, and **(b)** state 1

Table 3: The move function of the one-bit memory given in Fig. 8

	Input	
Present state	w_0	w_1
0	$0\, r_0$	$1\, r_0$
1	$0\, r_1$	$1\, r_1$

The tape cell module has ten input ports corresponding to ten kinds of input symbols listed in Table 4. They are interpreted as instructions to the tape unit or response signals to the finite control of an RTM. For each input symbol, there is a corresponding output symbol, which is indicated by the symbol with $'$, and thus the tape cell has ten input ports and ten output ports. Making an infinite number of copies of it, and connecting them to form a rightward infinite array, we can obtain a tape unit for the RTM. To the left of the tape unit a finite control of an RTM will be connected. We assume there is only one tape cell whose RE-column is in the state 1 in the initial setting, and thus there is only one head. Giving a signal to the tape unit, read/write and head-shift operations are performed.

Table 4: Ten kinds of symbols for the tape cell module and their meanings [15]

Symbol	Instruction/Response	Meaning
W0	Write 0	Instruction of writing the tape symbol 0 at the head position. By this instruction, read operation is also performed
W1	Write 1	Instruction of writing the tape symbol 1 at the head position. By this instruction, read operation is also performed
R0	Read 0	Response signal telling the read symbol at the head is 0
R1	Read 1	Response signal telling the read symbol at the head is 1
SL	Shift-left	Instruction of shift-left operation
SLI	Shift-left immediate	Instruction of placing the head on this cell by shifting left
SLc	Shift-left completed	Response (completion) signal of shift-left operation
SR	Shift-right	Instruction of shift-right operation
SRI	Shift-right immediate	Instruction of placing the head on this cell by shifting right
SRc	Shift-right completed	Response (completion) signal of shift-right operation

First, consider the case where the head is not on this tape cell (Fig. 7 **(a)**). Since its RE-column is in the state 0, a signal from the input port W0, W1, R0, R1, SL, SR, SLc, or SRc simply goes to the output port W0′, W1′, R0′, R1′, SL′, SR′, SLc′, or SRc′, respectively, without changing its state (see Table 1). It means that these signals skip tape cells having no head. Note that processing of a signal SLI or SRI in this case is discussed later.

Second, consider the case where the head is on this cell (Fig. 7 **(b)**). The first subcase is that an input signal Wt ($t \in \{0,1\}$) is given, which is for writing the tape symbol t in this tape cell. We assume the one-bit memory in its right part is in the state s. The signal changes the state of the RE-column to 0, and appears on the line w_t in Fig. 7 (see Table 1). Then the state of the one-bit memory changes to t, and the signal appears on the line r_s (see Table 3). This signal restores the RE-column to the state 1, and finally goes out from the port Rs'. Hence, the writing operation also performs a reading operation to keep reversibility of the tape cell. The signal Rs' moves leftward through tape cells having no head, and finally reaches the finite control of the RTM. Note that if an RTM needs to read a tape symbol, it is performed by sending a signal to the input port W0 of the tape unit. By this, the tape unit gives a response signal at the output port Rs', and the tape symbol at the head position is cleared to 0. Thus, this is a destructive readout.

The second subcase is that a signal is given to the input port SL of the tape cell with a tape head, which will shift the tape head to the left. This signal changes the RE-column to the state 0, and goes out from the port SLI′ (Table 1). This signal is sent to the left-neighboring tape cell. If the latter tape cell receives an input signal SLI, then it sets the state of the RE-column to 1, and sends an output signal SLc′ to the left. By such a process, shift-left operation is performed correctly.

The third subcase is that a signal is given to the input port SR of the tape cell

with a tape head, which will shift the tape head to the right. The ports SR, SRI, and SRc are similar to the ports SL, SLI, and SLc, except that an output signal SRI′ is sent to the right-neighboring tape cell.

By above, we can see that read/write and head-shift operations are correctly performed by a tape unit.

3.3.2 State module

Before introducing a state module we first explain a subroutine call mechanism. A *subroutine* is a black box having at least one calling (i.e., input) port, and at least one return (i.e., output) port (Fig. 9) that satisfies the following: If a calling signal is given to one of the calling ports, a return signal eventually comes out from one of the return ports. No signal should be given to a calling port before a return signal for the previous calling signal comes out. Here, to make a simple subroutine-call mechanism using REs, we restrict both the numbers of calling ports and return ports to be at most two. If there are two calling ports, say c^0 and c^1, we can give two kinds of information 0 and 1, regarded as an input argument, to the subroutine. Likewise, if there are two return ports r^0 and r^1, we can obtain two kinds of information 0 and 1, regarded as an output value, from the subroutine.

A tape unit acts as three subroutines by suitably specifying calling and return ports. The first one is the subroutine having the calling ports W0 and W1, and the return ports R0′ and R1′. The second has the calling port SL, and the return port SLc′. The third has the calling port SR, and the return port SRc′.

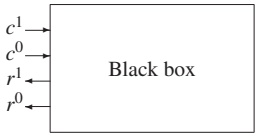


Figure 9: Subroutine. Here, it has two input ports c^0 and c^1 for calling it from a main routine, and two output ports r^0 and r^1 for returning to the main routine

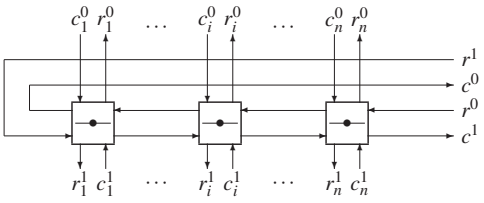


Figure 10: Subroutine caller that can be used from n points of a main routine

A *subroutine caller* is a mechanism for calling one subroutine from many points of a main routine. Fig. 10 is a caller for a subroutine having calling ports c^0 and c^1, and return ports r^0 and r^1. In this figure, c_i^s ($i \in \{1, \ldots, n\}$, $s \in \{0, 1\}$) is the i-th calling port for the main routine with the input s, and r_i^t ($i \in \{1, \ldots, n\}$, $t \in \{0, 1\}$) is the i-th return port for the main routine with the output t.

Initially, all the REs in Fig. 10 are set to the state H. If a signal is given to the port c_i^s ($i \in \{1, \ldots, n\}$, $s \in \{0, 1\}$), then the state of the i-th RE changes to V, and the signal goes out from the port c^s. If the signal returns via the port r^t ($t \in \{0, 1\}$), then the state of the i-th RE is restored to H, and then the signal goes out from the port r_i^t. In this way, n points of the main routine can share the same subroutine. Note that if a subroutine has only one calling port or only one return port, then unnecessary lines in the caller are removed.

A *state module* simulates one state, say q_i, of an RTM. It is shown in Fig. 11. It is composed of three submodules, which are *write-and-merge*, *head-shift*, and *read-and-branch* submodules. This figure shows the case where q_i is a right-shift state. The case for a left-shift state is similar. Because of the reversibility condition (Definition 2.2), shift direction is uniquely determined by the state. Note that a state module for an initial state consists only of a read-and-branch submodule, and that for a halting state consists of write-and-merge and head-shift submodules.

If the number of states of an RTM is m, then prepare m state modules, and connect them in a row to make a finite control of the RTM. At the left end of the array, SLc′, SRc′, R1′ and R0′ are connected to SL, SR, W1 and W0, respectively.

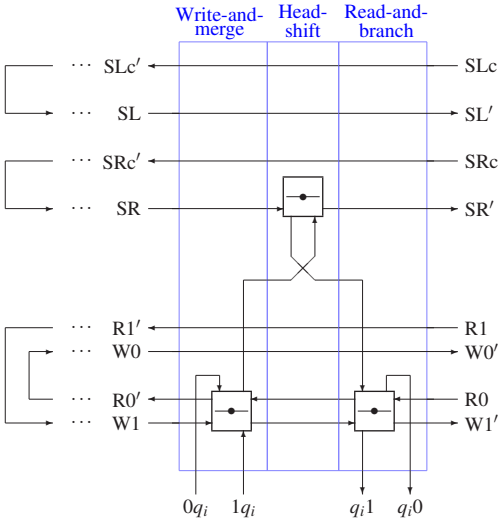


Figure 11: State module for a right-shift state q_i of an RTM

Write-and-merge submodules and read-and-branch submodules in the m state modules form a subroutine caller that share the subroutine having the calling ports W0 and W1, and the return ports R0′ and R1′ in the tape unit. By this, read/write operations on the tape unit can be performed in each state. Head-shift submodules of the right-shift states also form a subroutine caller that share the subroutine having the calling port SR, and the return port SRc′ in the tape unit. Likewise, head-shift submodules of the left-shift states form a subroutine caller that share the subroutine having the calling port SL, and the return port SLc′.

Let $T = (Q, \{0, 1\}, q_0, F, 0, \delta)$ be a two-symbol RTM. First consider how the write-and-merge submodule works. Assume $[p, s, 0, d, q_i], [p', s', 1, d, q_i] \in \delta$. Note that it also works well for the case only one of these two quintuples exists. We further assume that the write-and-merge operation is done just after a read-and-branch operation that performs a destructive readout. Thus, the tape symbol at the head position is now 0. If the submodule receives a signal from the input port $0q_i$ ($1q_i$, respectively), then it sends a calling signal to the subroutine from the port W0′ (W1′) to perform a writing operation. Since the old symbol at the head position is 0, the submodule receives a signal from the return port R0 in both cases of writing 0 and 1. By this, two different signal paths of writing 0 and 1 are reversibly merged into one, and the signal is sent to the head-shift submodule.

The head-shift submodule works as follows. If it receives a signal from the write-and-merge submodule, it sends a signal to the calling port SL′ or SR′, by which shifting is performed in the tape unit. It receives a signal from the return port SLc or SRc. Then, the signal is sent to the read-and-branch submodule.

If the read-and-merge submodule receives a signal from the head-shift submodule, it sends a signal to the calling port W0′ of the tape unit. Then the tape unit sends back a response signal via the return port R0 or R1 depending on the read symbol. The submodule finally gives a signal to the port q_i0 or q_i1. By above, read-and-branch operation is performed.

State transitions of the RTM is realized by connecting state modules in the following way. If there is a quintuple $[q_i, s, t, d, q_j] \in \delta$, then the output port q_is of the state module for q_i is connected to the input port tq_j of the state module for q_j.

3.3.3 Composing RTMs

Assembling tape cells and state modules, and connecting them as explained above, we can systematically compose a circuit made of REs that simulates any given two-symbol RTM. The circuit for the RTM T_{parity} in Example 2.1 is shown in Fig. 12. If we give a signal to the port “Start”, then it is sent to the state module for the initial state. By this, T_{parity} begins to compute. If T_{parity} halts, then the signal from the state module corresponding to the accepting or rejecting state is sent to the port “Accept” or “Reject” showing that the computation is completed.

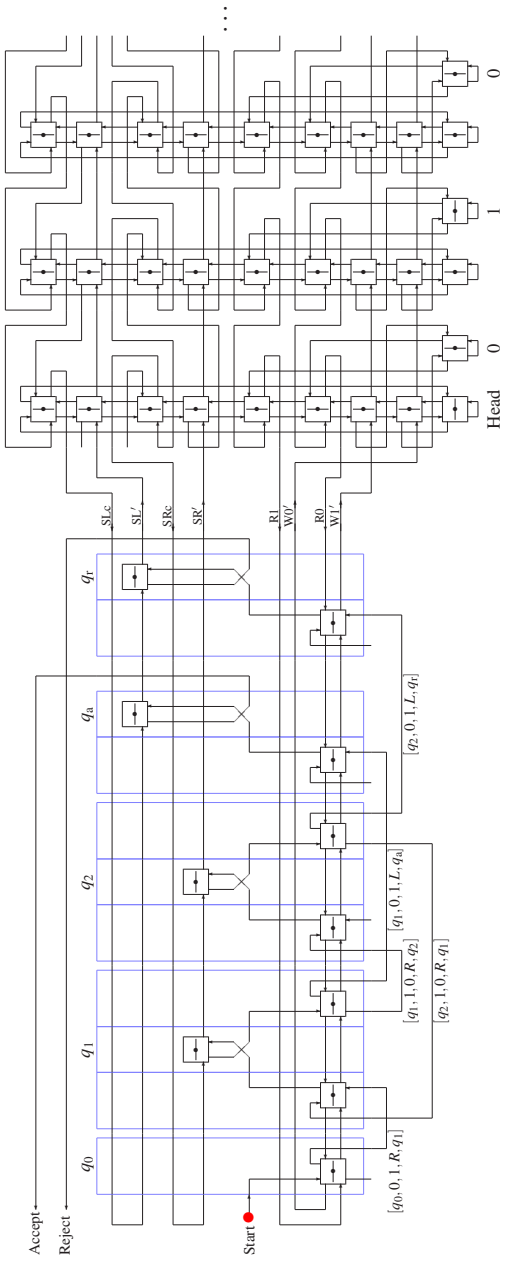


Figure 12: RTM T_{parity} composed of REs

3.4 Universality of RLEMs

There are infinitely many RLEMs even if we consider only two-state RLEMs. We use a special graphical representation for two-state RLEMs. Fig. 13 shows the representation of RLEM 3-10, where "3" means that it has three input/output symbols, and "10" is its serial number in the class of 3-symbol RLEMs. Two boxes in Fig. 13 indicate its two states. The dotted and solid lines give the input-output relation in each state. If an input signal goes through a dotted line, the state does not change (Fig. 14 **(a)**). If it goes through a solid line, the state changes (Fig. 14 **(b)**). Note that RE can be also represented by such a figure, but we employ Fig. 3 for ease in understanding.

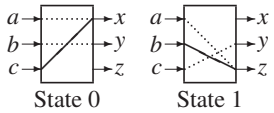


Figure 13: Two states of RLEM 3-10.

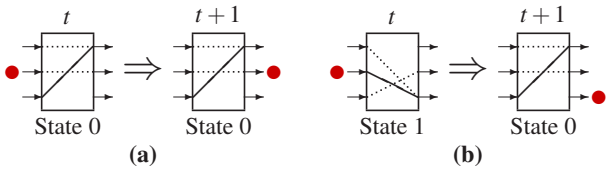


Figure 14: Operations of RLEM 3-10. **(a)** The case where the state does not change, and **(b)** the case where the state changes

Among RLEMs there are universal RLEMs in the following sense.

Definition 3.2. An RLEM R is called *universal* if any RSM can be realized by a circuit composed only of R.

As we have already seen in Sect. 3.2.2, RE is universal. We can observe that RLEM 3-10 is also universal, since RE can be composed of RLEM 3-10 as in Fig. 15 [20]. This figure shows the state H of an RE. By complementing the states of the bottom four RLEMs, we have the state V of an RE. In Fig. 15, it is easy to see that if a signal is given to the port e (w, respectively), then it goes out from the port w' (e') in two steps without changing the state of the circuit. This is a parallel case (Fig. 4 **(a)**). On the other hand, if a signal is given to the port n, then the circuit evolves as shown in Fig. 16. The signal finally goes out from the port w', and the states of the bottom four RLEMs are complemented. This is an orthogonal case (Fig. 4 **(b)**). In such a way, RE is correctly simulated.

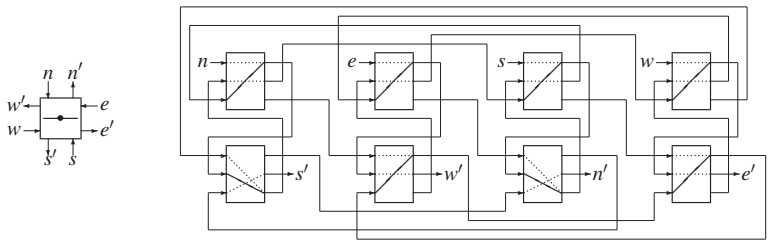


Figure 15: A circuit composed of RLEM 3-10 that simulates RE [20].

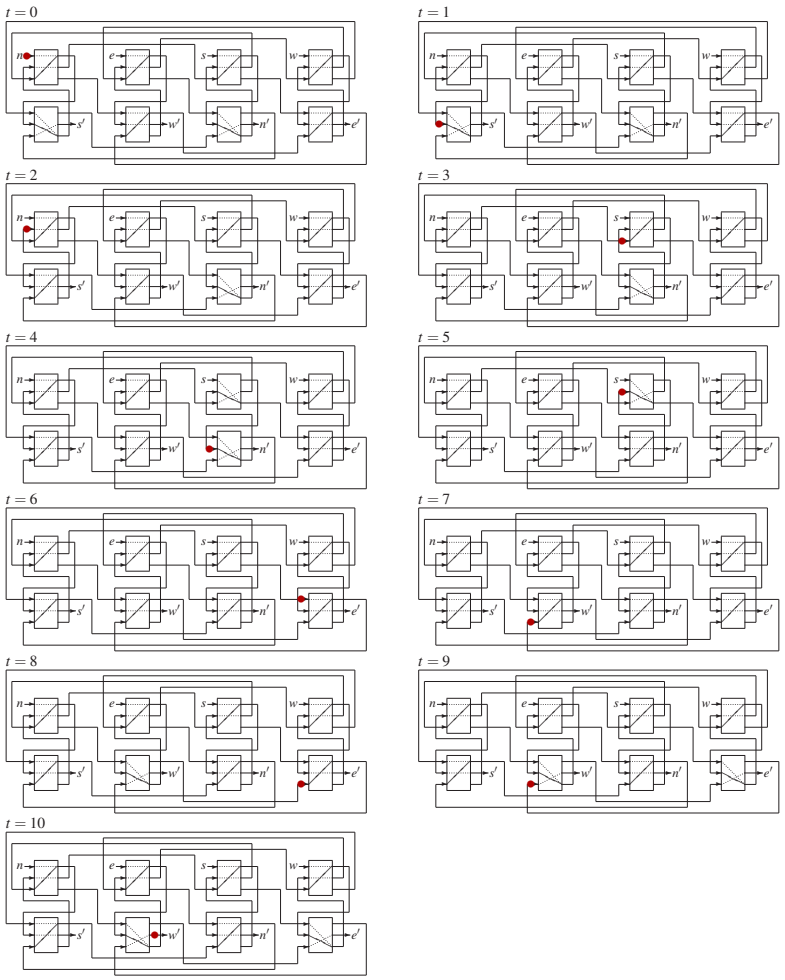


Figure 16: Process of simulating RE by RLEM 3-10 when the state of RE changes

Furthermore, we can compose RLEM 3-10 out of RLEMs 2-3 and 2-4 (Fig. 17). The circuit that simulates RLEM 3-10 is shown in Fig. 18 [10]. Hence, the set {RLEM 2-3, RLEM 2-4} is universal, though each of RLEM 2-3 and RLEM 2-4 has been proved to be non-universal [21]. This result is useful for realizing a universal RLEM such as RE in a reversible environment having a very simple microscopic law of evolution (see Sect. 4.5).

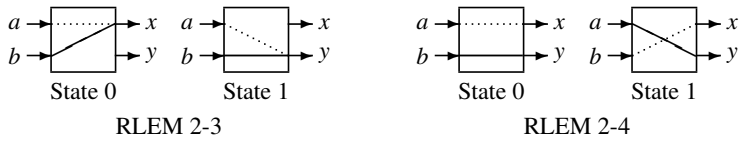


Figure 17: RLEMs 2-3 and 2-4

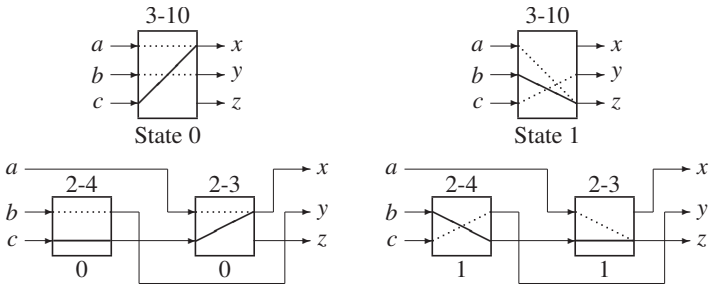


Figure 18: RLEM 3-10 is simulated by a circuit made of RLEMs 2-3 and 2-4 [10]

It is known that *every* non-degenerate two-state RLEM having three or more I/O symbols is universal. It was proved by showing the fact that, for any one of these RLEMs, there are circuits composed only of it that simulate RLEMs 2-3 and 2-4 [20]. Note that, here, a degenerate RLEM means that it is either equivalent to an RLEM with fewer I/O symbols, or equivalent to connecting wires (see [15] for its precise definition). Hence, we consider only nondegenerate RLEMs. Fig. 19 summarizes the results.

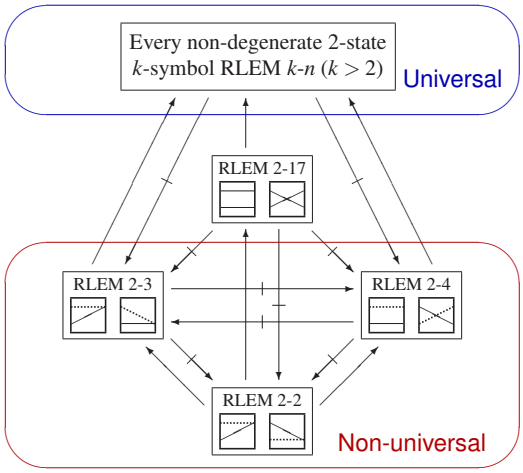


Figure 19: Universality/non-universality of two-state RLEMs [15]

4 Simple Reversible Cellular Automaton

As a reversible environment, we use a reversible cellular automaton having very simple local transition rules, which can be seen as a microscopic law of evolution.

4.1 Elementary square partitioned CA (ESPCA)

A 4-neighbor *square partitioned cellular automaton* (SPCA) is a two-dimensional CA whose square cell is divided into four parts as in Fig. 20 **(a)**. The next state of a cell is determined depending on the present states of the four adjacent parts of the neighboring cells (not depending on whole the states of the four neighboring cells) as shown in Fig. 20 **(b)**. Note that the next state of a cell does not depend on the previous state of the cell itself.

Definition 4.1. A 4-neighbor *square partitioned cellular automaton* (SPCA) is defined by

$$P = (\mathbb{Z}^2, (T, R, B, L), ((0,-1), (-1,0), (0,1), (1,0)), f).$$

Here, $\mathbb{Z}^2$ is the set of all points with integer coordinates where cells are placed. The items T, R, B and L are non-empty finite sets of states of the top, right, bottom and left parts of a cell. The set of states of a cell is thus $Q = T \times R \times B \times L$. The quadruple $((0,-1), (-1,0), (0,1), (1,0))$ is a *neighborhood*. The item $f : Q \to Q$ is a *local (transition) function*.

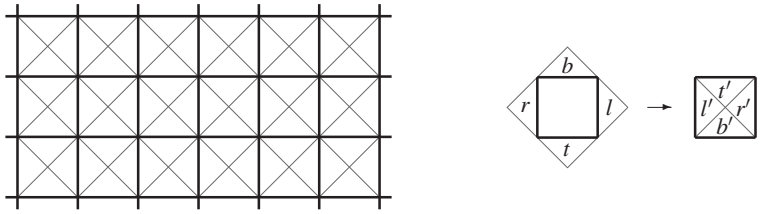


Figure 20: **(a)** Cellular space of a 4-neighbor square partitioned cellular automaton (SPCA), and **(b)** its local transition rule $f(t, r, b, l) = (t', r', b', l')$

If $f(t, r, b, l) = (t', r', b', l')$ holds for $(t, r, b, l), (t', r', b', l') \in Q$, this relation is called a *local transition rule* of P. It is also indicated as in Fig. 20 **(b)**. The local function f is thus defined by a set of local transition rules.

Definition 4.2. Let $P = (\mathbb{Z}^2, (T, R, B, L), ((0, -1), (-1, 0), (0, 1), (1, 0)), f)$ be an SPCA. A *configuration* of P is a function $\alpha : \mathbb{Z}^2 \to Q$. The set of all configurations of P is denoted by $\mathrm{Conf}(P)$, i.e., $\mathrm{Conf}(P) = \{\alpha \mid \alpha : \mathbb{Z}^2 \to Q\}$. Let $\mathrm{pr}_T : Q \to T$ be the *projection function* that satisfies $\mathrm{pr}_T(t, r, b, l) = t$ for all $(t, r, b, l) \in Q$. The projection functions $\mathrm{pr}_R : Q \to R$, $\mathrm{pr}_B : Q \to B$ and $\mathrm{pr}_L : Q \to L$ are defined similarly. The *global function* $F : \mathrm{Conf}(P) \to \mathrm{Conf}(P)$ of P is defined as the one that satisfies the following.

$$\begin{aligned} &\forall \alpha \in \mathrm{Conf}(P), \forall (x, y) \in \mathbb{Z}^2 : \\ &\quad F(\alpha)(x, y) \quad = \quad f(\mathrm{pr}_T(\alpha(x, y-1)), \mathrm{pr}_R(\alpha(x-1, y)), \mathrm{pr}_B(\alpha(x, y+1)), \\ &\qquad\qquad\qquad\qquad \mathrm{pr}_L(\alpha(x+1, y))) \end{aligned}$$

Definition 4.3. An SPCA P is called *reversible* if its global function is injective.

As for the notions related to reversibility, see Sect. 10.3 of [15] for the details. The next Lemma shows that injectivity of the global function of a PCA is equivalent to that of the local function [15, 19]. By this, we can easily obtain a reversible CA, since it is sufficient to design a PCA whose local function is injective.

Lemma 4.1. *Let P be an SPCA. Its global function F is injective if and only if its local function f is injective.*

Here, we define the simplest subclass of SPCAs such that its local function is rotation-symmetric, and each of four parts has only two states. It is called an *elementary* SPCA (ESPCA) as in the case of a one-dimensional *elementary cellular automaton* (ECA) [28]. We first define the notion of rotation-symmetry.

Definition 4.4. Let $P = (\mathbb{Z}^2, (T, R, B, L), ((0,-1), (-1,0), (0,1), (1,0)), f)$ be an SPCA. The SPCA P is called *rotation-symmetric* (or *isotropic*) if the following conditions (1) and (2) hold.

(1) $T = R = B = L$

(2) $\forall (t, r, b, l), (t', r', b', l') \in T \times R \times B \times L :$
$f(t, r, b, l) = (t', r', b', l') \Rightarrow f(r, b, l, t) = (r', b', l', t')$

Definition 4.5. Let $P = (\mathbb{Z}^2, (T, R, B, L), ((0,-1), (-1,0), (0,1), (1,0)), f)$ be an SPCA. We say P is an *elementary triangular partitioned cellular automaton* (ESPCA), if $T = R = B = L = \{0, 1\}$, and it is rotation-symmetric.

Since an ESPCA is rotation-symmetric, its local function $f : \{0,1\}^4 \to \{0,1\}^4$ is defined by only six local transition rules, which are described by the following six values.

$$f(0,0,0,0),\ f(0,0,1,0),\ f(0,0,1,1),\ f(1,0,1,0),\ f(0,1,1,1),\ f(1,1,1,1)$$

Here, $f(0,0,1,0),\ f(0,0,1,1),\ f(0,1,1,1) \in \{0,1\}^4$. However, $f(1,0,1,0) \in \{(0,0,0,0),\ (0,1,0,1),\ (1,0,1,0),\ (1,1,1,1)\}$ and $f(0,0,0,0), f(1,1,1,1) \in \{(0,0,0,0), (1,1,1,1)\}$, since it is rotation-symmetric. Hence, there are $16^3 \times 4 \times 2^2 = 65,536$ ESPCAs in total.

Reading the 4-bit values of $f(0,0,0,0),\ f(0,0,1,0),\ f(0,0,1,1),\ f(1,0,1,0),\ f(0,1,1,1),\ f(1,1,1,1)$ as six binary numbers, we can express an ESPCA by a 6-digit hexadecimal identification number *uvwxyz*. For example, if $f(0,0,1,0) = (t, r, b, l)$, then $v = 2^3t + 2^2r + 2^1b + 2^0l$. An ESPCA with the identification number *uvwxyz* is denoted by ESPCA *uvwxyz*.

4.2 A particular ESPCA P_0

Here, we consider a particular reversible ESPCA with the identification number 01caef. Hereafter, it is denoted by P_0 for short. It is first studied in [17]. Fig. 21 shows a pictorial representation of the six local transition rules of ESPCA 01caef. Though its local function is simple, its behavior is complex. Therefore, it is generally difficult to follow evolution processes of P_0 using only paper and pencil. To see its evolution processes, we created an emulator of P_0 on the general purpose CA simulator *Golly* [27]. The emulator files and pattern files for P_0 are available in [16].

It is easy to see that the local function of P_0 is injective. Therefore it is a reversible ESPCA. An ESPCA is called *conservative* if the number of particles (i.e., state 1) is conserved in each local transition rule. It is an analog of various conservation laws in physics. We can see that ESPCA P_0 is conservative.

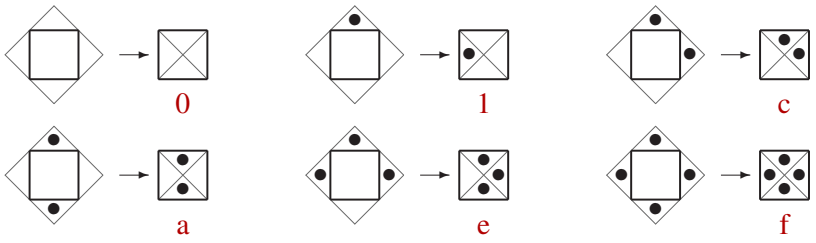


Figure 21: Local function defined by the six local transition rules of a particular reversible and conservative ESPCA 01caef, which is denoted by P_0 hereafter

4.3 Useful patterns in ESPCA P_0

A *pattern* is a finite segment of a configuration (see Sect. 10.2.1 of [15] for its precise definition). A *periodic pattern* is one such that the same pattern appears at the same position after some time steps. The periodic pattern given in Fig. 22 is called a *blinker*. It is of period 2. Though there are many kinds of periodic patterns in P_0, a blinker is particularly useful among them as we shall see in Sect. 4.4

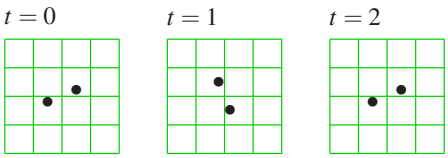


Figure 22: Blinker in P_0 [17]. It is of period 2

A *space-moving pattern* is one such that the same pattern appears at a different position after some time steps. In P_0 there are many kinds of space-moving patterns of various periods [17]. In fact, if we start from a random-like pattern, then we often observe that space-moving patterns appear.

The pattern having the shortest period among the space-moving patterns so far found is called a *glider* (Fig. 23). It travels one cell diagonally in 12 steps. It will be used as a signal when constructing reversible Turing machines, since it has interesting and desirable properties as described in Sect. 4.4.

The pattern shown in Fig. 24 is called a *block*. It is a *stable pattern*, which is a periodic pattern of period 1, and thus does not change its pattern if no other pattern touches it. In the following, it will be used only for writing comments and indicating a border of a logic element in the cellular space. Hence, it has no functional role for composing reversible Turing machines.

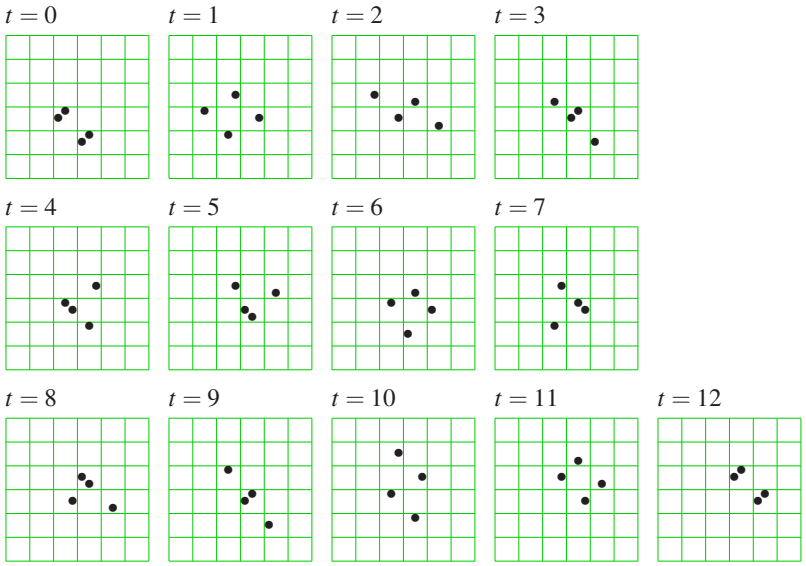


Figure 23: Glider in P_0 [17]. It is of period 12

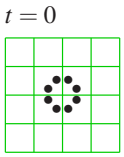


Figure 24: Block in P_0 [17]. It is a stable pattern

4.4 Three useful phenomena in ESPCA P_0

We make three experiments of interacting a glider with a blinker. However, each of these experiment needs a large number of steps. In particular the third experiment takes more than 2000 steps. Therefore, it is not possible to show correctness of the results in this paper. They are verified by computer simulation [16].

The first experiment is shown in Fig. 25. Colliding a glider with a blinker in this manner, a right-turn of a glider is realized.

The second experiment is in Fig. 26. By this, a glider makes a U-turn. It is used to test if a blinker exists or not at a specified position. It is also used to reversibly merge two signal paths into one (it is explained in Sect. 4.5).

The third experiment is in Fig. 27. By this, the position of the blinker is shifted by 6 cells, and the glider makes a right-turn. Using this phenomenon, a kind of

memory device is realized, where the memory states are kept by the positions of the blinker. At the same time, it can test if a blinker exists at a specified position, and can merge two signal paths into one.

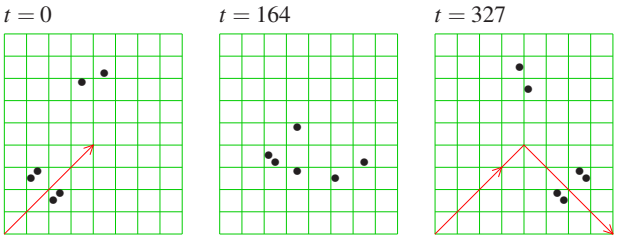


Figure 25: Right-turn of a glider in P_0 [17]

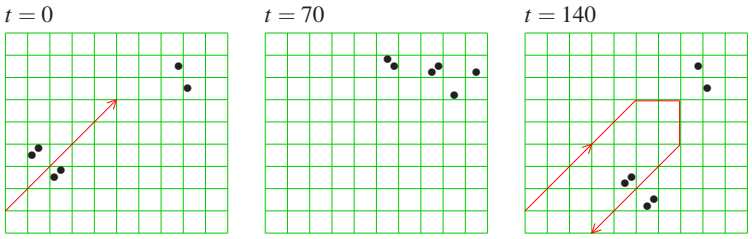


Figure 26: U-turn of a glider in P_0 [17]

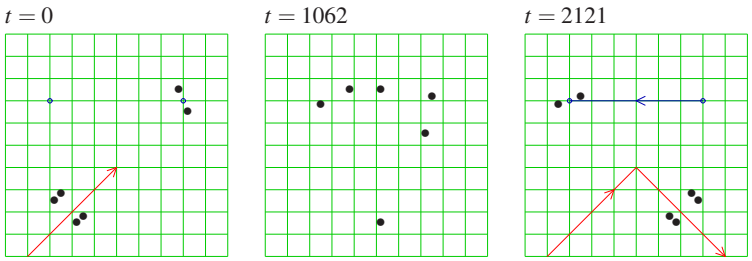


Figure 27: Shifting a blinker by a glider in P_0 [17]

4.5 Composing RLEMs in ESPCA P_0

Using three useful phenomena given in Sect. 4.4, we implement RLEMs in P_0. Since it is difficult to make RE directly, we first compose RLEMs 2-3 and 2-4. Using them, we make RLEM 3-10, and then RE. In [17], RLEM 4-31 is implemented in P_0, whose pattern size is smaller than that of RE. However, here, we use RE, since its operation is easier to understand. Note that the essential parts of RLEMs constructed here consist only of blinkers, since blocks are used to write comments and to indicate borders of the RLEMs.

RLEM 2-3

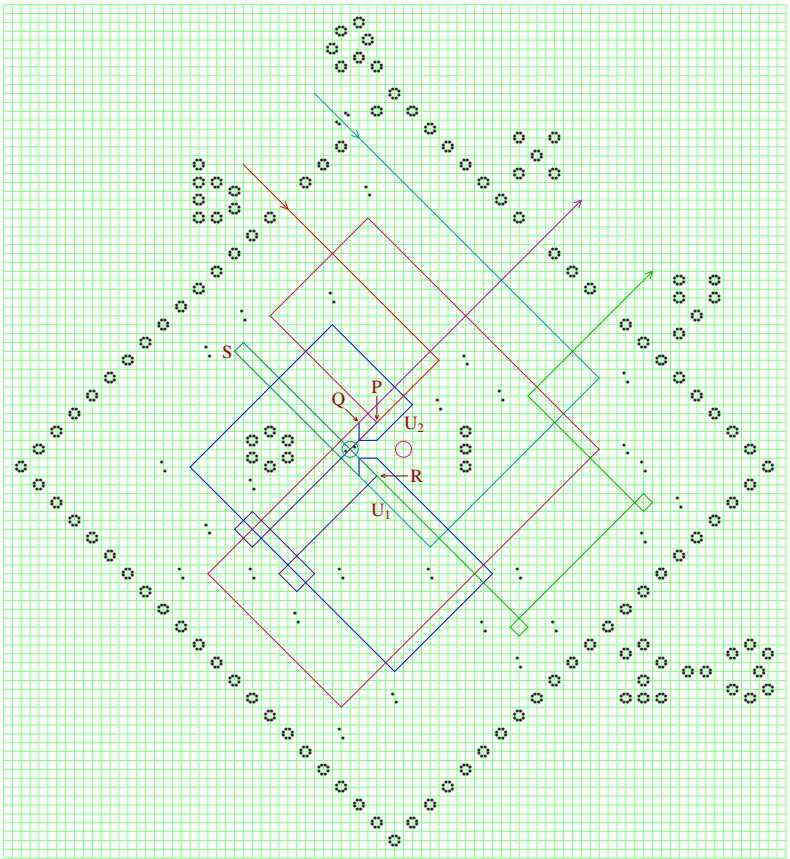


Figure 28: RLEM 2-3 implemented in P_0

The pattern shown in Fig. 28 simulates RLEM 2-3. There are many blinkers in this pattern. One is used as a *position marker* for keeping the memory state 0 or 1, while others are used for turning a signal. Two small circles near the center of the pattern show possible positions of the position marker. If the marker is at the left (right, respectively) position, we regard that the RLEM is in the state 0 (1).

First, consider the case where the state is 0 and an input signal is given to the port a as in this figure. The signal makes a U-turn at the U-turn gadget U_1 since the state is 0. Then it goes to the gadget U_2, and again makes a U-turn passing through Q. Note that U_2 is used to reversibly merge the path with that of the second case. Finally the signal goes out from the port x.

Second, consider the case where the state is 0 and an input signal is given to the port b. At P the signal shifts the position marker to the right, and makes a right-turn. Thus, the state changes to 1. Then, the signal goes out from the output port x via the point Q. This signal path is merged with that of the first case at Q.

Third, consider the case where the state is 1 and an input signal is given to the port a. In this case, the signal goes out from the output port y via S and R without interacting the position marker.

Fourth, consider the case where the state is 1 and an input signal is given to the port b. The signal goes straight ahead at the point P. Then, it shifts the position marker to the left and makes a right-turn at R. Thus, the state changes to 0. Finally it goes out from y. This signal path is merged with that of the third case at R.

Note that, in an RLEM, an incoming signal interacts with the state of the RLEM, not with other signals. Therefore, there is no need of synchronizing two or more signals as in the case of logic gates. Therefore, it greatly simplifies implementation of RLEMs and connecting them in P_0

RLEM 2-4

The pattern shown in Fig. 29 simulates RLEM 2-4. As in the case of RLEM 2-3, one blinker near the center of the pattern is used as a position marker for keeping the memory state 0 or 1. If the marker is in the right (left, respectively) small circle, we regard that the RLEM is in the state 0 (1).

First, consider the case where the state is 0 and an input signal is given to the port a. The signal makes a U-turn at U_2. Then it goes to U_1, and again makes a U-turn passing through T. Finally the signal goes out from the port x.

Second, consider the case where the state is 1 and an input signal is given to the port b. At R the signal goes straight ahead. Then it passes through the points S and T. Finally it goes out from the port x. This signal path is merged with that of the first case at T.

Third, consider the case where the state is 1 and an input signal is given to the port a. The signal goes straight ahead at Q. Then, at P it shifts the position marker

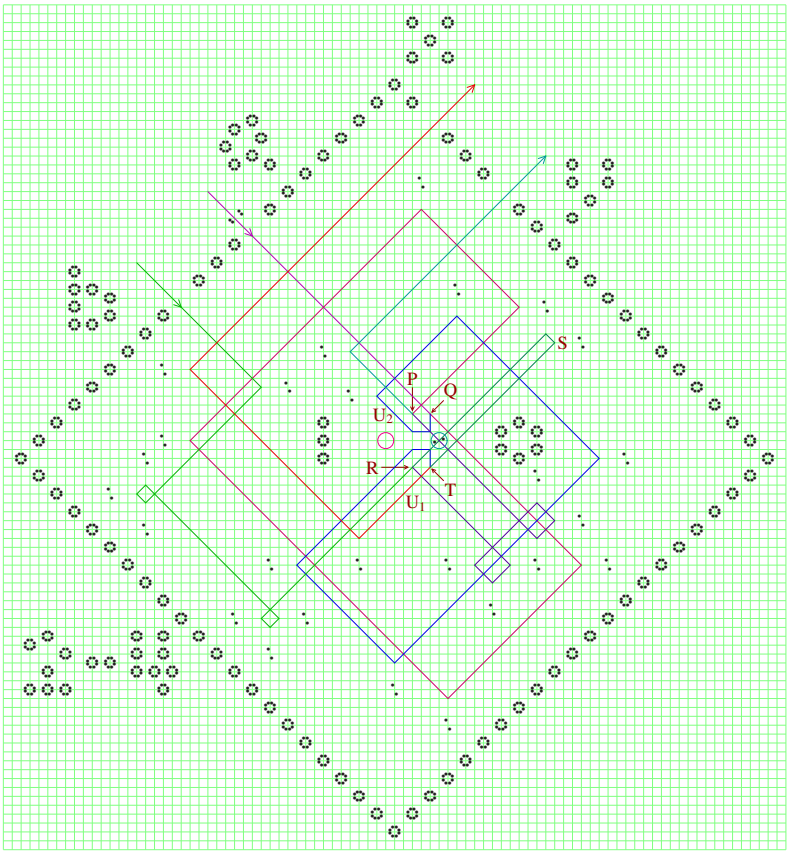


Figure 29: RLEM 2-4 implemented in P_0

to the right, and makes a right-turn. By this, the state changes to 0. Finally it goes out from the port y.

Fourth, consider the case where the state is 0 and an input signal is given to the port b. At R the signal shifts the position marker to the left, and makes a right-turn. By this, state changes to 1. Then, it passes through P, and finally goes out from y. In this case, the signal path is merged with that of the third case at P.

It should be noted that the move function of RLEM 2-4 is the inverse of that of RLEM 2-3. The move function of RLEM 2-3 is as follows.

$$(0,a) \mapsto (0,x),\ (0,b) \mapsto (1,x),\ (1,a) \mapsto (1,y),\ (1,b) \mapsto (0,y)$$

Its inverse is as follows, and is isomorphic to that of RLEM 2-4.

$$(0,x) \mapsto (0,a),\ (1,x) \mapsto (0,b),\ (1,y) \mapsto (1,a),\ (0,y) \mapsto (1,b)$$

In [18], it is shown that in a reversible triangular partitioned CA (ETPCA), a pattern for the "inverse functional module" can be easily obtained from the original pattern by a simple transformation. This is due to the time-symmetry [18] of reversible ETPCAs. A similar property also holds for reversible ESPCAs (but its details are omitted here). By this, the pattern in Fig. 29 is obtained by putting blinkers at the mirror image positions of blinkers of the pattern of in Fig. 28.

RLEM 3-10

Combining the patterns for RLEMs 2-3 and 2-4 to form the circuit shown in Fig. 18, we can easily obtain a pattern that simulates RLEM 3-10 as in Fig. 30.

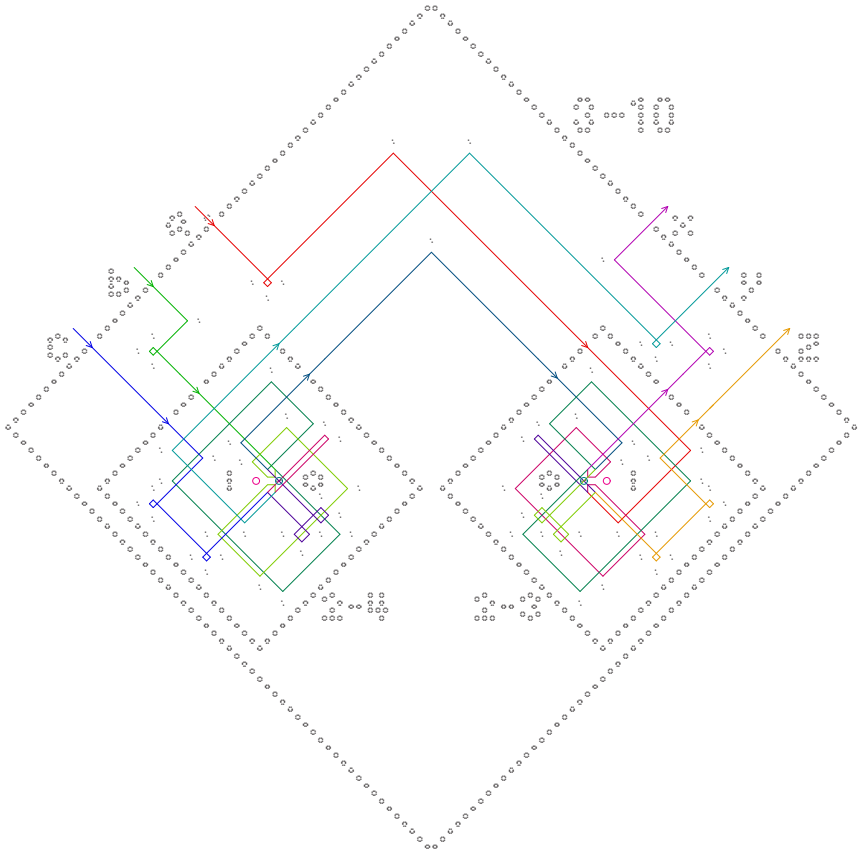


Figure 30: RLEM 3-10 implemented in P_0 composed of RLEMs 2-3 and 2-4

RE

Placing eight copies of the pattern of RLEM 3-10 (Fig. 30) and connecting them to form the circuit shown in Fig. 15, we can obtain a pattern for RE. However, since it is very large (2,000 × 2,000), its figure is omitted here (the pattern can be seen using the file `08_RE_by_3-10.rle` in [16] on Golly).

4.6 Making RTMs in ESPCA P_0

Putting copies of the patterns of RE in P_0 at the positions of REs in Fig. 12, and connecting them appropriately, we have a configuration of P_0 that simulates the RTM T_{parity} of Fig. 12. Any RTM can be implemented in P_0 in this manner.

Fig. 31 shows the configuration for the RTM T_{power} in Example 2.2 simulated on Golly [16]. It takes more than one billion (1,137,250,105) steps to have an answer for the unary input $n = 4$. Therefore, when simulating the computing process of T_{power} on Golly, its speeding-up mode should be used.

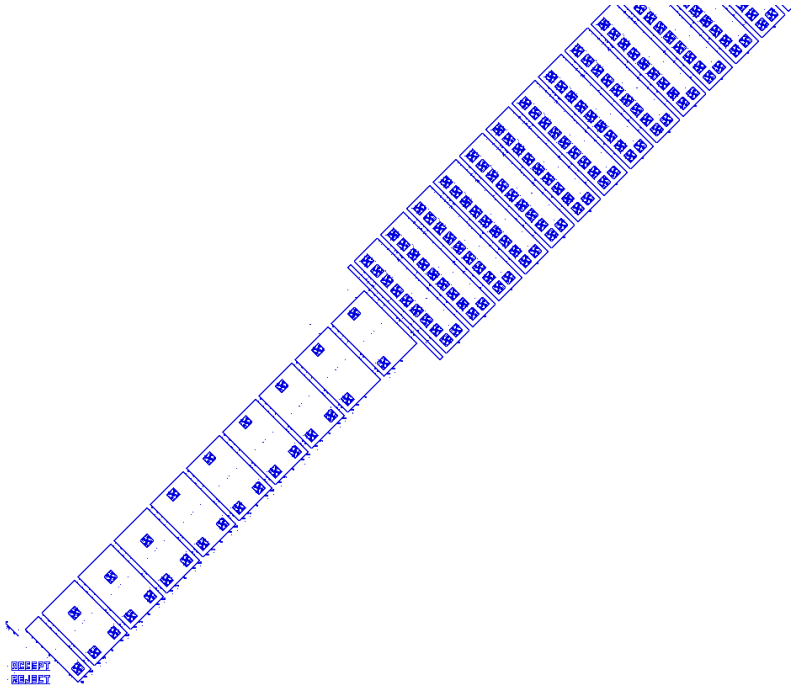


Figure 31: RTM T_{power} implemented in ESPCA P_0 simulated on Golly [16]. Each small square is a pattern that simulates RE

5 Concluding Remarks

We saw that even from a very simple reversible microscopic law, reversible computing machines like RTMs can be realized by a construction method shown in Fig. 1. Here we used a particular reversible ESPCA P_0 as a reversible environment. However, there are many possibilities of other simple reversible CAs, or other frameworks of reversible environments. For example, in [18], a reversible elementary triangular partitioned cellular automaton (ETPCA) is used to construct RTMs. An important fact observed in this paper is that only a few useful reversible phenomena (shown in Sect. 4.4) are sufficient to compose reversible computing machines. We expect such a fact also holds in various reversible environments.

Acknowledgements
The author is grateful to the reviewers for their valuable comments. He also express his thanks to the development team of Golly [27].

References

[1] Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**, 525–532 (1973). https://doi.org/10.1147/rd.176.0525

[2] Cook, M.: Universality in elementary cellular automata. Complex Syst. **15**, 1–40 (2004)

[3] Fredkin, E., Toffoli, T.: Conservative logic. Int. J. Theoret. Phys. **21**, 219–253 (1982). https://doi.org/10.1007/BF01857727

[4] Gurevich, Y.: Reversify any sequential algorithm. Bulletin of EATCS **134**, 42–65 (2021)

[5] Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Trans. Amer. Math. Soc. **117**, 285–306 (1965). https://doi.org/10.2307/1994208

[6] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Prentice Hall (2006)

[7] Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: Proc. 36th FOCS. pp. 66–75. IEEE (1997). https://doi.org/10.1109/SFCS.1997.646094

[8] Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**, 183–191 (1961). https://doi.org/10.1147/rd.53.0183

[9] Lecerf, Y.: Machines de Turing réversibles — récursive insolubilité en $n \in \mathbf{N}$ de l'équation $u = \theta^n u$, où θ est un isomorphisme de codes. Comptes Rendus Hebdomadaires des Séances de L'académie des Sciences **257**, 2597–2600 (1963)

[10] Lee, J., Peper, F., Adachi, S., Morita, K.: An asynchronous cellular automaton implementing 2-state 2-input 2-output reversed-twin reversible elements. In: Proc. ACRI 2008 (eds. H. Umeo, et al.), LNCS 5191. pp. 67–76 (2008). https://doi.org/10.1007/978-3-540-79992-4_9

[11] Margolus, N.: Physics-like model of computation. Physica D **10**, 81–95 (1984). https://doi.org/10.1016/0167-2789(84)90252-5

[12] Morita, K.: Universality of a reversible two-counter machine. Theoret. Comput. Sci. **168**, 303–320 (1996). https://doi.org/10.1016/S0304-3975(96)00081-3

[13] Morita, K.: A simple reversible logic element and cellular automata for reversible computing. In: Proc. MCU 2001 (eds. M. Margenstern, Y. Rogozhin), LNCS 2055. pp. 102–113 (2001). https://doi.org/10.1007/3-540-45132-3_6

[14] Morita, K.: A deterministic two-way multi-head finite automaton can be converted into a reversible one with the same number of heads. In: Proc. RC 2012 (eds. R. Glück, T. Yokoyama), LNCS 7581. pp. 29–43 (2013). https://doi.org/10.1007/978-3-642-36315-3_3

[15] Morita, K.: Theory of Reversible Computing. Springer, Tokyo (2017). https://doi.org/10.1007/978-4-431-56606-9

[16] Morita, K.: Data set for simulating a reversible elementary square partitioned cellular automaton with the ID number 01caef on Golly. Hiroshima University Institutional Repository, http://ir.lib.hiroshima-u.ac.jp/00051974 (2021)

[17] Morita, K.: Computing in a simple reversible and conservative cellular automaton. In: Proc. First Asian Symposium on Cellular Automata Technology (eds. S. Das, G.J. Martinez) AISC 1425, Springer pp. 3–16 (2022). https://doi.org/10.1007/978-981-19-0542-1_1

[18] Morita, K.: Gliders in the Game of Life and in a reversible cellular automaton. In: The Mathematical Artist – A Tribute To John Horton Conway (eds. S. Das, S. Roy, K. Bhattacharjee). Springer (in press)

[19] Morita, K., Harao, M.: Computation universality of one-dimensional reversible (injective) cellular automata. Trans. IEICE **E72**, 758–762 (1989), http://ir.lib.hiroshima-u.ac.jp/00048449

[20] Morita, K., Ogiro, T., Alhazov, A., Tanizawa, T.: Non-degenerate 2-state reversible logic elements with three or more symbols are all universal. J. Multiple-Valued Logic and Soft Computing **18**, 37–54 (2012)

[21] Mukai, Y., Ogiro, T., Morita, K.: Universality problems on reversible logic elements with 1-bit memory. Int. J. Unconventional Computing **10**, 353–373 (2014)

[22] Neary, T., Woods, D.: Four small universal Turing machines. Fundam. Inform. **91**, 123–144 (2009). https://doi.org/10.3233/FI-2009-0036

[23] Rogozhin, Y.: Small universal Turing machines. Theoret. Comput. Sci. **168**, 215–240 (1996). https://doi.org/10.1016/S0304-3975(96)00077-1

[24] Shannon, C.E.: A universal Turing machine with two internal states. In: Automata Studies. pp. 157–165. Princeton University Press, Princeton, NJ (1956)

[25] Toffoli, T.: Computation and construction universality of reversible cellular automata. J. Comput. Syst. Sci. **15**, 213–231 (1977). https://doi.org/10.1016/S0022-0000(77)80007-X

[26] Toffoli, T.: Reversible computing. In: Automata, Languages and Programming (eds. J.W. de Bakker, J. van Leeuwen), LNCS 85. pp. 632–644 (1980). https://doi.org/10.1007/3-540-10003-2_104

[27] Trevorrow, A., Rokicki, T., Hutton, T., et al.: Golly: an open source, cross-platform application for exploring Conway's Game of Life and other cellular automata. http://golly.sourceforge.net/ (2005)

[28] Wolfram, S.: A New Kind of Science. Wolfram Media Inc. (2002)

The Computational Complexity Column

by

Michal Koucký

Computer Science Institute, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
koucky@iuuk.mff.cuni.cz
https://iuuk.mff.cuni.cz/~koucky/

BEATCS no 140

Automata and Formal Languages: Shall we let them go?

Michal Koucký
Computer Science Institute
Charles University, Prague
koucky@iuuk.mff.cuni.cz

Abstract

In this article I give my thoughts on the role of automata and formal languages in our computer science curriculum.

1 Introduction

This article is a reflection of my thoughts on the content of the course on automata and formal languages at Charles university in Prague. Similar courses are a mandatory part of computer science curricula at many other universities around the globe. Four years ago I was asked by our math department to redesign their graduate level course *Automata and computational complexity*. As the name suggests traditionally that course contains a large portion of automata and formal language theory. After discussions with the head of the math department I realized that they do not necessarily care for any particular topic what they care for is that the course covers theoretical foundations of computer science. So I redesigned their course to match my view of foundations of computer science.

As you might expect from the title of this article the role of automata diminished substantially in the new course. I will come back to that new course later. However, this prompted me to take a fresh look at our own computer science course *Automata and grammars*. I believe the time has come to let the automata go and replace the content of the course by what it perhaps always meant to be: theoretical foundations of computer science.

Next I will briefly review the origins of the current course and I will try to put it into historical context of development of computer science over past 90 years. Then I will propose what should be covered in a modern course on theoretical foundations of computer science, and I will go over some hurdles which one might encounter when trying to modify the course.

1.1 Automata and grammars

Our course on automata and formal languages was designed more than 40 years ago and it is largely based on the classical book by Hopcroft and Ullman [8], and its Czech cousin by Chytil [2]. Over the years the course underwent various updates and modifications as the allotted time for the course varied. Today the course mostly follows Sipser's book [10] or the current version of Hopcroft-Ullman [6]. However, the core focus of the course remains the same: automata, grammars and languages recognized by those models.

The course used to be accompanied by a course on recursion theory and later also by a course on computational complexity. The two latter courses moved to graduate level courses during the past 20 years and were substantially overhauled.[1] The course *Automata and grammars* remains mostly unchanged with its focus on automata, grammars and classification of problems according to the Chomsky hierarchy.

My view is that the course was originally designed to reflect then-current knowledge of theoretical foundations of computer science and complexity theory. Let us briefly review the historical context of its origins and development in theory of computing over the years.

1.2 Brief history of modern theory of computing

Here, I am going to present my personal take on history of theoretical computer science. It might not be perfectly accurate but it should be approximately correct. With few exceptions I will ignore the names of many great computer scientists who contributed to this development so I will focus only on the main ideas. A thorough historical account of development of computability is given in the book by Soare [11]. Development of complexity theory is covered in the article by Fortnow and Homer [4].

The development of modern computer science was instigated by logicians. Their program from the turn of the 20th century summarized by Hilbert asked whether mathematical truth can be established by mechanical means. Those questions led to development of models for mechanical procedures: In 1931-34, Gödel proposed definition of recursive functions and Church proposed his λ-calculus, then in 1936 Turing defined what we call *Turing machines* [12]. All those models were quickly established to be equivalent. Arguably, a Turing machine is the right model to capture computation, and it allowed for the development of theory of computation as we know it today. Considerations about what problems are algorithmically solvable lead to development of *computability theory* (*recursion theory*).

[1] In line with European-wide changes to university systems, after the year 2000 our originally five year program was divided into a three year bachelor program and a two year master program.

The recursion theory is concerned with what can be computed by an algorithm and what cannot be computed by an algorithm. A prototypical uncomputable problem is the *Halting problem*.

Soon after development of the concept of computability many people realized that not all algorithmically solvable problems are born equal: Some problems are harder to solve than others, they are more *complex*, their computation might require more steps to complete. This aspect was famously referred to by Gödel in *Gödel's lost letter*.

A simple tool to gauge the complexity of a computational problem is provided by a finite automaton. Finite automata were proposed in 1940's. Motivated by parsing human languages and programming languages, over the next three decades finite automata flourished into a rich theory for formal language classification: from finite automata, to automata with multiple heads, marking automata, push-down automata, etc. This development is captured till large extent by the book of Hopcroft and Ullman [8].

One of the central concepts in this area, the *Chomsky hierarchy*, was formulated in the late 50's [1, 3]. Chomsky hierarchy provides a tool to classify computational problems into easy to solve: *regular languages*, moderate: *context-free languages*, harder: *context-sensitive* and hardest: *recursively-enumerable*. This is a crude classification of their *computational complexity*.

This is the development which is covered by the typical courses on automata and formal languages.

The 1960's saw the origins of a different approach to classification of problems into easy and hard to solve: Hartmanis and Stearns [7] defined space and time complexity, and proved basic hierarchy theorems. This was in the context of nascent *computational complexity theory*. This theory took a central stage with the introduction of NP-completeness in the 1970's. During that decade, the notions of *efficient algorithms*, *complexity measures* and *complexity classes* took a firm hold in theoretical computer science. In its generality those concepts are the focus of *structural complexity theory*. For concrete algorithmic problems this is the focus of *algorithm and data structure design*.

The new point of view stimulated rapid development of new algorithmic techniques and data structures. The complexity approach also laid foundations for modern cryptography. The introduction of public key cryptography in the 1970's in connection with complexity theory led to modern day theoretical cryptography. The 1980's saw development of circuit complexity (studied in the Soviet Union already in 60's and 70's) The late 1980's and early 90's gave us interactive proof-systems, zero-knowledge proofs and the PCP Theorem. This stimulated the development of non-approximability and approximation algorithms in the 1990's and after 2000.

The class P (polynomial time) was established as the equivalent of efficient computation already in the 1970's. Despite many of its desirable properties (e.g.

closeness under poly-time reductions) not everyone was happy with that definition. The late 1990's saw first incursions into sub-linear time algorithms which morfed into the area of property-testing. Concurrently, massive amounts of data that needed to be processed led to the notion of streaming algorithms which blossomed in the first two decades of the 21st century. Streaming algorithms are on some level similar to finite automata: they perform one pass over the data and they allow limited but non-constant memory. However, they go hand in hand with relaxing the requirement for correctness as they allow the answer to be only approximately correct and typically they are randomized.

Late in the first decade of the 21st century new area emerged: fine-grained complexity. The fine-grained complexity pushes the realm of *efficiently computable* closer to usual algorithm design. It establishes very efficient reductions between various concrete problems of interest. It also links the difficulty of algorithm design for NP-complete problems such as Satisfiability with the difficulty of improving algorithms for ordinary problems in P such as All-Pairs Shortest Paths.

All those areas including automata theory are active to this day although their focus has shifted in new directions, and the mainstream of theoretical computer science has changed since the 1970's. The main take-away message from the past 90 years of development of theory of computation is the quest to capture what is and what is not efficiently solvable. The notion of *efficiency* is not static. It progressed from being computable (recursion theory), to being in P (complexity theory), to being linear or quadratic (streaming algorithms and fine-grained complexity).

Arguably the course on automata and formal languages should reflect the progress of our understanding.

2 Foundations of computer science - new syllabus

Here I will present my take on what a redesigned course should focus on. On a high level the focus should stay the same as before, the course should introduce the following ideas: There are problems that can be solved by a computer, and there are problems that cannot be solved by a computer (*computability*). Some problems that can be solved by a computer are easier to solve than others (*complexity*). Those two points should be the primary focus. Here is the syllabus of my *Automata and computational complexity* course I designed for the math department [9]:

1. Computational models: computer, RAM, Turing machine, Boolean circuit.

2. Undecidable problems, Halting problem, reductions.

3. Time complexity, class P.

4. Class NP, NP-hardness, NP-completeness, Cook-Levin Theorem.
5. Space complexity, class PSpace, PSpace-complete problem QBF, Polynomial Hierarchy.
6. Class Log, *s-t*-Connectivity, Savitch's Theorem.
7. Finite automata, regular languages.
8. Hierarchy Theorems, fine-grained complexity.

To start, any argument about computability or complexity needs a rigorous definition of an algorithm, so we need models of computation: *Turing machines*, *RAM*, *their equivalence*. Once we prove that the *Halting Problem* is uncomputable it is useful to extend the result to other problems via *reductions*. Then *time complexity* should come in the picture together with the *class* P representing efficiently computable problems. Afterwards we can move to the *class* NP defined as a class of problems for which we can efficiently verify solutions but we may not know how to find their solutions efficiently. NP-*hardness* via *polynomial time reductions* is a natural next step. This sets the stage for NP-*completeness* of *Satisfiability (*SAT*)* and the *Cook-Levin Theorem*.

Space is another resource which one cares about so we should move on to space complexity: *the class* PSpace with the *complete problem Quantified Boolean Formulas (*QBF*)* as a natural generalization of SAT. By restricting the number of quantifier alternations in QBF formulas we get the Σ_k-SAT, and the *levels of the Polynomial Hierarchy* as the classes of problems reducible to the Σ_k-SAT by polynomial time reductions.

PSpace contains the whole Polynomial Hierarchy and especially NP, so it contains problems that we do not know how to solve efficiently. So we should turn our focus to small space algorithms, the *class* Log of problems solvable in logarithmic space. Arguing about problems in Log requires *log-space reductions* which can be *composed*. A complete problem for Log is the *s-t*-Connectivity on undirected graphs (without giving Reingold's algorithm). One can venture into the *non-deterministic log-space (*NLog*)* as the class of problems log-space reducible to *s-t*-Connectivity on *directed* graphs. Once there it makes sense to show *Savitch's Theorem* that *s-t*-Connectivity can be solved in space $O(\log^2 n)$.

At this point we can go even further with restricting space: we get the class of problems solvable in *constant space*. Constant space on Turing machines is equivalent to *no-space* as we can push the content of the tapes into the state of the Turing machine. Problems decidable by a Turing machine with no-space can be decided by a no-space Turing machine which moves its input head only in one direction: *finite automaton*. Now we reached the well known class of *regular*

languages and our journey downwards stops here. We can show that the language 0^n1^n is *not regular* (*without* using the Pumping Lemma.)

Since we defined all the complexity classes we can compare them to each other by means of *Time Hierarchy* and *Space Hierarchy Theorems*. An excursion into *fine-grained complexity* is a natural next step: a connection between the hardness of improving algorithms for NP-complete problems and problems in P is one of the most enlightening discoveries of the past few decades. My favorite is the *reduction from* SAT *to the Orthogonal Vector Problem*. It is easy and it gets the point across.

Optionally, one can throw in *Boolean circuits* as the model of non-uniform algorithms. This is a model which many students are already familiar with. They know Boolean circuits because they model real hardware, and because of the special case: neural networks.

Nondeterminism. One can completely avoid talking about non-deterministic computation per-se. Personally I am not sure whether we should teach *all* students about the abstract construct which is the non-deterministic computation. If I need to present this concept to my students in my graduate level classes I like to start with randomized computation which is more natural and realistic. Students are usually familiar with randomized computation because they know some randomized algorithms for particular problems. For beginners I prefer the presentation of NP purely using the efficient verification paradigm (see e.g. the textbook by [5]). Other non-deterministic classes (Polynomial Hierarchy, NLOG) can be presented as the classes of problems efficiently reducible to their respective complete problems.

Comparison with Automata and formal languages. Arguably, the high-level structure of current courses on automata and formal languages is the same as that of the proposed syllabus. The traditional course focuses on *recognition of languages* by various models of computation with limits on their computational resources. With respect to the new syllabus the only difference is the choice of restrictions on those resources, the choice of the models, and going top-down instead of bottom-up.

The focus on the current computational complexity will align the course with other course such as on algorithm and data structure design. Those courses revolve around design of efficient algorithms and data structures with respect to time and space complexity where we are concerned with the asymptotic behaviour of the two measures. This is at odds with the Chomsky hierarchy, the classification of problems according to what type of automata recognizes them (finite automata/push-down automata/Turing machines).

3 Adopting a new syllabus

One of the technical hurdles to adopting a new syllabus is the availability of an appropriate textbook. The best current textbook to cover the new syllabus could still be the book by Sipser [10] if one skips Chapters 1 and 2. However, the book in its current form is not ideal as there are many topics and exercises in the later chapters that are concerned with a bit artificial problems on automata from earlier chapters.

Another hurdle is the momentum of the education system. For better or worse, university environment is a rather conservative place with respect to modification of its curricula. There are competing interests of various parties, and different courses are intertwined. Some of the concepts covered by the classical automata and formal languages courses are relied upon in particular branches of computer science.

Natural language processing historically relied upon grammars although most of the current system rely on deep-neural networks. Similarly, theory of programming languages relies on grammars although compilers and interpreters for many current languages do not use them for parsing directly. The *notion* of finite automaton is useful for software engineering to model systems which can be in restricted number of distinct states. The same is true for description of cryptographic primitives and network protocols. This has an impact on the area of software and system verification. Niche applications of regular expressions are in some text editors for searching and in system programming for rule specification. However, none of those latter applications relies on the ability of finite automata to recognize precisely regular languages which is the primary focus of the classical courses on automata and formal languages.

So there are many areas which rely to some degree on the notions covered by the course but perhaps they do not care so much about the closure of regular languages under concatenation, union, intersection, etc. So they could perhaps be satisfied with much more modest coverage of the topics.

Acknowledgements

I love finite automata and the beautiful theory surrounding them. I was introduced to it by Michal Chytil in the very course I am discussing in this article. My master thesis done under the guidance of late Vašek Koubek deals with the hierarchy of marking one-way multi-head finite automata. I also love recursion theory. This was sparked by passionate and thoroughly enjoyable lectures of Antonín Kučera. I owe him for a discussion on the history of computability and a pointer to the book by Soare [11]. Many years ago, Antonín Kučera in his intellectual honesty dissuaded

me from pursuing recursion theory for my doctoral studies and suggested to focus my attention elsewhere. I know I wasn't the only one receiving that advice from him. I leave the reader with a question: *How well do we serve our students if we teach them a theory that they cannot develop substantially further?*

References

[1] Noam Chomsky. Systems of syntactic analysis. *J. Symb. Log.*, 18(3):242–256, 1953.

[2] Michal Chytil. *Automaty a gramatiky*. SNTL, 1984.

[3] Noam Chomsky and Marcel-Paul Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier, 1963.

[4] Lance Fortnow and Steven Homer. A short history of computational complexity. *Bull. EATCS*, 80:95–133, 2003.

[5] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

[6] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition, 2nd Edition*. Addison-Wesley, 2003.

[7] Juris Hartmanis and Richard Edwin Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965.

[8] John E. Hopcroft and Jeffrey D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley series in computer science and information processing. Addison-Wesley, 1969.

[9] Michal Koucký. NMMB415: Automata and computational complexity. `https://iuuk.mff.cuni.cz/~koucky/vyuka/AVS-ZS2021/index.html`, 2021.

[10] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[11] Robert I. Soare. *Turing Computability - Theory and Applications*. Theory and Applications of Computability. Springer, 2016.

[12] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, s2-42(1):230–265, 1936.

The Algorithmics Column

BY

Thomas Erlebach

Department of Computer Science
Durham University
Upper Mountjoy Campus, Stockton Road, Durham, DH1 3LE, UK
thomas.erlebach@durham.ac.uk

BEATCS no 140

What if we tried Less Power?

Lessons from studying the power of choices in hashing-based data structures

Stefan Walzer

Abstract

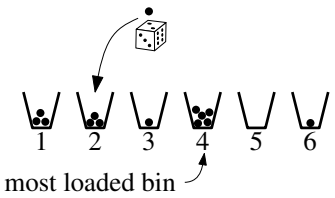
The celebrated *power of two choices* paradigm underlies *cuckoo hash tables* as follows: If you have n balls and $m = (2 + \varepsilon)n$ bins and throw each ball into a bin at random, then likely some bin will receive $\Omega(\frac{\log n}{\log \log n})$ balls. If, however, you can choose between *two* random bins for each ball, you can likely arrange for a *private bin* for each ball.

In the first part of this column, we review some related space-efficient data structures on a high level. We'll find that the additional power afforded by *more than* 2 *choices* is often outweighed by the additional costs they bring. In the second part, we present a data structure where choices play a role at coarser than per-ball granularity. In some sense, we rely on the *power of* $1 + \varepsilon$ *choices* per ball.

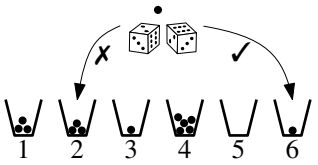
This column is a "best-of" of my dissertation and related work reviewed from a fresh perspective. I've tried to make it a pleasant read conveying intuition while being unencumbered by technical details. So allow me to be your guide through the garden of my interests, present and past. Our winding path will circle a recent construction called Bumped Ribbon Retrieval *[24], with which our tour will conclude.*

Part 1: Data Structures using the Power of Two Choices

The classical demonstration of the *power of two choices* goes as follows [5, 55]. Assume you have n balls, $m = \Theta(n)$ bins and you distribute the balls independently and uniformly at random into the bins.



Then the most loaded bin will contain $\Theta(\frac{\log n}{\log \log n})$ balls with high probability (whp)[1]. In contrast, assume you generate **two options** for each of the balls and place the balls sequentially, putting each ball into the **bin with the least load** among its two options (breaking ties arbitrarily).

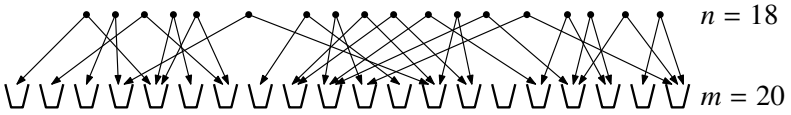


Now the **maximum load** is only $\Theta(\log \log n)$ whp, which is **exponentially less**! The described setting is known as **online load balancing**. Bins might correspond to servers and balls to jobs that have to be assigned to servers on creation. What if we generate further options for each ball? For $d \geq 2$ choices per ball, the maximum load becomes $\frac{\ln \ln n}{\ln d} + \Theta(1)$ whp, i.e. d only affects a constant factor. In other words, there is a massive difference between one and two choices and just a small difference between 2 and $d \geq 3$ choices. Hence the name power of *two* choices.

Since its discovery, the power of two choices paradigm has been influential in data structure design, which is the focus of this paper.

1.1 The Dictionary Problem & Cuckoo Hashing

Now consider **offline load balancing** where we generate the two random bins for all balls in advance and think carefully about all choices at the same time.



Which option should each ball take to avoid collisions?

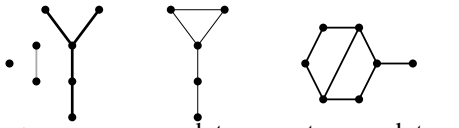
It is helpful to use a different visualisation where each bin is a vertex and each ball an edge connecting the two bins it may be placed into:

(1)

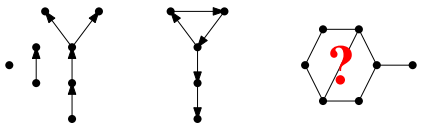
If we ignore the possibility of duplicate edges, then we get a graph with m vertices and n uniformly random edges.This is known as an *Erdős-Renyi random graph*.

[1]Defined as probability $1 - o(1)$.

In their ancient and seminal paper "on the evolution of random graphs" [28] Erdős and Renyi show that if the edge density satisfies $\frac{n}{m} < \frac{1}{2} - \varepsilon$ then whp all connected components of the graph are small **trees** (#edges = #vertices − 1) or pseudotrees (#edges = #vertices). For $\frac{n}{m} > \frac{1}{2} + \varepsilon$, on the other hand, there is whp a "**giant component**" (comprising $\Theta(n)$ vertices) that has more edges than vertices. Redrawing our example we find:



The task of placing all balls without collision corresponds to the task of assigning a **direction** to each of the edges in the graph such that every vertex has indegree at most 1. For tree and pseudotree components this is easy: For trees, we pick an arbitrary root vertex and direct each edge away from the root. For a pseudotree, which contains a single cycle, we direct the cycle in some consistent way (say "clockwise") and every other edge away from the cycle. For the remaining component(s) there is no solution by the pigeon hole principle.



The important observation here is: If we have n balls and $m = (2 + \varepsilon)n$ bins (for constant $\varepsilon > 0$) then **only trees and pseudotrees** arise whp and we can place all balls without a single collision.

Classic Cuckoo Hash Table. This observation gives us a cuckoo hash table [58], a simple data structure that stores n elements, which we call *keys*[2], using one[3] array of m memory cells and two hash functions that assign two uniformly random[4] cells to each key.

Say we wish to store the set {☆, △, □, ◌}.[5] We consider the keys' hashes and find a **collision-free placement** that puts each key into one of its two assigned cells. (If no placement exists, we restart the construction with fresh hash functions.) To

[2] In general elements could be key-value pairs, but values play no role in the following.

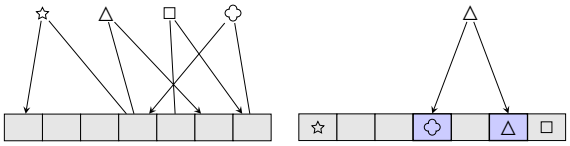
[3] Most implementations use two arrays for reasons that need not concern us here.

[4] See Digression 1.

[5] We'll use comically undersized examples throughout this text that hopefully still get the point across. Practically relevant instances would typically have thousands or millions of keys.

Digression 1: Simple Uniform Hashing Assumption (SUHA).
We assume that hash functions assign hash values to the keys independently. This *simple uniform hashing assumption* is **unrealistic** as $\tilde{\Omega}(n)$ bits of entropy would be needed for independence while popular practical hash functions like *MurmurHash* [1] or *xxhash* [12] use seeds of only $\tilde{O}(1)$ bits.
There are many standard ways of addressing this missmatch. We can try to work with weaker notions like ***k*-independence**, where any set of k keys have independent hash values but any $k+1$ hash values may be correlated [69]. A good overview on how this can help is given in [63] and highly practical 2-independent families are described in [64]. A cryptographer might instead offer some insight into how **pseudorandomness** can be indistinguishable from randomness. [3]
A well-subscribed lazy approach, that we also adopt here, is to simply use the SUHA as a **modelling assumption** and point to its excellent track record of capturing how popular hash functions behave in practice.

answer a query – say we wish to know if △ is in the set – we evaluate the hash functions on △ and search both cells for the requested key.



situation during construction situation during query

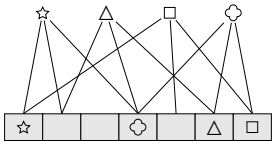
(2)

Query times are $O(1)$ in the **worst-case** (not just in expectation). A down-side is that the **load factor** is $c = \frac{n}{m} < \frac{1}{2}$, meaning twice as much memory is required compared to naively storing keys consecutively. That's less than ideal. But what if we tried more power?

Illustration from *What If?* by Randall Munroe

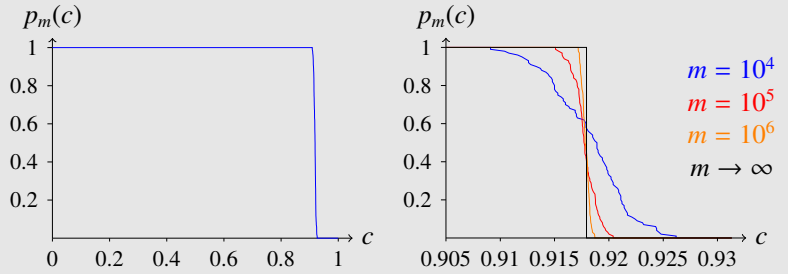
1.1.1 Higher Power: Generalisations of Cuckoo Hashing

A natural generalisation is to **assign more cells to each key** [30]. For $k > 2$ cells the graph with m vertices and n edges from (1) becomes a *hyper*graph with m vertices and n *hyper*edges of size k. This would make for a messy picture so we stick with bipartite illustrations like in (2).



(3)

Digression 2: Sharp Load Thresholds.
Consider the probability p that all n keys can be placed into a table of size m when each key is assigned $k = 3$ cells at random. In the picture on the left, we plot (experimental approximations of) $p = p_m(c)$ for varying load factor $c = \frac{n}{m} \in [0, 1]$ and fixed table size $m = 10^4$.



As expected, p decreases as c increases. What is *not* obvious is that the transition from $p_m(c) \approx 1$ to $p_m(c) \approx 0$ happens within a tiny interval. On the right, we zoom in and also plot the function for $m = 10^5$ and $m = 10^6$. This shows that the **transition becomes steeper for larger m** and starts to resemble a **step function**. In fact, there is a **sharp load threshold** $c_3^* \approx 0.9179$ such that for $c < c_3^*$ we have $\lim_{m\to\infty} p_m(c) = 1$ and for $c > c_3^*$ we have $\lim_{m\to\infty} p_m(c) = 0$.

Using $k > 2$ allows for the assumption $\frac{n}{m} < \frac{1}{2} - \varepsilon$ on the load factor to be relaxed. For $k = 3$ for instance, we find an increased *load threshold* of $c_3^* \approx 0.92$ up to which all keys can be placed whp. See Digression 2 for some background on the phenomenon of thresholds.

Load thresholds c_k^* are known for any k. They can be characterised implicitly as solutions to certain equations, but for our purposes, tabulated values will do. We include $c_1^* = 0$ to emphasise that avoiding collisions with just **one hash function requires $m = \Omega(n^2)$** due to the **birthday paradox**, which gives a load factor of $\frac{n}{n^2} \to 0$.

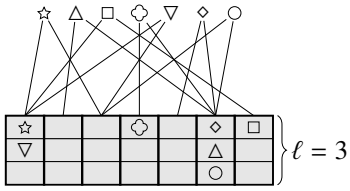
k	1	2	3	4	5	6	7
c_k^*	0	0.5	0.91794	0.97677	0.99244	0.99738	0.99906

Values as determined in [58, 17, 35, 32].

The values are of order $c_k^* = 1 - e^{-\Theta(k)}$ (this can be derived form [32]) so we can achieve load factors arbitrarily close to 1 by choosing k large enough. However, any practitioner will be quick to point out that a query operation that has to check k random memory locations is likely to incur ***k* cache misses**, so increasing k is

not particularly enticing.

The far more popular generalisation sticks with $k = 2$ hash functions, but each hash value identifies a *bucket* of ℓ memory cells [8, 29, 31]. Each key may then be placed into any cell in any one of its **two buckets**.



Thresholds $c^*_{2,\ell}$ for the load factor achievable this way are also known:

ℓ	1	2	3	4	5	6
$c^*_{2,\ell}$	0.5	0.89701	0.95915	0.98037	0.98955	0.99407

Values as determined in [58, 8, 29, 17, 31].

Unsurprisingly, both avenues for generalisation can be combined such that a key can be placed within any of k buckets of size ℓ each. The corresponding thresholds $c^*_{k,\ell}$ are all known [31, 49, 46].

Which generalisation is better? Should we use **more hash functions or bigger buckets?** *On the one hand*, if you consider the **number of memory cells a query touches** in the worst case, then using more hash functions seems superior to using bigger buckets. For instance, with 3 hash functions we get a threshold of $c^*_3 \approx 0.918$ and have to scan three memory cells per query. When using 2 hash functions and buckets of size 2 we get a lower threshold of $c^*_{2,2} \approx 0.897$ and have to scan four memory cells per query. It turns out the four correlated cells in the two buckets do not constitute as powerful a choice as three independent locations.

On the other hand, the reduced number of **hash function evaluations and cache misses** strongly favours using larger buckets rather than additional hash functions, even if the number of memory cells associated with a key has to be higher to achieve the same load factor.

From what I have seen, practitioners aiming for high load factors seem to be pretty happy with either $k = 2$ or a middle ground using $k = 3$ hash functions and some bucket size $\ell \in \{3, \ldots, 8\}$, see e.g. [50, 71]. A related compromise assigns to each key k cells *within the same memory page* and 1 additional cell in a *backup page* [18] (see also [62]). It turns out most keys can then be stored on their primary page.

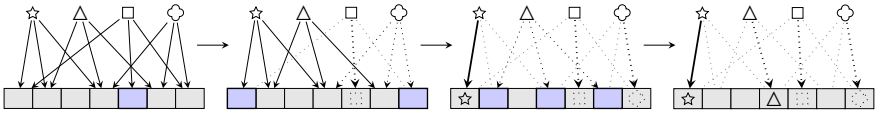
The message so far can be summarised as follows: Cuckoo hash tables are powered by the first two independent choices. Adding the third choice can help but competes for attention with other measures for increasing the load factor. **Two choices is all you really need.**

1.1.2 Power Dynamics: Cuckoo Table Construction and Insertion

The thresholds mentioned so far only relate to whether or not a rule-conforming placement of all keys in the hash table *exists*. But how can such a placement be *found and maintained*? For simplicity, we focus on cuckoo hashing with k hash functions and buckets of size $\ell = 1$.

Table construction. Constructing a cuckoo hash table is about finding a matching in a bipartite graph such as (3). Out of the box, the maximum matching algorithm by Hopcroft and Carp [41] has a *worst case* running time of $O(n^{3/2})$, but a specialised algorithm with *expected* running time $O(n)$ for $k \geq 2$ is known [44].

A conceptually interesting greedy algorithm is ***peeling***. It **identifies cells in the table that are an option for only one (remaining) key**. In the following illustration on the left, at first only □ can be placed in this way into cell 5, then ☆ and ◌ can be placed, and, finally, △ ends up with three cells to itself and can be placed in any one of them.



The peeling process

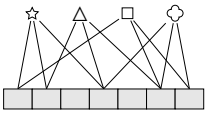
To better assess the utility of peeling, consider the load thresholds $c_k^{\triangle}$ up to which peeling manages to place all keys in a cuckoo hash table with k hash functions:

k	3	4	5	6	7
$c_k^{\triangle}$	0.81847	0.77228	0.70178	0.63708	0.58178

Values as determined in [56]. Variants in [14, 60, 43] and [51, Chapter 18].

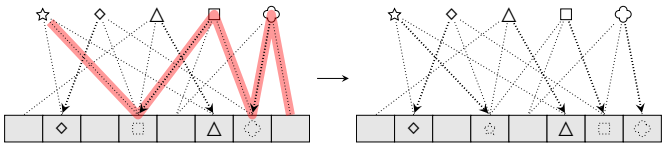
There are two things to notice here. *Less relevant* is that the value for $k = 2$ is missing. To see why, recall the graph with m vertices and n edges from (1). Peeling can handle trees but gets stuck if there is at least one cycle. Unfortunately, the probability for a cycle to exist is bounded away from zero as soon

as $\frac{n}{m}$ is bounded away from 0. So for peeling to work *whp* we need $k \geq 3$. *More relevant* is that $c_k^* < c_k^\triangle$, meaning there are load factors where a **placement exists whp but peeling fails to find one whp**. Even worse, the gap $c_k^* - c_k^\triangle$ between the thresholds increases with k, even converging to 1 for $k \to \infty$. This **disqualifies peeling as a general-purpose construction** algorithm for cuckoo hash tables. However, peeling will make a comeback in more specific settings (stay tuned!).



A placement exists, but peeling is stuck immediately.

Insertions. A construction algorithm suffices for a *static* key set, but for a dynamic data structure, we need to maintain a placement as keys are inserted and deleted over time. A deletion is trivial: Simply locate the key, by checking all associated cells, and remove it from there. An insertion on the other hand amounts to modifying an existing matching to incorporate a new key. This suggests that we look for an **augmenting path**.



In the picture, ☆ is the newly added key and moves into the cell previously used by □, which moves into the cell previously used by ◇, which moves into an empty cell.

There are two well-known strategies for finding such an augmenting path, both proposed in [30]. ***Breadth first search (BFS)*** insertion computes the **shortest augmenting path** in the natural BFS way. ***Random walk*** **(RW)** insertion goes ahead and places the unplaced key into one of its cells at random and *evicts* the key that was there before (if any). The evicted key is then also placed randomly and so on until an evicted key is placed into an empty cell. Strategies more clever than this have also been considered, some of which store some auxiliary data in the cells [45]. How good are these algorithms? It seems that, up to constant factors between them, they are equally **excellent in practice** in the sense that their running time does not depend on n, i.e. is $O(1)$.

Conjecture 1 (Echoing sentiments from [34, 68, 45, 36, 30])**.** *Consider a cuckoo hash table with $k \geq 2$ hash functions, buckets of size $\ell \geq 1$, and a load factor $\frac{n}{m} \leq c_{k,\ell} - \varepsilon$ for some $\varepsilon > 0$. Assume the keys are inserted sequentially using BFS insertion or RW insertion. Conditioned on the high probability event that a placement of all keys exists, the expected time to perform each insertion is $O(f(\varepsilon))$ for some function f that does not depend on n.*

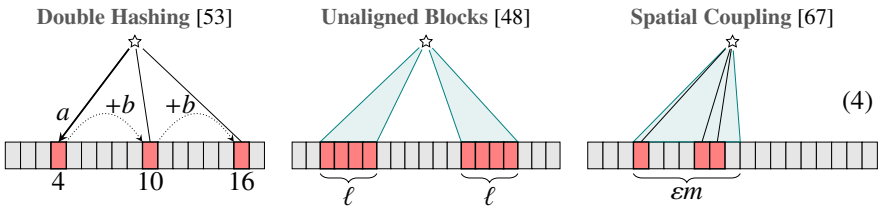
I know of **no proof**, neither for RW nor for BFS and for no pair of k and ℓ, except for the classical case of $k = 2$ and $\ell = 1$, where there is no choice regarding which key to evict from a given bucket (there is just one) and no choice regarding where to relocate an evicted key to (there is just one alternative). Partial proofs exist for the case with $\ell = 1$ for

- BFS, for $k > 8$ and under a stronger restriction on the load factor [30],
- RW, for large k and under a stronger restriction on the load factor [34],
- RW, for load factors below the *peeling* threshold $c_k^{\triangle}$ [68],
- RW, but only guaranteeing running times of $O(\text{polylog}\, n)$ [36, 33].

More progress towards proving the conjecture would be exciting. Maybe techniques from statistical physics, which have been a powerful tool for determining thresholds in the static case [49, 47, 46] can help with the dynamic case as well.

1.1.3 Creatively Wielding the Power

The number k of hash functions and the size ℓ of buckets are not the only degrees of freedom in the design space. Here is a short list of **further variants** that were considered and the reasons why.



(4)

Double Hashing. Mitzenmacher and Thaler [53] proposed that the buckets $b_1, \ldots, b_k$ assigned to a key are not chosen independently. Instead, only b_1 and an offset d are chosen at random, and $b_2, \ldots, b_n$ are defined as $b_i := b_1 + (i-1) \cdot d$, modulo the number of buckets. This **reduces** the amount of **entropy** in a key's hash values from $k \log m$ to $2 \log m$ with no apparent downsides. In particular the thresholds $c^*_{k,\ell}$ remain the same [47, 52].

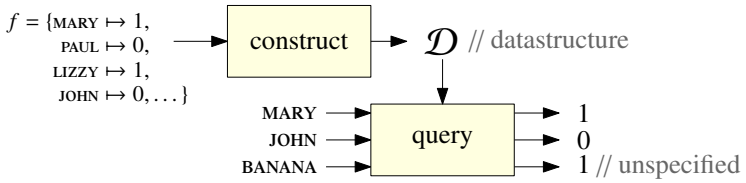
Unaligned Blocks. Lehman and Panigrahy proposed to use buckets that do not form a partition of the set of cells. Rather, *any contiguous sequence* of ℓ cells can occur as a bucket, regardless of the offset. This scheme yields **higher thresholds** than $c^*_{k,\ell}$ without affecting the access pattern of queries [48, 65].

Spatial Coupling. Walzer proposed that the k buckets assigned to a key are chosen within the same interval of εm buckets for some $\varepsilon > 0$. Assuming ε is small enough and the load factor is less than $c^*_{k,\ell}$, not only does a placement exist whp, but **peeling works whp**. [67]

I cannot help but wonder which other surprising effects can be achieved by further creative cuckoo hashing variants.

1.2 The Retrieval Problem & Random Matrices

In the ***retrieval problem***[6] we are given a set S and a function $f : S \to \{0,1\}^r$ for some $r \in \mathbb{N}$. Let us assume $r = 1$. The task is to construct a data structure that returns $f(x)$ when queried for $x \in S$. What is unusual is that a query for some $y \notin S$ may return an **arbitrary result**. In an instructive example due to Pagh and Dietzfelbinger [19], S is a set of names and f tells us if a name $x \in S$ is female ($f(x) = 1$) or male ($f(x) = 0$).



When f reflects typical English names, the data structure should return 0 for JOHN $\in$ domain(f) and 1 for MARY $\in$ domain(f). When queried for BANANA $\notin$ domain(f) both 0 and 1 are allowed, we don't care.

The **trivial solution** for this problem stores the **set of pairs**

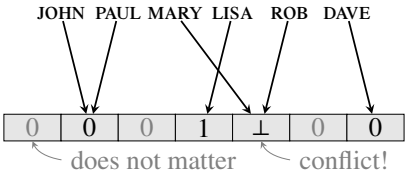
$$f = \{(\text{JOHN}, 0), (\text{MARY}, 1), (\text{LIZZY}, 1), (\text{PAUL}, 0), \dots\}$$

using a dictionary. This requires storing at least $n = |S|$ strings. However, we will soon see that the most space-efficient solutions require little more than n *bits*. Note that n bits are necessary if we make no further assumptions on the input.[7]

As a warm-up, here is a **compact solution** that needs $O(n)$ bits. We use an array of $m = O(n)$ cells that can store values from $\{0, 1, \perp\}$. We associate each $x \in S$ with a random cell $h(x) \in [m]$ and set a cell to 0 or 1 whenever a consistent value exists and $\perp$ in case of conflicts:

[6]The first mention of the problem under the name "retrieval" that I could find is in [16] in 2006. Bloomier filters [10] in 2004 are a clear spiritual predecessor and related to approximate membership. More background is given in [19].

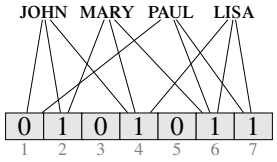
[7]If there are regularities such as the majority of names in domain(f) being male or most female names ending in a vowel, then compression maybe possible, see [42, 6, 38].



The keys involved in a conflict make up a constant fraction of all keys in expectation. For these, we can build a fallback data structure recursively. Some readers may have fun working out that we get **constant expected access time** and a total space consumption of roughly $e/2$ array cells per key for $m = n/2$.[8] This amounts to **$e \approx 2.72$ bits per key** when using the naive encoding $\{0 \mapsto 00, 1 \mapsto 11, \perp \mapsto 01\}$.An improved version of this idea is known as **filtered retrieval** [57].

1.2.1 The Power to be Independent: Retrieval via Random Linear Systems

To get closer to **succint retrieval** we consider strange cousins of cuckoo hash tables, cf. [7, 39]. While not employing the power of two choices in the traditional sense, their setup and analysis are closely related. Hash functions assign to each key **several cells** in an array of size $m \geq n$ that is populated with bits in such a way that taking the **XOR of all bits** associated with $x \in S$ yields $f(x)$.



(5)

A query for JOHN would, for instance, compute $f(\text{JOHN}) = 0 \oplus 1 \oplus 1 = 0$, where $\oplus$ denotes XOR. To construct the data structure we had to choose values $z_1, z_2, \ldots, z_7 \in \{0, 1\}$ to put into the array to simultaneously satisfy the equations

$$\begin{aligned} z_1 \oplus z_2 \oplus z_4 &= 0 \quad (\text{JOHN}) \\ z_2 \oplus z_4 \oplus z_6 &= 1 \quad (\text{MARY}) \\ z_1 \oplus z_6 \oplus z_7 &= 0 \quad (\text{PAUL}) \\ z_4 \oplus z_6 \oplus z_7 &= 1 \quad (\text{LISA}) \end{aligned}$$

Since $\oplus$ is addition in the **two element field** $\mathbb{F}_2 = \{0, 1\}$ these equations are **linear equations** over $\mathbb{F}_2$. So fasten your seatbelt for a bit of linear algebra, but don't

[8] Hint: Assume that $n/2$ names are male and $n/2$ names are female (you can later check that this is the worst case). Then use that for a fixed name x, the number of names of the opposite gender that share the hash value of x has distribution $\text{Bin}(\frac{n}{2}, \frac{1}{m})$, which converges to the Poisson distribution $\text{Po}(\frac{n}{2m}) = \text{Po}(1)$ that satisfies $\Pr[\text{Po}(1) > 0] = 1 - 1/e$.

worry, it's not so bad. We can write the above equations in **matrix form** as

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \begin{matrix} (\text{JOHN}) \\ (\text{MARY}) \\ (\text{PAUL}) \\ (\text{LISA}) \end{matrix}$$

When does such a system have a solution $\vec{z} \in \{0,1\}^m$? Well, the *columns* of the $n \times m$ matrix A should better span all of $\{0,1\}^n$, so that the right hand side vector $(f(x))_{x \in S} \in \{0,1\}^n$ can surely be attained as a linear combination of these columns. In other words, the column rank of A should be n. Since column rank and row rank are the same thing, the n *rows* of A have to be **linearly independent**.

What are our goals here? Remember that n is part of the input and we are free to choose two things: The number m of columns and the way in which keys are associated with row vectors via hash functions. Ideally we want that ***m* is small** so that $z \in \{0,1\}^m$ is **cheap to store** and we want row vectors to have **small Hamming weight** so queries are **cheap to evaluate**. Both of these goals are intuitively in tension with the independence requirement.

An encouraging fact is that even *square* matrices (i.e. $m = n$) where every entry is chosen by a biased coin with probability $p = \frac{\log n}{n}$ (i.e. rows have expected Hamming weight $\log n$) are regular with constant probability [13]. This is, however, not the most natural setup for our purposes.

1.2.2 New Data Structure, Same Thresholds: How Cuckoo Hashing Connects to Retrieval

A more natural setup already depicted in (5) associates k random cells with every key, which produces a matrix with ***k* ones per row** in random positions.

There is a **threshold** $c_k^\square$ for the ratio $c = \frac{n}{m}$ such that A has rank n whp when $c < c_k^\square - \varepsilon$ and A has rank less than n whp when $c > c_k^\square + \varepsilon$. Remarkably, it **coincides with the threshold for cuckoo hashing** with k hash functions, i.e. $c_k^\square = c_k^*$. Only the direction $c_k^\square \leq c_k^*$ is easy to show here. If A has rank n, then some selection of n columns induces a regular submatrix A' (highlighted below).

$$\begin{pmatrix} 1 & \mathbf{1} & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \mathbf{1} & 0 & 1 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & \mathbf{1} \end{pmatrix}$$

The determinant of A' is a non-zero number in $\mathbb{F}_2$, hence $\det(A') = 1$. By the Leibniz formula for determinants we have $1 = \det(A') = \sum_{\pi \in S_n} \prod_{i \in [n]} a'_{i,\pi(i)}$ where $(a'_{i,j})_{i,j \in [n]}$ are the entries of A' and S_n is the symmetric group with n elements. At least one $\pi \in S_n$ must yield a non-zero contribution to $\det(A')$, and all entries $a'_{i,\pi(i)}$ for $i \in [n]$ must then be 1. These entries are shown in bold in the matrix above and correspond to an injective placement of all keys in a cuckoo hashing setting. Bluntly:

$$c < c_k^{\square} \Leftrightarrow \text{“retrieval works whp”} \Rightarrow \text{“cuckoo hashing works whp”} \Leftrightarrow c < c_k^*$$

hence $c_k^{\square} \leq c_k^*$. The converse statement $c_k^{\square} \geq c_k^*$ requires a lot more work to prove [26, 59, 17, 11].

1.2.3 Longer Blocks and Bitparallel Queries

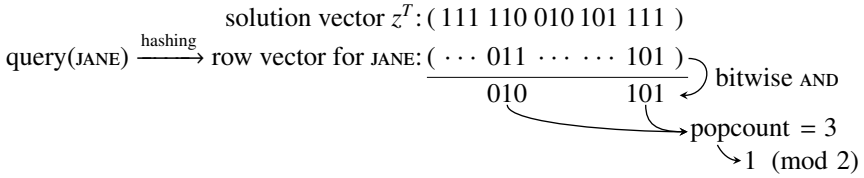
To get **thresholds closer to 1** there are similar options as in cuckoo hashing. **Increasing k** works but requires queries to combine data from more independent cells, resulting in **more cache misses**. Meh. We can also adopt the idea of using buckets of size ℓ. Naively connecting each key to each cell in two blocks of size ℓ *does not work* however. We merely obtain copies of identical columns that do not contribute to the matrix rank, as shown below on the left (from now on we use a dot “$\cdot$” to indicate implicit zeroes that do not explicitly occur in any representation and we omit the right hand side of equations). *Instead* we should associate each key with a **random non-empty *subset* of cells of each of its blocks** as shown on the right.

$$\begin{pmatrix} \cdots & 111 & \cdots & 111 & \cdots \\ 111 & \cdots & \cdots & \cdots & 111 \\ \cdots & 111 & \cdots & \cdots & 111 \\ 111 & \cdots & 111 & \cdots & \cdots \\ \cdots & 111 & 111 & \cdots & \cdots \\ 111 & \cdots & \cdots & 111 & \cdots \end{pmatrix} \qquad \begin{pmatrix} \cdots & 010 & \cdots & 110 & \cdots \\ 100 & \cdots & \cdots & \cdots & 110 \\ \cdots & 111 & \cdots & \cdots & 010 \\ 010 & \cdots & 110 & \cdots & \cdots \\ \cdots & 111 & 001 & \cdots & \cdots \\ 101 & \cdots & \cdots & 001 & \cdots \end{pmatrix} \tag{6}$$

bad idea: each key (row) is associated with all cells within two blocks of size $\ell = 3$.

okay idea: each key (row) is associated with *some* cells within two blocks of size $\ell = 3$.

Precise thresholds are not known and not identical to the cuckoo hashing thresholds $c^*_{2,\ell}$. What we do know is that for suitable $\ell = \Theta(\log n)$ and $m = n + \Theta(\log n)$ we obtain a matrix with rank n whp [20]. This yields a **succinct** retrieval data structure with an almost optimal space requirement of $n + O(\log n)$ bits. Now consider a query. Given the solution vector $z \in \{0,1\}^m$ and the row vector associated with a key $x \in S$ we have to compute their **scalar product** to obtain $f(x)$.



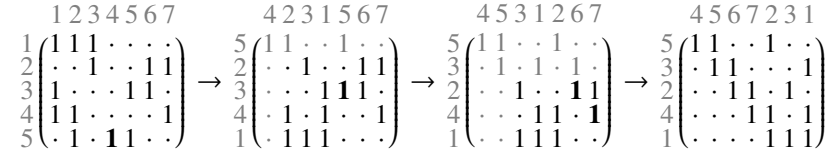
An element-wise product of two vectors in $\mathbb{F}_2^\ell$ is a **bitwise AND** and the sum of the entries of a vector in $\mathbb{F}_2^\ell$ is a **population count modulo 2**. On a **word RAM** with word size $w = \Omega(\ell)$, we can perform all we need for a **query in $O(1)$** steps.

With constant time queries and almost optimal space, this seems like an ideal data structure until you realise that there is no fast way to construct it.

1.2.4 To Gauss or not to Gauss: Constructing Retrieval Data Structures

A problem with all this is that computing the solution vector $z \in \{0,1\}^m$ **requires solving a system** of linear equations. Spending $O(n^3)$ time on Gaussian elimination seems prohibitively expensive. Let us consider some alternatives.

Linear Time Solvers using Peeling. We can use a setup where peeling works whp, traditionally with $k = 3$ cells per key and a load factor below $c_3^\triangle \approx 0.82$ [7, 56]. Peeling means we look for a variable only appearing in one equation. We postpone initialising this variable to the end and ignore the equation until then. The remaining system may again have a variable appearing in only one of the remaining equations and so on. A different way of saying that a linear system is peelable is that its matrix can be transformed into echelon form by **row and column *permutations* alone**, with no need for row *additions*.



The row and column of the highlighted 1 is swapped to the front to create an echelon form.

After finding the peeling order of equations and variables in **linear time**, we can find a solution with back substitution in linear time. The setup with peeling thresholds close to 1 from Section 1.1.3 reconciles this approach with close to optimal space efficiency both in theory [67] and in practice [40].

Quadratic Time Solver using Wiedemann's Algorithm. If $\mathbb{F}$ is a finite field, $A \in \mathbb{F}^{n\times n}$ a regular matrix with ψ non-zero entries and $b \in \mathbb{F}^n$ then a solution z to $A \cdot$

$z = b$ can be computed in $O(n\psi)$ field operations using **Wiedemann's algorithm** [70]. For sparse matrices with $\psi = O(n)$ this gives a **quadratic time** solver that can be generalised to non-square matrices A. Exploiting this for retrieval has been tried [4, 66], but it did not perform convincingly in experiments.

Bringing out the Big Gauss Rifles. A line of papers starting with Genuzio et al [37] cooked up ideas that made biting the bullet of performing Gaussian elimination seem like an almost appetising prospect.

The most important realisation is that we can **partition the input set** into many *shards* of expected size C using a hash function and construct a data structure for each of these shards. This reduces the running time from $O(n^3)$ to $O(\frac{n}{C} \cdot C^3) = O(nC^2)$. The price is an additional level of indirection during queries and a few bits of metadata for each of the $\frac{n}{C}$ shards.[9] This can be combined with **bit-level parallelism**, a **structured Gaussian elimination** that tries to keep the matrix as sparse as possible for as long as possible, and the **method of four Russians** [2], which eliminates entries from $O(\log n)$ columns at the same time. This yields fairly efficient solvers that are trivial to **parallelise**.

Part 2: The Power of *less than two* Choices

2.1 Ribbon: Choice is overrated

After a talk on the retrieval data structure using two blocks per matrix row (see (6) on the right) a listener asked a question.

Seth Pettie to my coauthor Martin Dietzfelbinger at Dagstuhl-Seminar 19051.

Supposedly he had this in mind:

$$\begin{pmatrix} \cdot\cdot\cdot\cdot\cdot\cdot 1111 \cdot\cdot\cdot \\ \cdot\cdot\cdot\cdot 0011 \cdot\cdot\cdot\cdot\cdot \\ 1011 \cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot \\ \cdot\cdot\cdot\cdot\cdot\cdot 1011 \cdot\cdot\cdot \\ \cdot\cdot\cdot\cdot 1001 \cdot\cdot\cdot\cdot\cdot \\ \cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot 1001 \\ \cdot\cdot\cdot\cdot\cdot 0101 \cdot\cdot\cdot\cdot \\ \cdot\cdot\cdot\cdot\cdot\cdot 1100 \cdot\cdot\cdot \\ \cdot\cdot 1100 \cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot \\ \cdot\cdot\cdot\cdot\cdot 1011 \cdot\cdot\cdot\cdot \end{pmatrix} \tag{7}$$

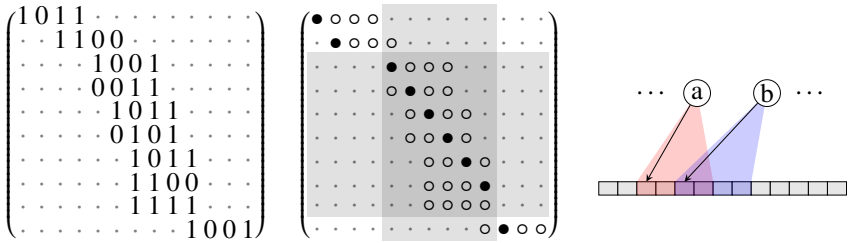
In the suggested scheme there is a block size w and each key is associated with a *starting position* $i \in [m - w + 1]$ and a coefficient vector $c \in \{0, 1\}^w$ that together

[9] In theory, a construction with two levels of indirection and shards of size $C = \sqrt{\log n}$ can lead to a succinct data structure with linear construction time [61]. However, astronomical input sizes are required for the asymptotic behaviour to kick in.

lead to a row in A with zeroes everywhere except for **one randomly placed block of w random bits**.

This blasphemous single-block-per-key suggestion throws the power of choices paradigm completely out the window. Could this work?[10] More precisely, for which values of m, n and w is such a matrix likely to have independent rows?

Simplification: Bounded Linear Probing. First, let us **sort the rows** by starting position. Second, let us simplify the problem by ignoring the patterns of zeroes and ones and just **consider the placement problem** where we wish to select one position within the block of each row without selecting the same column twice. Our question is now simply: In a linear probing hash table where each key hashes to a starting position, **can all keys be placed within $w - 1$ cells of their starting positions?**



rows sorted by starting position

simplified task: select one position in the block of each row using distinct columns

bounded linear probing

A greedy algorithm suffices to decide this question: Go through the keys in ascending order of starting position and place each key as far left as possible. The filled dots above indicate where the keys are placed. If we fail to place a key this way, as we do in the example, then this is witnessed by a range of N positions such that at least $N + 1$ keys have their block completely contained within the range (shaded grey with $N = 6$).

Connection to queuing theory. There is an equivalent way to describe the greedy placement algorithm just discussed (cf. [21]). We maintain a **FIFO queue** Q of keys and go through the table cells from left to right. When handling cell i we add the keys with starting position i to the back of Q and then place the first key in Q (if any) into position i and remove it from Q. This procedure places every key within its block if and only if the size of Q never exceeds w.[11] We therefore analyse the size q_i of Q after step i. If x_i is the number of keys with starting position i

[10] Spoiler alert: Yes, and it kicked off an entire line of papers for us [21, 25, 24].

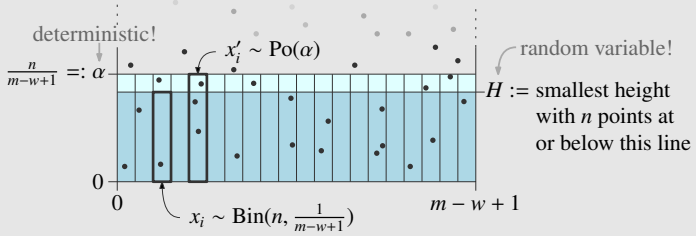
[11] Can you see why?

then

$$q_i = \max(0, q_{i-1} + x_i - 1) \text{ for } i \geq 1, \text{ and } q_0 = 0. \tag{8}$$

Each x_i has Binomial distribution $\mathrm{Bin}(n, \frac{1}{m-w+1})$ because each of the n keys has an independent chance of $\frac{1}{m-w+1}$ to have i as its starting position. Since we know $\sum_i x_i = n$, there is a slight but annoying negative correlation between the x_i. A common approximation in such a situation **replaces** the family $(x_i)_{i\in[m-w+1]}$ of **Binomial** random variables with an *independent* family $(x'_i)_{i\in[m-w+1]}$ of **Poisson** distributed random variables with the same expectation $\mathbb{E}[x'_i] = \alpha := \frac{n}{m-w+1}$. See Digression 3 for an explanation.

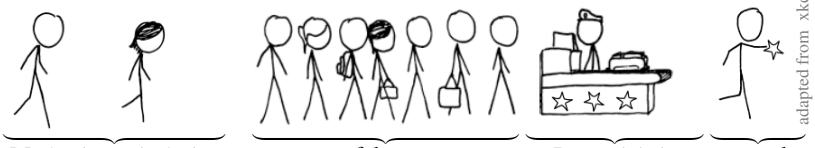
Digression 3: Poissonisation and random coupling (see also [54, Chapter 5.4]).



The picture shows a strip of width $m - w + 1$ that is open to the top and contains points randomly with a *rate* of 1 point per unit of area (known as a *Poisson point process*). Within the strip we consider $m - w + 1$ disjoint sub-strips of height α as shown. The number of points x'_i within the i-th strip has distribution $\mathrm{Po}(\alpha)$. Let now H be the smallest vertical position such that exactly n points fall at or below H. Note that H is a random variable. Because each of the points below H is within any of the vertical strips with the same probability, the number x_i of points below H in the i-th strip has distribution $\mathrm{Bin}(n, \frac{1}{n-w+1})$.

Technically, we have constructed a **coupling** between the two families $(x_i)_{i\in[m-w+1]}$ and $(x'_i)_{i\in[m-w+1]}$ of random variables, i.e. we have embedded them in the same probability space. The outcomes of $(x_i)_{i\in[m-w+1]}$ and $(x'_i)_{i\in[m-w+1]}$ are now tightly linked. If $H \geq \alpha$ then we have $x_i \geq x'_i$ for all i and if $H \leq \alpha$ then we have $x_i \leq x'_i$ for all i, and both happen with probability roughly $1/2$. By changing α very slightly we can ensure that one of the two cases occurs whp. Relying on some monotonicity property of our surrounding problem such as "additional keys can only reduce the probability that all keys can be placed" allows us to **transfer a result that holds for the Poisson model to the Binomial setting**.

With this change, the values $q_0, q_1, q_2, \ldots$ from (8) are the states of a so-called **M/D/1** queue, which is a **Markov chain** that can be illustrated like this:



Markovian arrivals, i.e. people arrive randomly with a *rate* of α

state of the process: number q of people waiting

Deterministic server handling **1** person per time unit

served people leave

If $\alpha < 1$, then the customers arrive, on average, slower than they can be served and there is a **stable distribution** for the number q^* of waiting customers in the long run. **Queuing theory** literature promises an average queue length of $\mathbb{E}[q^*] = O(1/(1-\alpha))$ [15] and the tail bound $\Pr[q^* \geq w] = \mathrm{e}^{-\Theta(w/(1-\alpha))}$ [27, Prop 3.4]. This means we have to pick $w = \Omega(\log(n)/(1-\alpha))$ to ensure that the size of Q never exceeds w within $O(n)$ steps whp, which means all keys are validly placed in bounded linear probing.

So is this any good? We have essentially reinvented a **linear probing** hash table with the twist that the **maximum probe length** is guaranteed to not exceed $w = O(\log(n)/(1-\alpha))$ when the load factor is $\frac{n}{m} \approx \frac{n}{m-w+1} = \alpha$. This is strongly related to *Robin Hood hashing* [9] and not too exciting at first.

The arguments just given can be strengthened to show that the matrix from (7) with block length w has independent rows whp.[12] Moreover, a corresponding linear system can be solved with $O(n/(1-\alpha))$ row operations in expectation using **Gaussian elimination** because after sorting the rows by starting position the matrix is **already close to being in echelon form** (as already seen above). The number of row operations per column is linked to the average length of Q. Assuming we can handle $O(\log n)$ bits at a time using bit parallelism, the scalar product to be carried out by a query takes $O(1/(1-\alpha))$ operations. A **query** is very cache efficient because it reads w **contiguous bits from memory**.

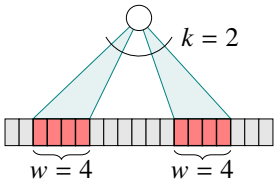
Overall we get a retrieval data structure with a load factor close to 1 that needs to access one contiguous sequence of bits from memory per query, with no bullet biting concerning expensive linear system solvers. My coauthor Martin and I were quite pleased with this solution [21], which has since been dubbed *ribbon retrieval* [24], because it improved upon the state of the art in 2019. Then we got an email from a database engineer who made it even better. [22]

[12] The corresponding variant of the M/D/1 queue involves customers that randomly pay attention only half the time when they are in the queue. At every step, the left-most customer that pays attention is served, if any. When the queue is long, the speed at which customers are served is hardly affected. The additional time a customer spends in the queue is $O(\log(n))$ whp.

2.2 Bumped Ribbon: The Power of 1 + ε Choices

The following ideas were intended for retrieval, but here I continue in the simpler hash table setting, where they also apply.

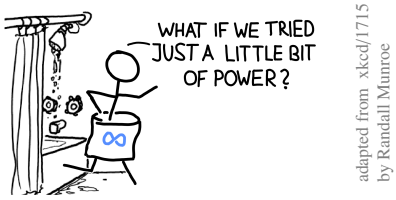
More than one, less than two. Let's take a step back. Consider a cuckoo hashing setting where each key is associated with k random starting positions $s_1, \ldots, s_k \in [m-w+1]$ and may be placed in cells with an index in $\bigcup_{i\in[k]}\{s_i, \ldots, s_i + w - 1\}$, i.e. within one of **$k$ randomly placed blocks of size w**.



For $k = 1$ we obtain the ribbon scheme discussed in Section 2.1 and for $k \geq 2$ a scheme by Lehman and Panigrahy briefly mentioned in Section 1.1.3. Let us now ask: What is the **smallest $w = w(k)$** such that all keys can be placed whp when the load factor is, say, $\alpha = 0.99$? We find:

k	1	2	3	4	5	6	7	...
$w(k)$	$\Theta(\log n)$	3	2	2	1	1	1	...

The point is: There is a **qualitative difference** between having just one choice of a block (if you even want to call it "choice") and the power of two or more choices. Could there be an interesting **middle ground between 1 and 2**?



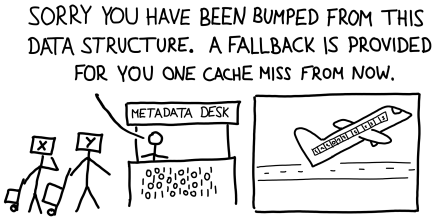
How I imagine Peter Dillinger came up with a great idea.

Could there be a power of $k = 1.5$ choices? What would this even mean? Well, a key could have 2 choices with probability 50% and only 1 choice with probability 50%. One might argue that this is 1.5 choices per key on average. I would nitpick and say that choices tend to aggregate multiplicatively (selecting among 2 choices and then among 3 choices gives 6 choices overall) so taking the geometric mean

and speaking of $\sqrt{2} \approx 1.41$ choices per key is more natural.[13] Either way, the problem is that placing the $\approx 50\%$ of keys that end up with 1 choice (and ignoring the others) still requires $w = \Omega(\log n)$ with no improvement over $k = 1$.

But here is a different kind of compromise. We partition the key set into groups of expected size b and offer ***c* choices for handling a group of keys** as a whole. This (arguably) corresponds to $k = c^{1/b}$ independent choices per key. Such a k might well be strictly between 1 and 2. Dillinger et al. [24] dubbed a concrete setup of this kind *bumped ribbon*.

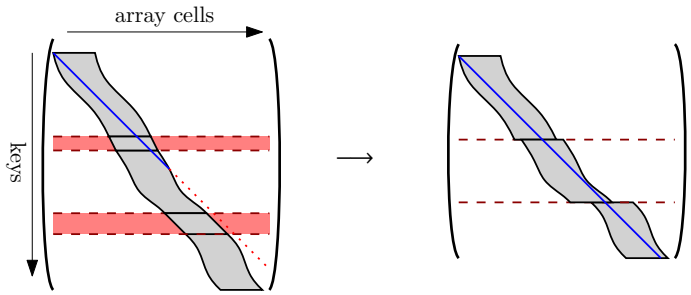
Bumped Ribbon. Like before, each key is associated with one block of w consecutive positions. Keys are partitioned into groups with consecutive starting positions. For each group, there are two choices. The keys are either stored normally or the entire group is ***bumped***, meaning the keys are moved to a recursively constructed **fallback data structure**.[14]



To get some intuition, consider the following matrix-like visualisations were rows correspond to the keys sorted by starting position and columns to array cells. Grey shading in position (i, j) indicates that the ith key may go into array slot j (imagine we have zoomed out so the border of the grey area looks like a continuous curve). We'd like for the **matrix diagonal** to run through the shaded region because then the ith key can go into the ith slot so (i) every key would be placed and (ii) every slot would be put to use. We can ensure (i) and *mostly* ensure (ii) with two measures. We start with a **slightly overloaded** data structure (a load factor $\alpha > 1$) and then **bump groups of keys in strategic positions**.

[13] This is the right view when considering the *average* amount of *information* that has to be recorded per key to encode the choices. This information in bits is the $\log_2$ of the number of choices.

[14] To bring this more in line with the previous setting, we can consider the fallback data structure to be a special segment in the primary data structure rather than separate and ensure that that bumped keys are not bumped *again* by the fallback data structure. In [23] such a variant is called Bu^{1}RR, but introducing it here would be a distraction from our main point.



an overloaded data structure where not all keys could be placed

after bumping some keys, the remaining keys perfectly fill all cells

Details on the exact proposal are given in Digression 4. For any $\varepsilon > 0$, bumped ribbon has $1+\varepsilon$ choices per key in the mean (for each *group* of $O(1/\varepsilon)$ keys, there are 3 choices: bump none, some, or all keys from the group). The block size is $w = \Omega(\log(1/\varepsilon)/\sqrt{\varepsilon})$. The memory overhead (the fraction of wasted space taking into account unused cells, metadata and fallback data structures) is of order $O(\varepsilon)$. In this sense, we are using a **power of $1+\varepsilon$ choices**! Framed in terms of the previous table:

k	1	$\mathbf{1+\varepsilon}$	2	3	4	5	6	7	…
$w(k)$	$\Theta(\log n)$	$\boldsymbol{O(\frac{\log(1/\varepsilon)}{\sqrt{\varepsilon}})}$	3	2	2	1	1	1	…

While we have not evaluated the performance of bumped ribbon as a static *dictionary*, the corresponding static *retrieval* data structure has significantly improved upon the state of the art by marrying three useful properties: First, courtesy

Digression 4: Some details on bumped ribbon.
We are actually bumping keys by starting position, meaning a key is bumped if and only if its starting position is marked as bumped. The starting positions are partitioned into groups of size $O(w^2/\log w)$. For each group, we store 2 bits of metadata that indicate which positions are bumped: (i) none, (ii) the first $\frac{3}{8}w$ positions, or (iii) all positions, where (iii) is a rarely used emergency option. The initial load factor (before bumping) is set to be $\alpha = 1 + \Theta(\frac{\log w}{w})$. The metadata can be set such that:

1. Each non-bumped key is placed within w cells of its starting position.
2. Only a $w^{-\Omega(1)}$ fraction of the cells remains empty.
3. The overall memory overhead is dominated by the metadata and thus $O(\frac{\log w}{w^2})$ bits per key.

of $1 + \varepsilon$ choices, the **block size w need not grow with n**. Second, there is a good chance of answering a query with a **single memory access** since most keys are not bumped. Third, we get **close to linear time construction** since the relevant linear system is already close to echelon form.

~~TL;DR~~ Conclusion

Our journey began with a discussion of cuckoo hash tables that have worst-case access times of $O(1)$ and a load factor of $\alpha = \frac{1}{2} - \varepsilon$. We considered various approaches for increasing the load factor and argued that increasing the number of independent choices that every key has is not the most attractive option due to the increased number of cache misses incurred by each query. While two choices per key is qualitatively different from a single choice (i.e. "no choice"), the step to more than two choices is comparatively underwhelming. What's more, when considering static retrieval data structures closely related to cuckoo hash tables we find that the flexibility afforded by multiple choices can come at the price of expensive construction algorithms.

Wondering if "two" is really the least amount of choice one can have per key motivates *bumped ribbon*. When used as a hash table it is much like linear probing, except that the maximum probe length is guaranteed to never exceed a constant w. Keys can be marked as "bumped", meaning they are placed in a fallback data structure, if their part of the hash table is too crowded. Superficially, there are two choices per key: bumped or not bumped. However, there is more rigidity in two important ways: First, only few keys are bumped so the entropy in any key's choice is much less than one bit. Second, bumping decisions are correlated with large groups of keys being bumped as a whole.

In a sense we have harnessed the power of $1 + \varepsilon$ choices per key. The qualitative power of choices is present, namely the ability to defuse certain worst-case constellations, but the price that flexibility brings is significantly attenuated.

References

[1] Austin Appleby. Murmurhash3, 2012. URL: `https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp`.

[2] V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev. On economical construction of the transitive closure of a directed graph. *Dokl. Akad. Nauk SSSR, 194*, 1970.

[3] Jean-Philippe Aumasson and Daniel J. Bernstein. Siphash: A fast short-input PRF. In *Proc. 13th INDOCRYPT*, pages 489–508, 2012. `doi:10.1007/978-3-642-34931-7_28`.

[4] Martin Aumüller, Martin Dietzfelbinger, and Michael Rink. Experimental variations of a theoretically good retrieval data structure. In *Proc. 17th ESA*, pages 742–751, 2009. doi:10.1007/978-3-642-04128-0_66.

[5] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, 1999. doi:10.1137/S0097539795288490.

[6] Djamal Belazzougui and Rossano Venturini. Compressed static functions with applications. In *Proc. 24th SODA*, pages 229–240. SIAM, 2013. doi:10.1137/1.9781611973105.17.

[7] Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Practical perfect hashing in nearly optimal space. *Inf. Syst.*, 38(1):108–131, 2013. doi:10.1016/j.is.2012.06.002.

[8] Julie Anne Cain, Peter Sanders, and Nicholas C. Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *Proc. 18th SODA*, pages 469–476, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283433.

[9] Pedro Celis. *Robin Hood Hashing*. PhD thesis, University of Waterloo, Canada, 1986.

[10] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The Bloomier filter: An efficient data structure for static support lookup tables. In *Proc. 15th SODA*, pages 30–39. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982797.

[11] Amin Coja-Oghlan, Mihyun Kang, Lena Krieg, and Maurice Rolvien. The k-XORSAT threshold revisited. *CoRR*, abs/2301.09287, 2023. arXiv:2301.09287.

[12] Yann Collet. xxhash - extremely fast hash algorithm, 2020. URL: https://github.com/Cyan4973/xxHash.

[13] Colin Cooper. On the rank of random matrices. *Random Structures & Algorithms*, 16(2):209–232, 2000. doi:10.1002/(SICI)1098-2418(200003)16:2<209::AID-RSA6>3.0.CO;2-1.

[14] Colin Cooper. The cores of random hypergraphs with a given degree sequence. *Random Struct. Algorithms*, 25(4):353–375, 2004. doi:10.1002/rsa.20040.

[15] Robert B. Cooper. *Introduction to queueing theory*. George Washington University, 3rd edition, 1990.

[16] Erik D. Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. De dictionariis dynamicis pauco spatio utentibus (*lat.* on dynamic dictionaries using little space). In *LATIN*, pages 349–361, 2006. doi:10.1007/11682462_34.

[17] Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via XORSAT. In *Proc. 37th ICALP (1)*, pages 213–225, 2010. doi:10.1007/978-3-642-14165-2_19.

[18] Martin Dietzfelbinger, Michael Mitzenmacher, and Michael Rink. Cuckoo hashing with pages. In *Proc. 19th ESA*, pages 615–627, 2011. doi:10.1007/978-3-642-23719-5_52.

[19] Martin Dietzfelbinger and Rasmus Pagh. Succinct data structures for retrieval and approximate membership (extended abstract). In *Proc. 35th ICALP (1)*, pages 385–396, 2008. doi:10.1007/978-3-540-70575-8_32.

[20] Martin Dietzfelbinger and Stefan Walzer. Constant-time retrieval with $O(\log m)$ extra bits. In *Proc. 36th STACS*, pages 24:1–24:16, 2019. doi:10.4230/LIPIcs.STACS.2019.24.

[21] Martin Dietzfelbinger and Stefan Walzer. Efficient Gauss elimination for near-quadratic matrices with one short random block per row, with applications. In *Proc. 27th ESA*, pages 39:1–39:18, 2019. doi:10.4230/LIPIcs.ESA.2019.39.

[22] Peter Dillinger. Ribbon: A practical and near-optimal static bloom alternative for rocksdb, 2020. URL: https://www.youtube.com/watch?v=XfwxUBL8xT8&t=1h16m10s.

[23] Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. Fast succinct retrieval and approximate membership using ribbon. *CoRR*, abs/2109.01892, 2021. arXiv:2109.01892.

[24] Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. Fast succinct retrieval and approximate membership using ribbon. In *20th SEA*, volume 233, pages 4:1–4:20, 2022. doi:10.4230/LIPIcs.SEA.2022.4.

[25] Peter C. Dillinger and Stefan Walzer. Ribbon filter: practically smaller than Bloom and Xor. *CoRR*, 2021. arXiv:2103.02515.

[26] Olivier Dubois and Jacques Mandler. The 3-XORSAT threshold. In *Proc. 43rd FOCS*, pages 769–778, 2002. doi:10.1109/SFCS.2002.1182002.

[27] Regina Egorova, Bert Zwart, and Onno Boxma. Sojourn time tails in the M/D/1 processor sharing queue. *Probability in the Engineering and Informational Sciences*, 20:429–446, 2006. doi:10.1017/S0269964806060268.

[28] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 1960. URL: http://www.renyi.hu/~p_erdos/1961-15.pdf.

[29] Daniel Fernholz and Vijaya Ramachandran. The k-orientability thresholds for $g_{n,p}$. In *Proc. 18th SODA*, pages 459–468, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283432.

[30] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.*, 38(2):229–248, 2005. doi:10.1007/s00224-004-1195-x.

[31] Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The multiple-orientability thresholds for random hypergraphs. *Combinatorics, Probability & Computing*, 25(6):870–908, 2016. doi:10.1017/S0963548315000334.

[32] Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp load thresholds for cuckoo hashing. *Random Struct. Algorithms*, 41(3):306–333, 2012. doi:10.1002/rsa.20426.

[33] Nikolaos Fountoulakis, Konstantinos Panagiotou, and Angelika Steger. On the insertion time of cuckoo hashing. *SIAM J. Comput.*, 42(6):2156–2181, 2013. doi:10.1137/100797503.

[34] Alan M. Frieze and Tony Johansson. On the insertion time of random walk cuckoo hashing. *Random Struct. Algorithms*, 54(4):721–729, 2019. doi:10.1002/rsa.20808.

[35] Alan M. Frieze and Páll Melsted. Maximum matchings in random bipartite graphs and the space utilization of cuckoo hash tables. *Random Struct. Algorithms*, 41(3):334–364, 2012. doi:10.1002/rsa.20427.

[36] Alan M. Frieze, Páll Melsted, and Michael Mitzenmacher. An analysis of random-walk cuckoo hashing. *SIAM J. Comput.*, 40(2):291–308, 2011. doi:10.1137/090770928.

[37] Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast scalable construction of (minimal perfect hash) functions. In *Proc. 15th SEA*, pages 339–352, 2016. doi:10.1007/978-3-319-38851-9_23.

[38] Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast scalable construction of ([compressed] static | minimal perfect hash) functions. *Information and Computation*, 2020. doi:10.1016/j.ic.2020.104517.

[39] Thomas Mueller Graf and Daniel Lemire. Xor filters: Faster and smaller than Bloom and cuckoo filters. *ACM J. Exp. Algorithmics*, 25:1–16, 2020. doi:10.1145/3376122.

[40] Thomas Mueller Graf and Daniel Lemire. Binary fuse filters: Fast and smaller than xor filters. *ACM J. Exp. Algorithmics*, 27:1.5:1–1.5:15, 2022. doi:10.1145/3510449.

[41] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

[42] Jóhannes B. Hreinsson, Morten Krøyer, and Rasmus Pagh. Storing a compressed function with constant time access. In *Proc. 17th ESA*, pages 730–741, 2009. doi:10.1007/978-3-642-04128-0_65.

[43] Svante Janson and Malwina J. Luczak. A simple solution to the k-core problem. *Random Struct. Algorithms*, 30(1-2):50–62, 2007. doi:10.1002/rsa.20147.

[44] Megha Khosla. Balls into bins made faster. In *Proc. 21st ESA*, pages 601–612, 2013. doi:10.1007/978-3-642-40450-4_51.

[45] William Kuszmaul. Brief announcement: Fast concurrent cuckoo kick-out eviction schemes for high-density tables. In *28th SPAA*, pages 363–365. ACM, 2016. doi:10.1145/2935764.2935814.

[46] M. Leconte, M. Lelarge, and L. Massoulié. Convergence of multivariate belief propagation, with applications to cuckoo hashing and load balancing. In *Proc. 24th SODA*, pages 35–46, 2013. URL: http://dl.acm.org/citation.cfm?id=2627817.2627820.

[47] Mathieu Leconte. Double hashing thresholds via local weak convergence. In *Proc. 51st Allerton*, pages 131–137, 2013. doi:10.1109/Allerton.2013.6736515.

[48] Eric Lehman and Rina Panigrahy. 3.5-way cuckoo hashing for the price of 2-and-a-bit. In *Proc. 17th ESA*, pages 671–681, 2009. doi:10.1007/978-3-642-04128-0_60.

[49] Marc Lelarge. A new approach to the orientation of random hypergraphs. In *Proc. 23rd SODA*, pages 251–264. SIAM, 2012. doi:10.1137/1.9781611973099.23.

[50] Tobias Maier, Peter Sanders, and Stefan Walzer. Dynamic space efficient hashing. *Algorithmica*, 81(8):3162–3185, 2019. doi:10.1007/s00453-019-00572-x.

[51] Marc Mezard and Andrea Montanari. *Information, Physics, and Computation*. Oxford University Press, Inc., USA, 2009.

[52] Michael Mitzenmacher, Konstantinos Panagiotou, and Stefan Walzer. Load thresholds for cuckoo hashing with double hashing. In *Proc. 16th SWAT*, pages 29:1–29:9, 2018. doi:10.4230/LIPIcs.SWAT.2018.29.

[53] Michael Mitzenmacher and Justin Thaler. Peeling arguments and double hashing. In *Proc. 50th Allerton*, pages 1118–1125, 2012. doi:10.1109/Allerton.2012.6483344.

[54] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, New York, NY, USA, 2nd edition, 2017.

[55] Michael David Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, Harvard University, 1996. URL: http://www.eecs.harvard.edu/~michaelm/postscripts/mythesis.pdf.

[56] Michael Molloy. Cores in random hypergraphs and Boolean formulas. *Random Struct. Algorithms*, 27(1):124–135, 2005. doi:10.1002/rsa.20061.

[57] Ingo Müller, Peter Sanders, Robert Schulze, and Wei Zhou. Retrieval and perfect hashing using fingerprinting. In *Proc. 14th SEA*, pages 138–149, 2014. doi:10.1007/978-3-319-07959-2_12.

[58] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. doi:10.1016/j.jalgor.2003.12.002.

[59] Boris Pittel and Gregory B. Sorkin. The satisfiability threshold for k-XORSAT. *Combinatorics, Probability & Computing*, 25(2):236–268, 2016. doi:10.1017/S0963548315000097.

[60] Boris Pittel, Joel Spencer, and Nicholas C. Wormald. Sudden emergence of a giant k-core in a random graph. *J. Comb. Theory, Ser. B*, 67(1):111–151, 1996. doi:10.1006/jctb.1996.0036.

[61] Ely Porat. An optimal Bloom filter replacement based on matrix solving. In *Proc. 4th CSR*, pages 263–273, 2009. doi:10.1007/978-3-642-03351-3_25.

[62] Ely Porat and Bar Shalem. A cuckoo hashing variant with improved memory utilization and insertion time. In *Proc. 22nd DCC*, pages 347–356, 2012. doi:10.1109/DCC.2012.41.

[63] Michael Rink. *Thresholds for Matchings in Random Bipartite Graphs with Applications to Hashing-Based Data Structures*. PhD thesis, Technische Universität Ilmenau, Germany, 2015. URL: http://www.db-thueringen.de/servlets/DocumentServlet?id=25985.

[64] Mikkel Thorup. High speed hashing for integers and strings. *CoRR*, 2015. arXiv:1504.06804.

[65] Stefan Walzer. Load thresholds for cuckoo hashing with overlapping blocks. In *Proc. 45th ICALP*, pages 102:1–102:10, 2018. doi:10.4230/LIPIcs.ICALP.2018.102.

[66] Stefan Walzer. *Random Hypergraphs for Hashing-Based Data Structures*. PhD thesis, Technische Universität Ilmenau, 2020. URL: https://www.db-thueringen.de/receive/dbt_mods_00047127.

[67] Stefan Walzer. Peeling close to the orientability threshold: Spatial coupling in hashing-based data structures. In *Proc. 32nd SODA*, pages 2194–2211. SIAM, 2021. doi:10.1137/1.9781611976465.131.

[68] Stefan Walzer. Insertion time of random walk cuckoo hashing below the peeling threshold. In *30th ESA*, volume 244 of *LIPIcs*, pages 87:1–87:11, 2022. doi:10.4230/LIPIcs.ESA.2022.87.

[69] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981. doi:10.1016/0022-0000(81)90033-7.

[70] Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62, 1986. doi:10.1109/TIT.1986.1057137.

[71] Jens Zentgraf and Sven Rahmann. Fast gapped k-mer counting with subdivided multi-way bucketed cuckoo hash tables. In *22nd WABI*, volume 242 of *LIPIcs*, pages 12:1–12:20, 2022. doi:10.4230/LIPIcs.WABI.2022.12.

The Formal Language Theory Column

BY

Giovanni Pighizzini

Dipartimento di Informatica
Università degli Studi di Milano
20133 Milano, Italy
pighizzini@di.unimi.it

BEATCS no 140

Formal Languages via Theories over Strings: An Overview of Some Recent Results

Joel D. Day * Vijay Ganesh † Florin Manea ‡

Abstract

In this note, we overview a series of results that were obtained in [16, 15]. In these papers, we have investigated the properties of formal languages expressible in terms of formulas over quantifier-free theories of word equations, arithmetic over length constraints, and language membership predicates for the classes of regular, visibly pushdown, and deterministic context-free languages. As such, we have considered 20 distinct theories and decidability questions for problems such as emptiness and universality for formal languages over them. In this note, we first present the relative expressive power of the approached theories. Secondly, we discuss the decidability status of several important decision problems, some of them with practical applications in the area of string solving, such as the emptiness and the universality problem. To this end, it is worth noting that the emptiness problem for some theory is equivalent to the satisfiability problem over the corresponding theory. Finally, we discuss the problem of deciding whether a language expressible in one theory is also expressible in another one, and show several undecidability results; these investigations are particularly relevant in the context of normal forms for string constraints, and, as such, they are relevant to both the practical and theoretical side of string solving.

The current note is heavily based on the contents of [16, 15], and the readers are encouraged to check these references for complete details.

1 Introduction

Logical theories based on strings (or words) over a finite alphabet have been an important topic of study for decades, as described, for instance, in the two fundamental handbooks in the area of combinatorics on words [33, 32] as well as

*Loughborough University, United Kingdom, J.Day@lboro.ac.uk
†University of Waterloo, Canada, vijay.ganesh@uwaterloo.ca
‡Universität Göttingen, Germany, florin.manea@cs.informatik.uni-goettingen.de

in the volumes 1 and 3 of the handbook of formal languages [39, 40]. Connections to arithmetic (see, e.g.,Q̃uine [38]) and fundamental questions about free (semi)groups underpinned interest in logics involving concatenation and equality. Combining these two topics leads to word equations: expressions $\alpha = \beta$ where α and β are terms obtained by concatenating variables and concrete words over a finite alphabet. For example, if x and y are variables, and our alphabet is $\Sigma = \{a, b\}$, then $xaby = ybax$ is a word equation. Its solutions are variable-substitutions unifying the two sides: $x \to bb, y \to b$ would be one such a solution for the previous example; other solutions are $x \to b^{n+1}, y \to b^n$, for $n \geq 0$.

The existential theory of a finitely generated free semigroup consists of formulas made up of Boolean combinations of word equations. In fact, the problem of deciding whether such a formula is true is equivalent to determining satisfiability of word equations, since any such formula can be transformed into a single word equation without disrupting satisfiability (see [33, 27]). While it was originally hoped that the problem of deciding if a word equation has a solution could facilitate an undecidability proof for Hilbert's famous Tenth problem, by providing an intermediate step connecting Diophantine equations to the computations of Turing Machines, Makanin showed in 1977 that satisfiability of word equations is algorithmically decidable [35], putting an end to these hopes, but also opening a new line of research. Since then, several improvements to the algorithm proposed by Makanin have been proposed. From a complexity point of view, Plandowski [37] showed that this satisfiability problem can be solved in PSPACE, which has been refined to nondeterministic linear space by Jeż via the Recompression technique [25]. On the other hand, Schulz [41] showed that the problem remains decidable even when the variables are constrained by regular languages, limiting the possible substitutions (see Chapter 12 of [33]). On the other hand, if length constraints (requiring that some pairs of variables are substituted for words of the same length) are permitted, then the (un)decidability of the problem is a long-standing open problem.

Word equations, and logics involving strings more generally, have remained a topic of interest within the Theoretical Computer Science community, in particular in the areas of Combinatorics on Words and Formal Languages, where they play a fundamental role. More recently, word equations became interesting to the Formal Methods community as well. This interest can be attributed to increasing popularity and influence of software tools called string-solvers, which seek to algorithmically solve constraint problems involving strings [6, 22]. In this setting, a string constraint formalizes a property of an unknown string (or string variable), and the string solvers try to determine whether strings (over a potentially infinite domain) exist which satisfy logical combinations of string constraints of various types. Word equations, regular language membership, and comparisons between lengths are all among the most prominent building blocks of string constraints

(as described in [6]), and when combined are sufficient to model several others (see, for instance, some examples in [16]). Various other string constraints are discussed in, e.g., [14]. String-solvers are also useful in other areas like Database Theory, particularly for evaluating path queries in graph databases [7] and in connection with Document Spanners [19, 18]. Recently, to overcome some difficulties related to solving string constraints over infinite domains, a finite-model version of the theory of concatenation was considered [20].

Many string-solvers are now available [26, 8, 11, 36, 28, 1, 43, 9, 2] (see also [6, 22] for an overview). However, the underlying task of determining the satisfiability of string constraints remains a challenging problem and a barrier to more effective implementations. Motivated in part by the applications in string-solving, and by the desire to make progress on seemingly very difficult open theoretical problems, some results already exist addressing the computability, complexity, and expressibility of combinations of string constraints. [34, 21, 29, 31, 30] identify restrictions on word equations which result in a decidable satisfiability problem even when length constraints are present. Several further ways of augmenting word equations (i.e., additional predicates or constraints on the variables), are shown to be undecidable in [13, 14, 12, 23].

Nevertheless, despite results such as those mentioned above, little is known about the true expressive power of word equations and of string logics involving word equations in conjunction with other common types of string constraints. A greater understanding in this regard would be of great help in settling open problems (such as for whether satisfiability for word equations with length constraints is decidable), and also with devising string solving strategies: often simply finding a solution to one constraint is not enough and the set of solutions must be considered more generally to account for other constraints which might be present, or to determine that no solution exists. Moreover, a common tactic is to rewrite constraints into some normal form before solving and understanding when and how this can be done also requires knowledge of the relative expressive power of subsets of constraints.

Our work [16] filled some gaps in the understanding of the properties and expressivity of some of the most important combinations of string constraints by considering languages expressible in the sense of [27]. In this regard, our results can be seen as extending [27] to a more general (and more practical) setting, where word equations are combined with language membership constraints and length constraints.

The framework: In [16], a landscape of string-based logics was considered, incorporating various types of atoms inspired by and strongly related to prominent varieties of string-constraints. In particular, we considered logics with different combinations of the following four types of predicates: equality between strings, concatenations of strings, membership of formal languages, and linear

arithmetic over string-lengths. In total, this covered 20 distinct families of logical theories (each family containing a different theory for each possible underlying alphabet Σ). We will overview them in detail in Section 3. Based on [27] (and partly on [10], where relation-definability by logics over strings was studied in a database-theory centred framework), we have analysed these logics from a formal language perspective by looking at the set of values a variable may take while preserving satisfiability of a formula. Specifically, given a formula f from a quantifier-free logical theory $\mathfrak{T}$, we say that the language expressed by a variable x occurring in f is the set of concrete values w such that substituting x for w in f yields a satisfiable formula. In the general case, we can think of the property that the formula f defines via the variable x. However, since we deal with logics in which x is substituted for finite strings, we get a formal language.

In our approach, we were interested both in the expressive power of the logical theories with respect to what languages they can express, and in their computational properties with respect to canonical decision problems within formal languages such as emptiness, universality, equivalence and inclusion.

Getting more into details, the 4 types of fundamental predicates we allowed in these logics cover many of the most prominent types of string constraints, as listed in [6]. While predicates related to equality between strings, concatenations of strings, and linear arithmetic over string-lengths do not need more explanations, given the motivation presented above, a discussion is in order with respect to our choice of language membership predicates. In this case, they are considered for the classes of regular, deterministic context-free, or as an intermediary between the two, visibly pushdown languages. While there are many classes of languages we might have chosen to consider between regular and deterministic context-free, there are several advantages to choosing the visibly pushdown languages in particular. Firstly, they exhibit an attractive balance of being computationally reasonable (they have many of the desirable closure and algorithmic properties of the regular languages) while simultaneously being powerful enough to provide a reasonable model in many verification and software analysis applications, in line with our motivations from string-solving. Moreover, since they directly generalise the regular languages, but with sufficient memory capabilities to model certain types of length comparisons, the combination of word equations and visibly pushdown language constraints generalises the combination of word equations with both length and regular constraints. The latter is of particular interest in the context of string-solving, but is also a case for which the decidability of satisfiability remains open and is likely to be difficult to resolve. In [16], we have shown that the satisfiability for the former is undecidable and thus that already a very limited extension to regular and length constraints is enough to reach this negative result.

The results: Firstly, a comparison of the relative expressive power of the dif-

ferent theories was obtained. On the one hand, we managed to group certain families together, where they express the same class of languages. We have shown that adding linear arithmetic over string-lengths to a theory allowing only language membership predicates for a class of languages with good language theoretic properties does not alter its expressive power. Thus, the theories in which only regular language (or visibly pushdown language) membership predicates are allowed and the theories in which length comparison is added to those membership predicates are equivalent. While in the case of theories based on regular language membership predicates we can also add concatenation without changing the expressive power, we have shown that adding this operation to theories based on visibly pushdown language membership predicates strictly increases their expressive power, and they can express all recursively enumerable languages. Moreover, we also discuss several separation results between the classes of language expressed by various theories. One of the ways this can be achieved (see [16] for a detailed discussion) is by non-trivially extending pumping-lemma style tools for word equations from [27] to our more general settings, as well as by developing a novel technique for showing inexpressibility by word equations with both regular and length constraints. The overall hierarchy of classes of languages expressible in our theories is depicted in Figure 2.

While these results seem already interesting from a language-theoretic point of view, they are also relevant for the emptiness problem for classes of languages expressed by our theories, which is equivalent to the satisfiability problem for formulas over those theories. As such, our results allowed us to non-trivially extend the state-of-the-art related to the satisfiability of string constraints. In particular, we settled the previously mentioned interesting case in which word equations are combined with visibly pushdown language membership constraints. When combined with existing results, our results establish a relatively complete description of when the emptiness problem is (un)decidable (see upper part of Figure 2). The cases left open are the combinations of word equations with length constraints with or without regular constraints, which remain long-standing open problems.

Further, we have considered the universality problem and a related variant, namely the subset universality problem in which we want to test whether a language is exactly S^* for a subset S of the underlying alphabet. Again, our results filled in gaps in the knowledge and allowed us to paint a comprehensive picture of the decidability status of these problems for our theories (see the right part of Figure 2). Since the universal language is expressible in all our theories, in combination with results from Section 4 and from the literature, we have obtained a complete picture for the equivalence and inclusion problems. However, a substantial further benefit (and a large part of our motivation for studying this problem) is that it allows us to use Greibach's theorem in numerous instances (as stated in Theorem 8) to establish further undecidability results (e.g., Theorems 10 and 9).

In particular, Theorem 10 is part of a larger line of thought, developed in Section 6, in which we have considered the question of when it is (un)decidable if a language expressed in one theory can be expressed in another. Such problems are particularly interesting in the context of practical string solving because they essentially ask whether a property defined by one kind of string constraint can be algorithmically converted to another. Often, it is the combinations of different kinds of string constraints which lead to high complexities in solving, so being able to rewrite constraints in different forms can be a powerful pre-processing technique. We also identify some interesting cases where Greibach's theorem is not applicable, and thus where other approaches are needed (e.g., Theorem 11). The potential implications our results might have in practice are discussed in [16].

The structure of the presentation: Our aim in [16, 15] was to obtain a more complete understanding of the computational properties and expressivity of languages expressed by various combinations of commonly occurring types of string constraints. Naturally, we were able to account for several cases by recalling, or extending existing results from literature, so at the beginning of each section, we give a single theorem that summarizes existing results and discuss their consequences. This allowed us to subsequently focus on the most interesting remaining cases, many of which we were able to resolve by drawing on a range of techniques rooted in formal languages, automata theory, combinatorics on words and computability theory. The results reported in [16, 15] were a substantial improvement of the state of understanding of the theories considered, particularly with respect to their expressive power. In those cases, we were unable to resolve, we have identified several interesting new open problems, which we list here as well.

The proofs of the results overviewed in this paper are given in [16, 15].

2 Preliminaries

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$. Let $\mathbb{Z}$ denote the set of integers. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet. We denote by Σ^* the set of all words over Σ including the empty word, which we denote ε. In other words, Σ^* is the free monoid generated by Σ under the operation of concatenation. For words $u, v \in \Sigma^*$ we denote their concatenation either by $u \cdot v$ or simply as uv. Given a set of variables $\mathcal{X} = \{x_1, x_2, \ldots\}$ and an alphabet Σ, a *word equation* is a pair $(\alpha, \beta) \in (X \cup \Sigma)^* \times (X \cup \Sigma)^*$, usually written as $\alpha = \beta$. A solution to a word equation is a substitution of the variables for words in Σ^* such that both sides of the equation become identical. Formally, we model solutions as morphisms. That is, we say a substitution is a (homo)morphism $h : (X \cup \Sigma)^* \to \Sigma^*$ satisfying $h(a) = a$ for all $a \in \Sigma$, and a solution to a word equation $\alpha = \beta$ is a substitution h such that $h(\alpha) = h(\beta)$.

We refer to [24] for standard definitions and well-known results from formal language theory regarding, for example, recursively enumerable languages (RE), regular languages (REGLang), context free languages (CFLang), deterministic context-free languages (DCFLang), finite and pushdown automata, etc.

In addition, we refer to [3, 4, 5] for background on visibly pushdown automata and visibly pushdown languages (VPLang) but also give here the main definitions. More precisely, a pushdown alphabet $\widetilde{\Sigma}$ is a triple $(\Sigma_c, \Sigma_i, \Sigma_r)$ of pairwise-disjoint alphabets known as the call, internal and return alphabets respectively. A visibly pushdown automaton (VPA) is a pushdown automaton for which the stack operations (i.e. whether a push, pop or neither is performed) are determined by the input symbol which is read. In particular, any transition for which the input symbol a belongs to the call alphabet Σ_c, must push a symbol to the stack while any transition for which $a \in \Sigma_r$ must pop a symbol from the stack unless the stack is empty and any transition for which $a \in \Sigma_i$ must leave the stack unchanged. Acceptance of a word is determined by the state the automaton is in after reading the whole word. The stack does not need to be empty for a word to be accepted. A $\widetilde{\Sigma}$-visibly pushdown language is the set of words accepted by a visibly pushdown automaton with pushdown alphabet $\widetilde{\Sigma}$. A language L is a visibly pushdown language (and is part of the class VPLang) if there exists a pushdown alphabet $\widetilde{\Sigma}$ such that L is a $\widetilde{\Sigma}$-visibly pushdown language. The class VPLang is a strict superset of the class of regular languages and a strict subset of the class of deterministic context-free languages, which retains many of the nice decidability and closure properties of regular languages. In particular, it is shown in [3] that VPLang is closed under union, intersection and complement and moreover that the emptiness, universality, inclusion and equivalence probelms are all decidable for VPLang.

By a *theory*, we mean a set $\mathfrak{T} = \{f_1, f_2, \ldots\}$ of formulas adhering to given syntax and to which we associate a particular semantics. The theories we consider (introduced in Section 3) consist of quantifier-free formulas. The typical computational questions one might consider with respect to a given theory $\mathfrak{T}$ are the following:

- *Satisfiability*: given formula $f \in \mathfrak{T}$, does there exist an assignment of the variables in f such that f becomes true under the associated semantics? and
- *Validity*: given formula $f \in \mathfrak{T}$, is f true under all assignments of the variables occurring in f?

The questions we overview here have a slightly different flavour: given formula $f \in \mathfrak{T}$ and variable x occurring in f, we are interested in properties of the set of all values w for which there is an assignment mapping x to w which makes the formula true. Thus, we consider the set of concrete values w for which f remains satisfiable once the variable x has been replaced by w. Since we shall focus on

theories in which variables represent words, we refer to the set of all such values w as the *language* expressed by the variable x in the formula f. In this respect, we extend the notion of languages expressible by word equations [27] to arbitrary string-based logical theories. We say a language L is *expressed* by a formula if it contains a variable x such that L is the language expressed by x in f. We say that L is *expressible* in a theory $\mathfrak{T}$ if there exists a formula $f \in \mathfrak{T}$ and variable x occurring in f such that L is expressed by x in f.

We shall discuss typical decision problems such as emptiness and universality for languages expressed by formulas in a given theory $\mathfrak{T}$. In this context, the input is a formula $f \in \mathfrak{T}$ and a variable x occurring in f. So, e.g, in the case of emptiness, we might be given a formula $x = aba \wedge x \cdot y = ababba$ along with the variable y, and we must decide whether the language L_y expressed by y in that formula is the empty set or not. In this case, $L_y = \{bba\} \neq \emptyset$ so the answer is no. Clearly, for any formula f and variable x, the emptiness problem for the language expressed by x in f is equivalent to the satisfiability problem for f. Thus, we consider a set of problems which directly generalise the satisfiability problem.

For theories containing word equations, we use and extend notions and results from [27] to reason about (in)expressibility of languages, such as the notion of a *synchronising* $\mathfrak{F}$-factorisation. The technical details about such factorisations can be found in [27], as well as in [16].

3 Logical Theories Over Strings Constraints

In this section, we present a variety of logical theories encompassing the most common kinds of string constraints (as overviewed in [6]). Consider three sets of terms, defined as follows. Let $\mathcal{X} = \{x_1, x_2, \ldots\}$ be an infinite set of string variables. Let Σ be a finite alphabet. Let $\mathcal{T}^{\Sigma}_{str} = \mathcal{X} \cup \Sigma^*$ be the set of *basic string terms*. Let $\mathcal{T}^{\Sigma}_{str,con} = (\mathcal{X} \cup \Sigma)^*$ be the set of *extended string terms*. Note that $\mathcal{T}^{\Sigma}_{str,con}$ is the closure of $\mathcal{T}^{\Sigma}_{str}$ under the concatenation $(\cdot)$ operation. Let $\mathcal{T}^{\Sigma}_{arith} = \{k_0 + k_1|s_1| + k_2|s_2| + \ldots + k_n|s_n| \mid n \in \mathbb{N}_0, k_i \in \mathbb{Z}, \text{ and } s_i \in \mathcal{T}^{\Sigma}_{str}\}$ be the set of *length terms*. We interpret $|s|$ as the length of the string term s, so $\mathcal{T}^{\Sigma}_{arith}$ is the set of linear combinations of lengths of string terms. Note that since we can express the length of a concatenation of string terms as a linear combination of lengths of basic string terms, it is not a restriction the fact that $s_i \in \mathcal{T}^{\Sigma}_{str}$ rather than $\mathcal{T}^{\Sigma}_{str,con}$ (this allows us to consider theories containing length terms both with and without concatenation). We construct three types of atoms from terms as follows:

(A1) Language membership constraints of the form $s \in L$ where $s \in \mathcal{T}^{\Sigma}_{str(,con)}$ and $L \subseteq \Sigma^*$ is a formal language,

(A2) Length constraints of the form $\ell_1 = \ell_2$ where $\ell_1, \ell_2 \in \mathcal{T}^{\Sigma}_{arith}$,

(A3) Word equations (string-equality constraints) of the form $s_1 = s_2$ where $s_1, s_2 \in \mathcal{T}^{\Sigma}_{str,con}$.

Formulas in our theories are constructed in general as follows:

(F1) Any atom is a well-formed formula,

(F2) If f_1, f_2 are well-formed formulas then $\neg f_1$ is a well-formed formula and $f_1 \oplus f_2$ is a well-formed formula for each $\oplus \in \{\wedge, \vee, \implies, \iff\}$.

Note that all formulas are quantifier-free. The semantics associated with these formulas are defined in the natural way: given a substitution for the variables $x_1, x_2, \ldots$ for words in Σ^*, each string term evaluates to a word in Σ^* (possibly as the result of concatenating several smaller words in the case of extended string terms). Each length term is a linear combination of lengths of strings and evaluates to an integer. Atoms of type A1 evaluate to "true" if the string term s evaluates to a word in the language L and false otherwise. Atoms of type A2 evaluate to true if the two length terms ℓ_1, ℓ_2 evaluate to the same integer and false otherwise. Atoms of type A3 evaluate to true if the string terms s_1 and s_2 evaluate to the same word and false otherwise. Finally, Boolean combinations of the form F2 are evaluated in the canonical way.

The most general logical theory we consider includes all of the above and we consider language membership constraints $s \in L$ where L is a deterministic context-free language, given, for instance, as a deterministic push-down automaton or a context-free grammar. However, we are not just interested in this theory alone, rather we want to consider various sub-theories in order to compare their expressive power and computability-related properties.

We have two ways of restricting expressive power. The first is to restrict the types of terms/atoms we allow, while the second is to restrict the kind of languages we allow in the language membership constraints (atoms of type A1). For the latter, we focus on three main possibilities: regular languages, visibly push-down languages, and deterministic context-free languages. For technical completeness, we can assume that all language constraints are given as automata (NFA, Visibly-PDA, or Deterministic-PDA respectively), however, since we do not focus on precise complexity-related issues, equivalent language descriptors such as grammars could equally be used. In particular, we might use simpler descriptors where convenient to do so and where it is obvious that an equivalent automaton could be constructed.

We consider all combinations of atom-types A1, A2 and A3, and in each case define versions in which only basic string terms from $\mathcal{T}^{\Sigma}_{str}$ are allowed and versions in which concatenations of string terms (i.e. terms from $\mathcal{T}^{\Sigma}_{str,con}$) are allowed. Note that whenever we allow word equations (so, atoms of type A3), we might as well

Theory Name	A1-Atoms ($s \in L$)	A2	A3	Example
REG	s basic, $L \in$ REGLang	×	×	$x_1 \in a^*b^* \vee x_1 \in \{c\}^*$
VPL	s basic, $L \in$ VPLang	×	×	$x_1 \in \{a^nb^n \mid n \in \mathbb{N}\}$
DCF	s basic, $L \in$ DCFLang	×	×	$x_1 \in \{a^nb^{2n} \mid n \in \mathbb{N}\}$
REG + CON	s extended, $L \in$ REGLang	×	×	$x_1abx_2 \in (ab)^* \wedge x_2 \in b^*$
VPL + CON	s extended, $L \in$ VPLang	×	×	$x_1abx_2 \in \{a^nb^n \mid n \in \mathbb{N}\}$
DCF + CON	s extended, $L \in$ DCFLang	×	×	$x_1cx_2 \in \{ucv \mid u, v \in \{a, b\}^*, \lvert u\rvert = \lvert v\rvert\}$
REG + LEN	s basic, $L \in$ REGLang	✓	×	$x_1 \in a^* \wedge x_2 \in b^* \wedge \lvert x_1\rvert = \lvert x_2\rvert$
VPL + LEN	s basic, $L \in$ VPLang	✓	×	$x_1 \in \{a^nb^n \mid n \in \mathbb{N}\} \wedge \lvert x_1\rvert = 8$
DCF + LEN	s basic, $L \in$ DCFLang	✓	×	$x_1 \in \{a^nb^{2n} \mid n \in \mathbb{N}\} \vee \lvert x_1\rvert = 3\lvert x_2\rvert$
REG+LEN+CON	s extended, $L \in$ REGLang	✓	×	$x_1x_2 \in a^*b^* \wedge x_2 \in b^* \wedge \lvert x_1\rvert = \lvert x_2\rvert$
VPL+LEN+CON	s extended, $L \in$ VPLang	✓	×	$x_1x_2 \in \{a^nb^n \mid n \in \mathbb{N}\} \wedge \lvert x_1\rvert = \lvert x_2\rvert$
DCF+LEN+CON	s extended, $L \in$ DCFLang	✓	×	$x_1x_2 \in \{a^nb^{2n} \mid n \in \mathbb{N}\} \wedge \lvert x_2\rvert = 2\lvert x_1\rvert$
WE	×	×	✓	$x_1abx_2 = x_2bax_1$
WE + REG	s basic, $L \in$ REGLang	×	✓	$x_1abx_2 = x_2bax_1 \wedge x_1 \in a^*$
WE + VPL	s basic, $L \in$ VPLang	×	✓	$x_1 = x_2x_3 \wedge x_1 \in \{a^nb^n \mid n \in \mathbb{N}\}$
WE + DCF	s basic, $L \in$ DCFLang	×	✓	$x_1 = x_2x_3 \wedge x_2 \in \{a^nb^{2n} \mid n \in \mathbb{N}\} \wedge x_3 \in c^*$
WE + LEN	×	✓	✓	$x_1abx_2 = x_2bax_1 \wedge \lvert x_1\rvert = 2\lvert x_2\rvert$
WE+REG+LEN	s basic, $L \in$ REGLang	✓	✓	$x_1x_2 = x_3 \wedge x_1 \in a^*b^* \wedge \lvert x_2\rvert = \lvert x_3\rvert$
WE+VPL+LEN	s basic, $L \in$ VPLang	✓	✓	$x_1x_2 = x_3 \wedge x_3 \in \{a^nb^n \mid n \in \mathbb{N}\} \wedge \lvert x_1\rvert = \lvert x_3\rvert$
WE+DCF+LEN	s basic, $L \in$ DCFLang	✓	✓	$x_1x_2 = x_3 \wedge x_3 \in \{a^nb^{2n} \mid n \in \mathbb{N}\} \wedge \lvert x_2\rvert = \lvert x_3\rvert$

Figure 1: A list of all the theory-families addressed in the current work, along side descriptions of the allowed atom-types and an example of a formula belonging to that theory family, in the case that the underlying alphabet $\Sigma = \{a, b, c\}$. Language membership constraints are given in shorthand for readability. In the case of visibly pushdown languages, $\{a^nb^n \mid n \in \mathbb{N}\}$ is a typical example under an alphabet-partition $\Sigma = (\Sigma_c, \Sigma_i, \Sigma_r)$ satisfying $a \in \Sigma_c$ and $b \in \Sigma_r$. For each theory-family, permitted atom-types are indicated by a ✓ in the case of A2 and A3, and, for A1-atoms, the class of languages and which kind of string terms are allowed are written explicitly. Atom-types which are not permitted are indicated with ×.

allow concatenations of string terms. If we allow concatenations in word equation terms, then we can model concatenation in all string terms anyway and if we were to restrict equality between string terms to basic string terms only, then we could easily eliminate all string equalities by direct substitution.

Moreover, we are not going to consider explicitly the case that only length constraints (atoms of type A2) are allowed, since this reduces to the existential fragment of Presburger arithmetic and is therefore not really a string-based logic. With these exclusions, we are left with a total of 20 theories to consider; see Figure 1. In fact, since the theories themselves depend on the underlying alphabet Σ, we have 20 families of theories. As such, it is convenient to introduce a naming convention for these (families of) theories.

If atoms of type A1 are allowed, we add either REG, VPL, or DCF to the name of the theory-family depending on the class of languages permitted: REGLang,

VPLang, or DCFLang, respectively. If atoms of type A2 are allowed, we add the abbreviation LEN, separated if necessary by a "+". Likewise, if atoms of type A3 are allowed, we add the abbreviation WE. Finally, if atoms of type A3 are not allowed, but extended string terms are (so we have concatenation but not equality between string terms), then we add the abbreviation CON. Note that CON is superseded by WE due to reasons explained above. For example, the most general theory which allows all three atom types (with deterministic context-free languages for atoms of type A1) is denoted by WE + DCF + LEN. Similarly, REG + LEN + CON describes the theory in which atoms of type A1 (where L is a regular language and s is an extended string term) and A2 are allowed.

For theories allowing VPLang membership constraints (i.e. belonging to families of the form VPL + ...), we assume a fixed partition of the alphabet Σ into the call, return and internal alphabets $\Sigma_c, \Sigma_r, \Sigma_i$. We conclude this section with the following remark.

Remark 1. *Since* REGLang *(respectively,* VPLang*) is closed under union, intersection and complement, the set of languages expressible in* REG *(respectively,* VPL*) is exactly* REGLang *(respectively,* VPLang*). However, the same is not true for* DCF *and* DCFLang*, since that class is not closed, for instance, under intersection. For* DCF *the expressible languages are exactly the Boolean closure of the deterministic context-free languages. Moreover, it can be inferred from well-known results on word equations (see, e.g., [27, 33]) that the languages expressed by* WE *are exactly those expressible by a single word equation in the sense of [27].*

4 Separation and Grouping of Theories

We are interested primarily in whether we can decide properties of a language expressed by a given formula and variable. Therefore, the first thing we consider is the relative expressive power of the various theories defined in the previous section. In particular, we want to understand how the classes of languages which may be expressed by a formula/variable from a given theory relate to each other. To make these comparisons formally, we define the following relation(s) on two logical theories $\mathfrak{T}_1, \mathfrak{T}_2$ whose formulas contain string variables.

Definition 1. *Let $\mathfrak{T}_1, \mathfrak{T}_2$ be theories whose formulas contain string-variables. We say that $\mathfrak{T}_1 \preceq \mathfrak{T}_2$ if, for every formula $f \in \mathfrak{T}_1$ and every (string) variable x occurring in f, there exists a formula $f' \in \mathfrak{T}_2$ and variable x' in f' such that the languages expressed by x in f and x' in f' are identical. Moreover, we say that $\mathfrak{T}_1 \sim \mathfrak{T}_2$ if both $\mathfrak{T}_1 \preceq \mathfrak{T}_2$ and $\mathfrak{T}_2 \preceq \mathfrak{T}_1$ hold. We write $\mathfrak{T}_1 \prec \mathfrak{T}_2$ if $\mathfrak{T}_1 \preceq \mathfrak{T}_2$ and $\mathfrak{T}_1 \not\sim \mathfrak{T}_2$.*

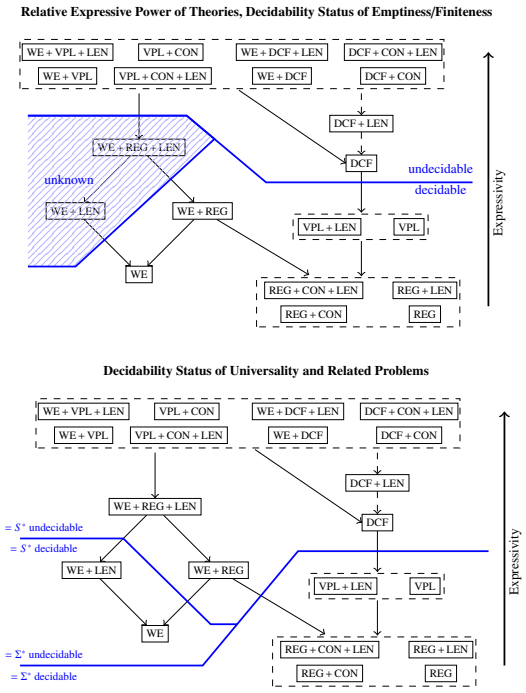


Figure 2: Visual representations of all 20 families of string-based logical theories considered. Theory-families are depicted in solid square boxes containing their names (see Section 3). Dashed square-boxes around multiple theory-families show equivalence with respect to the class of expressible languages (so equivalence under $\sim$). The arrows between theory-families and their transitive closure represent inclusion with respect to the class of expressible languages. Solid arrows indicate that the inclusion is known to be strict, while dashed arrows indicate that we do not know whether the inclusion is strict or not. The most expressive group of theories (i.e. those equivalent to VPL + CON) are able to express RE. The upper figure indicates for which (families of) theories the emptiness and finiteness problems are decidable or undecidable. The lower figure depictes (families) of theories for which the universality ($= \Sigma^*$) and subset-universality ($= S^*$) problems are decidable/undecidable. Equivalence and inclusion (where the two languages might come from different theories) are decidable if and only if both theories fall into cases where universality ($= \Sigma^*$) decidable.

Hence, $\mathfrak{T}_1 \preceq \mathfrak{T}_2$ if the class of languages expressible in $\mathfrak{T}_1$ is a subset of the class of languages expressible in $\mathfrak{T}_2$, and $\mathfrak{T}_1 \sim \mathfrak{T}_2$ if the two classes are equal. Note that the relation $\sim$ is an equivalence relation that is a weaker notion of equivalence than being isomorphic. That is, two theories need not be isomorphic to satisfy the equivalence $\sim$.

We extend Definition 1 for the families of theories defined in Section 3 as follows. Recall that each family contains all the theories consisting of a particular set of formulas, but whose underlying alphabet Σ may vary.

Definition 2. *Let $\mathfrak{F}_1, \mathfrak{F}_2$ be families of theories as defined in Section 3. We say that $\mathfrak{F}_1 \preceq \mathfrak{F}_2$ if, for every theory $\mathfrak{T}_1 \in \mathfrak{F}_1$, there is a theory $\mathfrak{T}_2 \in \mathfrak{F}_2$ such that $\mathfrak{T}_1 \preceq \mathfrak{T}_2$. The relations $\sim$ and $\prec$ are then defined analogously,*

Before moving on, let us make some remarks. It will often be the case that there exist formulas such that the language expressed by a variable x occurring in both formulas is the same, but the sets of satisfying assignments, when considered as a whole, are not identical (see Remark 2 below). This has an important implication for what conclusions we can and cannot draw from a statement of the form, e.g., $\mathfrak{T}_1 \sim \mathfrak{T}_2$. For example, while we will later show that REG $\sim$ REG + LEN, this does not imply that WE + REG + LEN $\sim$ WE + REG. Indeed we shall also show explicitly that the latter does not hold.

Remark 2. *Consider the* LEN *formula $|x| = 2|y|$ where x, y are string variables. Then the language expressed by x is the set of all even-length words over the underlying alphabet Σ, and the language expressed by y is simply Σ^*. Both of these languages are regular, and can be expressed in* REG. *However, if we were to consider, for example, a* WE + LEN *formula $x = yyy \wedge |x| = 2|y|$, then we cannot replace the condition $|x| = 2|y|$ with constraints based on the aforementioned regular languages. The problem with doing so would be that it allows us to decouple the sets of values for x and y satisfying the length constraint (so we get an x, y, x', y' such that $|x| = |y'|$ and $|x'| = |y|$ and $x = yyy$ holds, but where x' might be different from x and y' might be different from y.*

In [27] the authors consider expressibility of languages (and relations) by word equations and show that a language is expressible by WE if and only if it is expressible by a single word equation. The authors of [27] also show that, for $\Sigma \supseteq \{a, b, c\}$, the regular language $\{a, b\}^*$ is not expressible by a single word equation, and thus not in WE. The same holds for the language $\{a^n b^n \mid n \in \mathbb{N}_0\}$. Since these languages are clearly expressible in WE + REG and WE + LEN respectively, we may immediately conclude the following.

Theorem 1 ([27])**.** *The following hold:* WE $\prec$ WE + REG *and* WE $\prec$ WE + LEN.

On the one hand, all languages expressed in our theories are clearly recursively enumerable. On the other hand, in our first main result, we show that, in fact, all recursively enumerable languages can be expressed with only concatenation and VPL-membership.

Theorem 2. *The class of languages expressible in the familiy* VPL + CON *is exactly* RE*.*

Consequently, the class of languages expressible in each of VPL + CON + LEN, WE + VPL, WE + VPL + LEN, DCF + CON, DCF + CON + LEN, WE + DCF, and WE + DCF + LEN is the class of recursively enumerable languages RE. Thus, all these theories are equivalent under $\sim$. Clearly, we immediately get that the satisfiability problem for all these theories is undecidable. In fact, by Rice's Theorem, any non-trivial property is undecidable for languages expressible in these theories. As mentioned before, the case of WE + VPL is particularly interesting in the context of string solving and word equations. Since visibly pushdown languages are seemingly very close to regular languages, with many of the same positive closure and algorithmic properties, it is perhaps surprising to see that while satisfiability for WE + REG is decidable, satisfiability for WE + VPL is undecidable. Moreover, WE + VPL is a very natural extension of WE + LEN + REG. Since the satisfiability problem(s) for WE + LEN + REG and WE + LEN are both long standing open problems of significant interest both to the word equations and string-solving communities, the negative result for WE + VPL is both relevant and ominous.

In this context, it is now natural to ask whether we can separate the classes of languages expressible by WE + LEN + REG and WE + VPL, respectively. The existence of examples of recursively enumerable languages which are not expressible in the former is a necessary condition for having a decidable satisfiability problem, and if we wish to settle this open problem we must also settle the existence of such examples.

The next result does exactly this. In [16] we have established, with some involved argumentation, a sufficient criterion for languages to not be expressible in WE + LEN + REG and have used it to identify a concrete example of such language which is clearly recursively enumerable. To achieve this, we have first used techniques from [27], which were developed to show that certain languages are inexpressible by word equations only. We have first adapted these techniques in the presence of length constraints (so, to obtain languages which are not expressible in WE + LEN). With some care, we have also extended them for regular constraints (so to obtain languages which are not expressible in WE + REG). However, such techniques are altogether unsuitable for direct application in the presence of both length and regular constraints (so for WE + LEN + REG) and a novel approach was required. This new technique, as well as examples not expressible in the

aforementioned theories, are described in details in [16]. The result we obtain is the following.

Theorem 3. *There exist recursively enumerable languages which are not expressible in* WE + LEN + REG *and* WE + LEN + REG $\prec$ WE + VPL. *Moreover, we have* WE + LEN $\prec$ WE + REG + LEN *and* WE + REG $\prec$ WE + REG + LEN, *while the classes of languages expressible in* WE + LEN *and* WE + REG *are incomparable.*

With this result, the relations between the classes of languages expressed by the theories involving word equations is completely clarified; see the left side of Figure 2. Next, we turn our attention to the remaining theories which do not extend the expressive power of word equations. Since we have already seen that concatenation together with visibly pushdown (or deterministic context-free) membership constraints is enough to model recursively enumerable languages, and therefore word equations, the remaining theories consist of language membership without concatenation (but possibly with length constraints) and all combinations consisting of regular language membership constraints without word equations (so including either concatenation, length constraints, both, or neither).

In the following lemma, we state another important result. For this, let C be a class of formal languages which contains REGLang, is contained in CFLang, and is effectively closed under intersection and complement. We assume that the languages of C are specified by an accepting or generating mechanism which allows the construction of a context-free grammar generating that language. Let C_t be the theory defined as in Section 2 which allows only language membership predicates (of type A1) for the class of languages C. Let C_t + LEN be the theory which also allows length constraints. In this framework, the following holds (see the proof in [15]).

Lemma 1. $C_t + \mathrm{LEN} \sim C_t$.

As VPLang is a class which fulfills the properties of the class C from the above lemma, and is strictly included in RE, we immediately get the first claim of the following theorem. The second claim can also be shown with some additional effort (again, see [15]).

Theorem 4. *(1)* VPL $\sim$ VPL + LEN $\prec$ VPL + CON.
(2) REG $\sim$ REG + LEN $\sim$ REG + LEN + CON.

Recall that the languages expressible in REG (as well as the languages expressible in REG + LEN and REG + CON + LEN) and VPL (and VPL + LEN) are exactly the classes REGLang and, respectively, VPLang, and for each formula in one of these theories we can effectively construct a corresponding automaton accepting the language expressed by a given variable. See Remark 3 below.

Remark 3. *In fact, for a formula f in the theory* VPL + LEN *(which includes the theories* REG*,* REG + LEN + CON*, and* VPL*) we can effectively construct a formula g' which is a disjunction of conjunctions g_σ involving at most one membership predicate $x \in L_x$ per variable, where each language L_x is in* VPLang*. We can remove from g' the conjunctions g_σ which contain at least one membership predicate $x \in L_x$ with $L_x = \emptyset$. Now, it is easy to see that for the language expressed by x is exactly the union of the languages L_x for all membership predicates $x \in L_x$ occurring in g'. Therefore, this language is in the class* VPLang*, and we can effectively compute an automaton accepting it. Therefore, we can easily conclude that for two given formulae f and ϕ from* VPL + LEN *and a variable x occurring in f and a variable ϕ occurring in ϕ, we can decide whether the language expressed by x is the same as (respectively, included in) the language expressed by y.*

Let us now turn our attention to the theory DCF. The result of Lemma 1 does not apply in this case, as the class of languages DCFLang is not closed under intersection. In fact, for Lemma 1 to work, it would be enough to have that if L is a finite intersection of languages from the class C then the set $S = \{|w| \mid w \in L\}$ is semi-linear. However, this still does not hold for DCFLang. See Example 1 below.

Example 1. *Let $U_1 = \{a^n b^{2n} \mid n \geq 1\}$ and $L_1 = U_1^+$. Let $U_2 = \{b^n a^n \mid n \geq 1\}$ and $L_2 = aU_2^+ b^+$. It is clear that L_1 and L_2 are in* DCF*. Let $L = L_1 \cap L_2$. It is not hard to observe that $L = \{ab^2a^2b^4a^4b^8 \cdots a^{2^k}b^{2^{k+1}} \mid k \geq 1\}$. Further, let $S = \{|w| \mid w \in L\}$. We have that $S = \{2^{k+2} + 2^{k+1} - 3 \mid k \geq 1\}$. Clearly, S is not a semi-linear set (and it is not a deterministic context-free language either).*

In [15] we show an additional lemma.

Lemma 2. *$L = \{wcw \mid w \in \{a, b\}^*\}$ is expressible in* WE + REG *and not in* DCF*.*

By Theorem 2 and the existence RE-languages which are not expressible in DCF (see [45], as well as Lemma 2 or Example 1), we may infer the following relations: DCF $\preceq$ DCF + LEN $\preceq$ DCF + CON and DCF $\prec$ DCF + CON.

This also shows that at least one of the relations DCF $\preceq$ DCF + LEN and DCF + LEN $\preceq$ DCF + CON is strict. In fact, we can observe (see [15]) that the language $L = \{wcw \mid w \in \{a, b\}^*\}$, which is expressible in DCF + CON and not DCF, is not expressible by a restricted set of formulas in DCF + LEN. Accordingly, this indicates that the separation might occur between DCF + LEN and DCF + CON. We leave the following problem as open.

Open Problem 1. *Investigate whether the relations* DCF $\preceq$ DCF + LEN *and* DCF + LEN $\preceq$ DCF + CON *are strict or not (and note that* DCF $\prec$ DCF + CON*).*

As said before, REG + LEN + CON and VPL + LEN express exactly the classes of regular languages and VPL languages, respectively. Since the regular languages are a strict subset of the VPL languages, which in turn are a strict subset of the deterministic context-free languages, we may conclude the following strict inclusions in terms of expressibility: REG + LEN + CON $\prec$ VPL + LEN $\prec$ DCF.

Finally, note that there are languages expressible in WE (such as $\{xx \mid x \in \Sigma^*\}$) which are not regular nor visibly pushdown, and thus not expressible in REG or VPL or theories with equivalent expressibility. So, REG $\prec$ WE + REG holds. Moreover, we have already seen examples of regular languages which are not expressible in WE or WE + LEN.

Based on the previous results, we can now also discuss the emptiness problem, and the closely related finiteness problem. This is particularly interesting since emptiness for a language expressed by a formula f and variable x corresponds exactly to the satisfiability problem for f. Based on existing literature [3, 24, 41, 33], it is not hard to show that emptiness and finiteness are decidable for VPL and WE + REG but undecidable for DCF.

On the other hand, two cases where it seems particularly difficult to settle the decidability status of the satisfiability and, therefore, emptiness problems are WE + LEN and WE + REG + LEN. Emptiness for the former in particular is equivalent to the satisfiability problem for word equations with length constraints which is a long-standing and important open problem in the field. Similarly, the latter is prominent in the context of string-solving and as such satisfiability/emptiness also presents an important open problem which is likely to be closely related to that of WE + LEN. Consequently, WE + VPL presents a particularly interesting case as a "reasonable" generalisation of WE + REG + LEN and, in the absence of answers regarding this theory, it makes sense to consider the same problems for theories with slightly more or slightly less expressive power. If we extend the expressive power as far as WE + DCF, then undecidability is inherited directly from DCF. However, satisfiability and emptiness remain decidable for VPL. Moreover, visibly pushdown languages share many of the desirable computational properties of regular languages, and, as discussed, we can view WE + VPL as a slighter generalisation of WE + REG + LEN. Nevertheless, due to our result from Theorem 2 of the previous section, we know that VPL + CON expresses already RE, so emptiness and finiteness are undecidable for VPL + CON, and consequently for WE + VPL and other families $\mathfrak{F}$ of theories satisfying VPL + CON $\preceq \mathfrak{F}$. The left part of Figure 2 summarizes the understanding of the emptiness and finiteness problems, as resulting from our results.

5 Universality, Greibach's Theorem, and Expressibility Problems

Universality is an important problem for a number of reasons. Firstly, undecidability of universality implies undecidability of equivalence and inclusion for any theory in which the universal language Σ^* is expressible (which is true in any string-based theory containing at least one tautology). Secondly, an undecidable universality problem is the foundation for Greibach's theorem, which is helpful for proving that many other problems are undecidable. For instance, we shall make use of Greibach's theorem to show several problems concerning expressibility of languages in different theories are undecidable. We recall Greibach's theorem below.

Theorem 5 ([24])**.** *Let C be a class of formal languages over an alphabet $\Sigma \cup \{\#\}$ such that each language in C has some associated finite description. Suppose $\mathcal{P} \subsetneq C$ with $\mathcal{P} \neq \emptyset$ and suppose that all the following hold:*

1. *C and $\mathcal{P}$ both contain all regular languages over $\Sigma \cup \{\#\}$,*
2. *$\mathcal{P}$ is closed under quotient by a single letter,*
3. *Given (descriptions of) $L_1, L_2 \in C$ descriptions of $L_1 \cup L_2$, L_1R and RL_1 can be computed for any regular language $R \in C$,*
4. *It is undecidable whether, given $L \in C$, $L = \Sigma^*$.*

Then the problem of determining, for a language $L \in C$, whether $L \in \mathcal{P}$ is undecidable.

Note that in order to apply Greibach's theorem, we need a variant of the universality problem to be undecidable which refers to a sub-alphabet, rather than the whole alphabet.

Definition 3. *Let $\mathcal{T}$ be a theory defined in Section 3 with underlying alphabet Σ and such that $|\Sigma| \geq 3$. The subset-universality problem is: given a formula $f \in \mathcal{T}$, variable x occurring in f and $S \subset \Sigma$ with $|S| > 1$, is the language expressed by x in f equal to S^*?*

We recall the following results:

Theorem 6 ([21, 17, 3, 24])**.** *Universality is undecidable for* WE *and* DCF*, and decidable for* VPL*. Subset-universality is decidable for* VPL *but not* DCF*.*

To discuss the equivalence and inclusion problems, it makes sense to consider them in a general setting where the two languages may be taken from different theories. We therefore consider equivalence and inclusion problems for pairs of theories $(\mathfrak{T}_1, \mathfrak{T}_2)$. Combining the known results above with the constructive equivalences pointed out in Remark 3, we easily get that equivalence and inclusion for $(\mathfrak{T}_1, \mathfrak{T}_2)$ are undecidable whenever at least one of $\mathfrak{T}_1, \mathfrak{T}_2$ contains WE or DCF, but they are decidable for all other pairs of theories. In a similar way, one can show that cofiniteness is undecidable for WE.

We may, clearly, propagate undecidability of universality and related problems upwards through families of theories containing WE (or DCF) as a syntactic subset, or apply Rice's theorem to get such results for all theories expressing RE. In [15], we also show the following.

Theorem 7. *Subset-universality is decidable for* WE + LEN *and undecidable for* WE + REG. *In particular, for S large enough, for any theory $\mathfrak{T}$ from* WE + REG *with underlying alphabet $\Sigma \supset S$, the problem of whether a language expressed in $\mathfrak{T}$ is exactly S^* is undecidable.*

Theorem 7 allows us to apply Greibach's Theorem to many theories defined in Section 3.

Theorem 8. *Let $\mathfrak{F}$ be a family of theories defined in Section 3 which contains* WE + REG. *For large enough alphabets Σ, if C is the class of languages expressible by the theory $\mathfrak{T} \in \mathfrak{F}$ with underlying alphabet Σ, then the conditions of Greibach's theorem are satisfied by C.*

In the following, we give an example application of Theorem 8 with respect to the pumping lemma for regular languages (see, e.g., [24]). Aside from defining an interesting superclass of the regular languages itself, there are many reasons to be interested in notions of pumping. For example, when considering (in)expressibility questions (even beyond the regular languages), as well as part of a strategy for producing satisfiability results in the context of length constraints or other restrictions. We use the pumping lemma for regular languages because it is well-known, but the ideas are easily adapted to other useful notions of pumping and closure properties more generally. We recall first this lemma.

Lemma 3 ([24])**.** *Let L be a regular language. Then there exists a constant c such that for every $w \in L$ with $|w| > c$, there exist x, y, z such that (i) $|xy| < c$, and (ii) $w = xyz$, and (iii) $xy^n z \in L$ for all $n \in \mathbb{N}_0$.*

Now, Theorem 8 can be applied in this context (see [15]).

Theorem 9. *It is undecidable whether a language expressed by a formula in a theory from* WE + REG *satisfies the pumping lemma for regular languages.*

Theorem 7 also tells us that we cannot use Greibach's theorem as stated to show that properties of languages expressible in WE + LEN are undecidable. We leave as an open problem whether an equivalent of Greibach's theorem can be adapted to this context:

Open Problem 2. *Is there an equivalent of Theorem 5 for the classes of languages expressible in* WE + LEN *or* WE*?*

6 Expressivity Problems

Further, we consider decision problems related to expressivity. These problems have the general form: given a language L expressed by a formula in a theory $\mathfrak{T}_1$ and given a second theory $\mathfrak{T}_2$, can we decide whether or not L can be expressed by a formula in $\mathfrak{T}_2$?

We begin by noting that since it is decidable whether or not a deterministic context-free language is regular (see [42, 44]), the same holds true for visibly pushdown languages, and hence whether a language expressed in VPL can be expressed in REG. Therefore, it is clearly decidable whether a language expressed in VPL is expressible in REG. The same holds for theories from families equivalent to VPL and REG under the relation $\sim$.

Naturally, since we have already seen that VPL + CON is capable of expressing all RE-languages, it is undecidable whether a language expressed in a theory from VPL + CON is expressible in a theory from any of the families which have strictly less expressive power.

The separation results from Section 4 and Theorem 8 together mean we can get the following negative results as a consequence of Greibach's theorem. They have a particularly relevant interpretation in the context of string solving in practice. Specifically, it is often the case that string-solvers will perform some preprocessing of string constraints in order to put them in some sort of normal form which will make them easier to solve. One natural thing to want to do in this process is to reduce the number of combinations of sub-constraints of differing types by converting constraints of one type to another. This is useful particularly in cases where the combinations are difficult to deal with together in general. Word equations, regular constraints and length constraints are one such combination (recall from Section 4 that satisfiability for the corresponding theory including all three types of constraint is an open problem, but if length constraints are removed then satisfiability becomes decidable). Unfortunately, the following theorem reveals that we cannot, in general, decide whether length constraints can be eliminated by rewriting them using only regular membership constraints and word equations.

Theorem 10. *It is undecidable whether a language expressed in* WE + REG + LEN *can be expressed in* WE + REG.

The same undecidability result holds if, instead of removing length constraints by rewriting them as regular membership constraints and word equations, we want to remove word equations constraints by rewriting them as regular language membership constraints (possibly also with length constraints which, in the absence of word equations, do not increase the expressive power due to Theorem 4). While this result can also be obtained via Greibach's theorem, we can, in fact, state a stronger version for which we need a novel approach, detailed in [15]. In particular, we can show that it is already undecidable whether a language expressible by word equations (without additional constraints) is a regular language (i.e., can be expressed in REG).

Theorem 11. *It is undecidable whether a language expressed in* WE *is regular. In other words, it is undecidable whether a language expressed by a formula from* WE *is regular.*

Just as interesting as the result reported in Theorem 11 is the converse problem, which remains open.

Open Problem 3. *Is it decidable whether a regular language is expressible by word equations?*

Although a trivial consequence of Theorem 11, it is somehow surprising that it remains undecidable if a word equation combined with regular constraints expresses a regular language. Clearly, every regular language is trivially expressible in WE + REG.

Finally, we note the remaining cases which correspond to removing regular language membership constraints in the presence of word equations, and removing length constraints in the presence of word equations but without regular constraints. Thus, we leave the following questions open:

Open Problem 4. *Is it decidable whether a language expressed in* WE + REG *(respectively, in* WE + REG + LEN*) can be expressed in* WE *(respectively, in* WE + LEN*)? Is it decidable whether a language expressed in* WE + LEN *can be expressed in* WE*?*

7 Conclusions

Logics based on strings or words are an important topic in fields such as combinatorics on words and formal methods. Motivated primarily by tasks arising in

the automated analysis of software, string-solving, a collection of infinite-domain constraint satisfaction problems whose primary underlying objects are strings, is an area of increasing importance. However, despite considerable improvement in our understanding of this topic, there remains a wealth of open problems and many theoretical topics, particularly involving word equations, are still wide open for new developments.

In [16, 15], which are overviewed in this note, we have studied a variety of string-based logics inspired by typical types of constraints occurring in string-solving applications, such as word equations, length equality constraints and language membership constraints. By considering the formal languages obtained by looking at the set of values a single variable in these logics might take as part of a satisfying assignment for a given formula, we were able to obtain several novel results regarding the relative expressive power of these theories resulting in the hierarchy depicted in Figure 2. Within this broader picture, we have been able to also add new results regarding the computability of canonical decision problems for formal languages such as emptiness, finiteness, universality, equivalence and inclusion (see also Figure 2). Together with existing results, this has allowed us to portray a relatively complete picture of when these problems are and are not decidable within our framework.

Our results on decision problems - in particular (a variant of) the universality problem - created also a framework allogin us to apply Greibach's theorem to obtain further undecidability results. We made use of this tool alongside results overviewed in Section 4 to prove that in various cases, it is undecidable whether or not a language expressed by one theory can also be expressed in another.

On the other hand, we have also highlighted several interesting new open problems in the cases where we are not able to settle the decidability status of certain problems. We expect that studying these problems will lead to valuable new insights and techniques for the theory of word equations, as well as in the theory of formal languages and string-solving, more generally.

References

[1] P. A. Abdulla, M. F. Atig, Y. Chen, L. Holík, A. Rezine, P. Rümmer, and J. Stenman. Norn: An SMT solver for string constraints. In *Proc. Computer Aided Verification (CAV)*, volume 9206 of *Lecture Notes in Computer Science (LNCS)*, pages 462–469, 2015.

[2] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Julian Dolby, Petr Janků, Hsin-Hung Lin, Lukáš Holík, and Wei-Cheng Wu. Efficient handling of string-number conversion. In *Proceedings of the 41st ACM SIGPLAN*

Conference on Programming Language Design and Implementation, pages 943–957, 2020.

[3] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th ACM Symposium on Theory of Computing (STOC)*, STOC '04, pages 202–211, 2004.

[4] Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Congruences for visibly pushdown languages. In *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114. Springer, 2005.

[5] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009.

[6] Roberto Amadini. A survey on string constraint solving. *ACM Computing Surveys (CSUR)*, 55(1):1–38, 2021.

[7] P. Barceló Baeza and P. Muñoz. Graph logics with rational relations: the role of word combinatorics. *ACM Trans. Comput. Logic*, 18(2), 2017.

[8] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022. doi:10.1007/978-3-030-99524-9_24.

[9] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proc. Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science (LNCS)*, pages 171–177, 2011.

[10] Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. Definable relations and first-order query languages over strings. *J. ACM*, 50(5):694–751, 2003.

[11] Murphy Berzish, Mitja Kulczynski, Federico Mora, Florin Manea, Joel D Day, Dirk Nowotka, and Vijay Ganesh. An smt solver for regular expressions and linear arithmetic over string length. In *International Conference on Computer Aided Verification*, pages 289–312. Springer, 2021.

[12] J. R. Büchi and S. Senger. Definability in the existential theory of concatenation and undecidable extensions of this theory. In *The Collected Works of J. Richard Büchi*, pages 671–683. Springer, 1990.

[13] Taolue Chen, Yan Chen, Matthew Hague, Anthony W Lin, and Zhilin Wu. What is decidable about string constraints with the replaceall function. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–29, 2018.

[14] J. D. Day, V. Ganesh, P. He, F. Manea, and D. Nowotka. The satisfiability of word equations: Decidable and undecidable theories. In I. Potapov and P. Reynier, editors, *In Proc. 12th International Conference on Reachability Problems, RP 2018*, volume 11123 of *Lecture Notes in Computer Science (LNCS)*, pages 15–29, 2018.

[15] Joel D. Day, Vijay Ganesh, Nathan Grewal, and Florin Manea. Formal languages via theories over strings. *CoRR*, abs/2205.00475, 2022. `arXiv:2205.00475`, `doi:10.48550/arXiv.2205.00475`.

[16] Joel D. Day, Vijay Ganesh, Nathan Grewal, and Florin Manea. On the expressive power of string constraints. *Proc. ACM Program. Lang.*, 7(POPL):278–308, 2023. `doi:10.1145/3571203`.

[17] V. G. Durnev. Undecidability of the positive $\forall\exists^3$-theory of a free semigroup. *Siberian Mathematical Journal*, 36(5):917–929, 1995.

[18] D. D. Freydenberger. A logic for document spanners. *Theory of Computing Systems*, 63(7):1679–1754, 2019.

[19] D. D. Freydenberger and M. Holldack. Document spanners: From expressive power to decision problems. *Theory of Computing Systems*, 62(4):854–898, 2018.

[20] D. D. Freydenberger and L. Peterfreund. The theory of concatenation over finite models. In *48th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 198 of *LIPIcs*, pages 130:1–130:17, 2021.

[21] V. Ganesh, M. Minnes, A. Solar-Lezama, and M. C. Rinard. Word equations with length constraints: What's decidable? In *Haifa Verification Conference*, 2012.

[22] Matthew Hague. Strings at mosca. *ACM SIGLOG News*, 6(4):4–22, 2019.

[23] Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.

[24] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[25] A. Jeż. Word equations in nondeterministic linear space. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP)*, volume 80 of *LIPIcs*, pages 95:1–95:13, 2017.

[26] Shuanglong Kan, Anthony Widjaja Lin, Philipp Rümmer, and Micha Schrader. Certistr: a certified string solver. In *CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 210–224. ACM, 2022.

[27] J. Karhumäki, F. Mignosi, and W. Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.

[28] A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst. HAMPI: a solver for string constraints. In *Proc. ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 105–116. ACM, 2009.

[29] Quang Loc Le and Mengda He. A decision procedure for string logic with quadratic equations, regular expressions and length constraints. In *Asian Symposium on Programming Languages and Systems*, pages 350–372. Springer, 2018.

[30] Tianyi Liang, Nestan Tsiskaridze, Andrew Reynolds, Cesare Tinelli, and Clark Barrett. A decision procedure for regular membership and length constraints over unbounded strings. In *International Symposium on Frontiers of Combining Systems*, pages 135–150. Springer, 2015.

[31] Anthony W Lin and Pablo Barceló. String solving with word equations and transducers: towards a logic for analysing mutation xss. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 123–136, 2016.

[32] M. Lothaire. *Combinatorics on words*, volume 17. Cambridge university press, 1997.

[33] M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, Cambridge, New York, 2002.

[34] Rupak Majumdar and Anthony W Lin. Quadratic word equations with length constraints, counter systems, and Presburger arithmetic with divisibility. *Logical Methods in Computer Science*, 17, 2021.

[35] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Sbornik: Mathematics*, 32(2):129–198, 1977.

[36] Federico Mora, Murphy Berzish, Mitja Kulczynski, Dirk Nowotka, and Vijay Ganesh. Z3str4: A multi-armed string solver. In Marieke Huisman, Corina S. Pasareanu, and Naijun Zhan, editors, *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings*, volume 13047 of *Lecture Notes in Computer Science*, pages 389–406. Springer, 2021. doi:10.1007/978-3-030-90870-6_21.

[37] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proc. Foundations of Computer Science (FOCS)*, pages 495–500. IEEE, 1999.

[38] W. V. Quine. Concatenation as a basis for arithmetic. *The Journal of Symbolic Logic*, 11(4):105–114, 1946.

[39] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*. Springer, 1997. doi:10.1007/978-3-642-59136-5.

[40] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages, Volume 3: Beyond Words*. Springer, 1997. doi:10.1007/978-3-642-59126-6.

[41] K. U. Schulz. Makanin's algorithm for word equations-two improvements and a generalization. In *International Workshop on Word Equations and Related Topics*, pages 85–150. Springer, 1990.

[42] R.E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967.

[43] Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. Progressive reasoning over recursively-defined strings. In *International Conference on Computer Aided Verification*, pages 218–240. Springer, 2016.

[44] L. G. Valiant. Regularity and related problems for deterministic pushdown automata. *Journal of the ACM (JACM)*, 22(1):1–10, 1975.

[45] Detlef Wotschke. The boolean closures of the deterministic and nondeterministic context-free languages. In *GI Gesellschaft für Informatik e. V.*, pages 113–121. Springer, 1973.

THE DISTRIBUTED COMPUTING COLUMN

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

This month, in the Distributed Computing Column, Michel Raynal revisits two classical problems: mutual exclusion and consensus. Both of these problems are central to distributed computing. Many date the birth of distributed computing, as a field, to Dijkstra's first paper on mutual exclusion in 1965, and it remains an area of active research today (see, e.g., new breakthroughs on recoverable mutual exclusion this year). Similarly, ever since the consensus problem was first formally defined in a 1980 paper by Pease, Shostak, and Lamport, it has been recognized as lying at the heart of distributed computing. Recent innovation in blockchains (which rely on consensus to order blocks) only reinforces the continued relevance of consenus today. Yet research on these two central problems has followed somewhat different paths, focusing on different models and different aspects of concurrency and fault-tolerance. In this short note, Michel Raynal provides a historical overview of their development, and argues that they are really "two sides of the same coin."

The Distributed Computing Column is particularly interested in contributions that propose interesting new directions and summarize important open problems in areas of interest. If you would like to write such a column, please contact me.

BEATCS no 140

Mutual Exclusion vs Consensus: Both Sides of the Same Coin?

Michel Raynal
IRISA, CNRS, Inria, Univ Rennes, 35042 Rennes, France

Abstract

This short note shows that consensus is to logical objects what mutual exclusion (mutex) is to physical objects. Namely, both allow processes to cooperate in a consistent way through objects operations of which must be executed sequentially (i.e., objects defined by a sequential specification).

Keywords: Agreement, Asynchrony, Concurrency, Consensus, Liveness, Logical (immaterial) object, Mutual exclusion, Physical object, Safety, Sequential specification, Total order.

1 Concurrent Computing

While a sequential process describes the behavior of a given state machine [42], *concurrent computing* is about the study of asynchronous sequential processes that execute concurrently (i.e., possibly at the same time) but not independently from each other[1]. Asynchronous means that each process proceeds to its own speed, which can vary with time and remains always unknown to the other processes. The code executed by the processes can be specific or not to each process.

Considering a set of sequential processes, the concept of *concurrent processes* (multi-process program) captures the fact that the individual behavior of each sequential process must be controlled so that the global behavior of the set of processes remains consistent (which can be captured by predicates and invariants, e.g., [21, 22, 30, 31]). These fundamental notions have been introduced by E.W. Dijkstra in the early sixties [17, 18, 19, 20]. (The interested reader will find more historical and scientific developments in [2, 6, 32, 36, 39, 45, 53, 55, 57, 58].)

[1] At the very beginning, the corresponding underlying multiprocessor machine was simulated on a single mono-processor enriched with peripheral devices. Then it was a real physical multiprocessor. Today it is provided by what is sometimes called an *Internet machine* covering the world.

2 Parallel Computing vs Distributed Computing

Parallel computing Parallel computing is a natural extension of sequential computing in the sense that the aim of parallel computing is to detect and exploit *data independence* to obtain efficient programs: once identified, independent sets of data can be processed independently from each other on a multiprocessor. It is nevertheless important to notice that, while independent data can be processed in parallel, any parallel program could be executed on a single processor with an appropriate scheduler (the corresponding sequential execution could be of course highly inefficient!).

Distributed computing The nature of distributed computing is totally different. Namely, distributed computing is characterized by the fact that there is a set of predefined (and physically distributed) computing entities (processes) that are *imposed* to the programmers and these entities need to *cooperate to a common goal*. Moreover the behavior of the underlying infrastructure (also called environment) on which the distributed application is executed is not under the control of the programmers who have to consider it as an *hidden input*. Asynchrony and failures are the most frequent phenomenons produced by the environment that create a "context uncertainty" distributed computing has to cope with. In short, distributed computing is characterized by the fact that, in any distributed run, *the run itself is one of its entries* [54].

A duality To summarize, parallel computing is the exploitation of the independence of input data to obtain efficient algorithms (programs), while the aim of distributed computing is to allow predefined computing entities to cooperate to a common goal in a consistent way.

3 Mutex vs Consensus: Preliminary Remark

Both problems were originally addressed in different system models, namely *asynchronous systems with no failures* for mutex [18] (the only "adversary" was asynchrony), and *synchronous systems with Byzantine failures* for consensus [48] (the only "adversary" was Byzantine failures).

They were then extended to more general system models including both asynchrony and process failures, namely asynchronous systems with process crashes failures for mutex and asynchronous systems with process crashes/Byzantine failures for consensus. As we will see, this required to add computability power to the underlying system model for these problems can be solved.

4 1965: Mutual Exclusion

The very first objects that were shared by concurrent processes were *physical* objects (resources) such as discs, tapes, and shared memory. Mutual exclusion was then introduced to make their accesses by the processes consistent (what does happen if several processes simultaneously access such a physical object?). The problem and its answer were introduced by E.W. Dijkstra who proposed the notion of a *critical section*, namely a part of code that can be accessed by a single process at a time [18].

The mutex object To ensure this property, E.W. Dijkstra introduced a new concurrency-related object that we call here *mutex* (shortcut for mutual exclusion). This synchronization object provides processes with two operations denoted acquire() and release() that allow to bracket the critical section code as described by the following pattern:

acquire(); *critical section*; release().

As any computing object, the mutex object is specified with a set of properties that describe all its correct behaviors, namely:

- Mutual exclusion (safety). At most one process at a time executes the critical section.
- Starvation freedom (liveness). Any invocation of acquire() terminates (and consequently the invoking process eventually enters the critical section)[2]. (Let us observe that fact that any invocation of acquire() must terminate implies that any invocation of release() must also terminate.)

Encapsulation and sequential execution The two operations acquire() and release() are not visible at the application level. At this level a process invokes higher level operations, e.g., op() such that

operation op() **is** acquire(); *critical section*; release() **end operation**.

In some cases the object protected by a mutex object, say `mtr`, provides processes with several operations. This is for example the case of two resources $R1$ and $R2$ that, due to energy consumption, cannot be used at the very same time. In this case we have a single mutex object and two operations

operation opR1() **is** `mtr`.acquire(); *access* $R1$; `mtr`.release() **end operation**,

[2]When he introduced mutual exclusion, Dijkstra considered a weaker liveness property, named deadlock-freedom: If one or several processes invoke acquire(), at least one of them will enter the critical section.

and
operation opR2() **is** `mtr`.acquire(); *access R*2; `mtr`.release() **end operation**.

This approach has also naturally been used for data objects (for example a stack where opR1 is push() and opR2 is pop()).

It is easy to see that mutual exclusion allows processes to execute sequentially (we also say linearize [29]) predefined parts of code concerning their cooperation. So, mutual exclusion allows the processes to build a total order on the execution of the critical section codes protected by the same mutex object. From a historical point of view, mutex can be considered as the first distributed computing problem: it allows a predefined set of processes to cooperate to a common goal, namely preserve the consistency of an object in the presence of concurrency. A rigorous exposition of the mutex theory is presented in [38].

Instantiating a mutex algorithm Let us consider a n-process system. While it is possible to design mutex algorithms tailored for ad'hoc values of n (for example there are mutex algorithms specifically designed for two processes only), e.g. [50], and consequently such algorithms do not work for more than two processes. Nearly all mutex algorithms are designed to work for any value of $n \geq 2$, i.e., n is a parameter that can differ in each instance of the algorithm.[3].

On the fault-tolerance side A process crashes when it unexpectedly and definitively halts. Usual algorithms that solve mutex allow a process to crash when it is not executing acquire(), release() or the code in the critical section. Unfortunately mutual exclusion cannot solved if a process may crash at any time. This is due to the fact that if a process crashes while executing acquire(), release() or the code in the critical section, due to asynchrony, no other process can be informed of its crash. To solve this issue, the system must be enriched with additional computational power.

An approach consists in providing processes with information on failures. This is the *failure detector* approach introduced in [11]. The integer n being the number of processes, let us consider a model that allows up to t processes to crash. When considering systems where processes communicate through read/write registers, the weakest failure detector (denoted QP for Quasi-Perfect) that allows mutex to be solved has been introduced in [16]. Weakest means that no failure detector that provides processes with less information on failures than QP allows mutex to be solved in read/write systems. Assuming $t < n/2$ (i.e. the system

[3] As we will see in Section 7, due to computability issues, the situation is different for consensus where a consensus algorithm for n processes does not work for $(n + x)$ processes for $x \geq 1$. This is related to the additional computability power needed to solve consensus in crash-prone systems.

is partition-free), the weakest failure detector (denoted T) that allows mutual exclusion to solved in synchronous message-passing systems has been introduced in [15].

These two failure detectors have close but different definitions. Both are weaker that the perfect failure P and stronger than the eventually perfect failure detector $\diamond P$ defined in [11]. Moreover, both are stronger than the weakest failure detector (eventual leader denoted Ω) that allows consensus to be solved in read/write systems when $t < n$ [43] and in message-passing when $t < n/2$ [10].

Transactional memory The concept of transactional memory was proposed by M. Herlihy and J. Moss in 1993 [28], and later refined by N. Shavit and D. Touitou[56]. The idea is to provide the designers of multiprocess programs with a language construct (namely, the notion of an atomic operation called a *transaction*) that discharges them from the management of synchronization issues. More precisely, a programmer has to concentrate her efforts only on defining which parts of processes have to be executed atomically and not on the way atomicity (mutual exclusion) is realized, this last issue being automatically handled by the underlying system.

5 A Trivial Observation: Physical Objects vs Logical Objects

A *physical* object is an object that cannot be replicated by software (e.g., a printer), while a *logical* (or *immaterial*) object is an object the value of which can by replicated by software (data). Said differently, at the basic level the value of a logical object is a structured set of bits while a physical object is a hardware device.

6 1971, 1977: Once Upon a Time: the Readers/Writers Problem

A file is a logical object that provides processes with two operations: `read_file`() that allows a process to read the file and `write_file`() that allows a process to modify its content.

The Readers/Writers Problem It was observed by P. Courtois, F. Heymans, and D. Parnas [14] (1971) that mutual exclusion is stronger than necessary to provide the synchronization needed to correctly implement the `write_file`() and `read_file`() operations, namely, only each execution of `write_file`() must be

executed in mutual exclusion with any other operation execution (`read_file`() or `write_file`()), while the executions `read_file`() needs to be executed in mutual exclusion only with respect to `write_file`() (and not among themselves). As far as we know, this was the first (implicit) distinction between physical and logical objects.

The wait-freedom approach This approach was later generalized by L. Lamport to allow concurrent readings while writing [34] (1977), which showed that (as a file is a logical object) the readers/writer problem does not need mutual exclusion to be solved (see also [51]). This approach culminated in the notion of *wait-free* computing introduced by M. Herlihy [26]. Wait-free means that the progress of a process cannot be prevented by the behavior of the other processes (arbitrary unknown speed or crash failures).

From safe read/write bits to atomic read/write registers In a very interesting way, it has been shown by L. Lamport that, while the *atomicity* of basic read/write registers are sufficient to solve mutex, they are not necessary to solve it. More precisely, mutual exclusion can be solved on top of single-writer multi-reader *safe* registers (see [33, 37]). A safe register is a register that can be written by a single process and read by any number of processes. A write defines the new value of the register. A read whose execution is not concurrent with a write returns the last value written in the register. A read concurrent with a write returns *any value* that the register can contain (so it can return a value that has never been written in the register!). In a non-trivial way, multi-writer multi-reader atomic registers can be built on top of single-writer single-reader safe bits (despite asynchrony and process failures. A survey of such constructions is presented in Section V of [53].

7 1980: The Advent of Consensus: On Fault-Tolerant Distributed Computing

Definition The *consensus* problem was introduced by S. Pease, R. Shostak and L. Lamport in [41, 48] in the context of synchronous distributed systems prone to Byzantine process failures (arbitrary misbehavior of a process). This problem is at the core of distributed computing agreement problems. We consider here asynchronous read/write or message-passing systems prone to crash failures. Let a process be *correct* in a run if it does not crash during that run. In such a context a consensus object is defined by a single operation denoted propose() that takes a value as input parameter and returns a value. When a process invokes propose(v)

and obtains the value v' we say that it proposes v and decides v'. Consensus is defined by the following properties.

- Validity (safety). If a process decides v then a process proposed v.
- Agreement(safety). No two processes decide different values.
- Termination (liveness). If a process invokes propose() and does crashes, it decides.

Impossibility Unfortunately consensus is impossible to solve in the presence of asynchrony and even a single process crash, be the communication system message-passing [24] or read/write registers [44]. This means that the system has to be enriched with additional *computability power* to make consensus solvable. Several enrichments are possible.

- Enrich the system with synchrony assumptions (e.g. [23]).
- Enrich the system with scheduling assumptions (e.g. [5]).
- Enrich the system with randomization (e.g. [4, 46]).
- Restrict the set of input vectors that can be proposed (e.g. [47]). (An input vector has one entry per process containing the value it proposes. Of course a process knows only the value of its entry).
- Enrich the system with information on failures (failure detector approach [10, 11]).
- Enrich the system with asynchronous rounds such that, for each round r and each process p, the model provides the set of processes that p hears of at round r. The features of a specific system is then captured as a whole, just by a predicate over the collection of heard-of sets [12].

Consensus number of an object Let us consider an asynchronous crash prone system in which the processes communicate by reading and writing atomic registers (RW type). As just noticed, the previous impossibility results states that consensus cannot be solved in such a system. So, a fundamental question is: which additional computability power (defined not in terms of system behaviors but in terms additional object types) needs to be added to the system model so that the consensus can be solved. To this end, M. Herlihy introduced the notion of *consensus number* [26].

The consensus number of an object type T, denoted $CN(T)$, is the greatest number of processes for which consensus can be solved from any number of atomic read/write registers and any number of objects of type T. If there is no such greatest number, the consensus number of T is $+\infty$.

Let RW_TS be the type of RW registers accessed with Test&Set() operation, and RW_CS be the type of RW registers accessed with Compare&Swap() operation[4]. It has been shown in [26] that CN(RW_TS)= 2 and CN(RW_CS)= $+\infty$. More generally, [26] introduces an infinite hierarchy of objects, that cover all possible consensus numbers. The interested reader can look at [49] where is defined the notion of k-sliding window RW register. This object family spans the whole consensus hierarchy: the consensus number of the k-sliding window RW register is exactly k.

8 Consensus: a Simple Way to Agree on a Total Order

Ordering object operations Let us consider an object defined by a sequential specification, e.g., a stack with its two operations push() and pop(). To cope with asynchrony and failures, the stack (which is a logical object, i.e. a structured set of bits) is replicated on each process. So the main issue consists in ensuring that the push() and pop() operations issued by the processes are applied in the same order to all the local copies of the stack. A simple way to attain this goal consists for each process in:

1. announcing the operation it wants to execute,
2. regularly defines a sequence on the operations it sees as announced and not yet executed,
3. and proposes this sequence as input to a consensus instance.

Combined with a sequence of consensus instances (in which all processes agree a priori), this allows all the local copies of the stack to progress the same way [10, 26].

Hence, as it allows to build a total order on operations, consensus lies at the core of fault-tolerant implementations for the objects defined by a sequential specification. (For objects not defined by a sequential specification, i.e. concurrent objects, the reader can consult [7, 8, 52].)

[4]Roughly speaking both operations return the current value of the register and write a new value in it. The difference lies in the fact that Test&Set() is an unconditional write of a predefined value, while Compare&Swap() is a conditional write of a value.

Consensus vs mutex: illustration Let us consider money transfer as an object providing its users (a user is a process associated with one and only one money account) with two operations transfer() that allows a process to transfer money from its account to another account, and balance() that allows a user to read an account. Let us observe that an account is a logical object.

It has recently been shown that money transfer among a set of processes, each having its own account, does not need consensus [3, 13, 25] [5]. It is an announcement/broadcast problem that must satisfy some causality requirements.

When several persons share the same account, the associated process consists of several threads, one per person co-owner of the account. The invocations of the operations transfer() issued by the threads that are co-owners of the same account must then be ordered in order to prevent double-spending from the corresponding account. This could be realized with mutex (enriched with an appropriate failure detector or random numbers if the system is crash-prone).

But, as an account is a logical object this ordering can be realized (despite process failures and asynchrony) with the help of consensus. It follows that if each account can be accessed by at most k threads, an object the consensus number of which is k is sufficient to realize money transfer (this was first noticed in [25]).

9 Both Sides of the Same Coin

When considering objects the consistency of which is defined by a sequential specification (i.e, objects whose operations must appear as being executed sequentially), it follows from the previous simple observations that, while both mutex and consensus can be used to build a total order, mutex is for physical objects (which by nature cannot be replicated), and consensus is for logical objects (structured sets of bits which can be replicated)[6]. In this sense, mutex and consensus are the two sides of the same coin. The content of this note is summarized in Table 1.

The "Underlying coordination" column refers to the type of synchronization needed to implement mutex or consensus, namely, mutex ensures that the concerned object can be physically accessed by at most one process at a time, while consensus does not prevent several processes from invoking and simultaneously executing object operations (after these operations have been totally ordered by a consensus instance). The column "Helping needed" refers to the fact the al-

[5] It is pleasant to observe that the heavy Blockchain machinery was introduced to built a total order on the cryptocurrency operations issued by users, and this is not needed! For the interested reader, [13, 25] consider money transfer as an object defined by a sequential specification, while [3] considers money transfer as an object defined by a concurrent specification.

[6] Of course, in some specific contexts, it can be interesting to use mutex for logical objects, but this is another issue not addressed in this note.

Nature of the object	Possible replication	Total order obtained from	Underlying Coordination	Helping needed	Weakest FD
Physical	No	Mutex	strong	Yes	QP, T
Logical	Yes	Consensus	weak	Yes	Ω

Table 1: Total order: mutex vs consensus

gorithms implementing mutex or consensus need specific helping mechanisms to ensure the liveness of the operations on the object that is built [1, 9, 26, 54][7]. The last column "Weakest FD" concerns the weakest failure detectors that allows mutex or consensus to be solved. As already indicated, for read/write systems it is the failure detector QP for mutex [16] and Ω for consensus [43], while, for message-passing systems such that $t < n/2$, it is the failure detector T for mutex [15] and the eventual leader failure detector Ω for consensus [10]. It is worth noticing that the weakest information on failures that allows mutex to be solved includes a perpetual property [15, 16], while that the weakest information on failures needed to solve consensus needs to satisfy an eventual property only [10]. This is strongly related to the underlying nature of the object (physical vs logical).

Let us again insist on the fact that, in a crash-prone system where the processes communicate through read/write atomic registers (resp. message-passing when assuming $t < n/2$), the weakest failure detectors QP (resp. T) that allows mutex to be solved is stronger than the weakest failure detector Ω that allows consensus to be solved. As previously noticed, this is due to the fact that the implementation of mutex requires a stronger underlying synchronization than the one needed to implement consensus. More precisely, this is the main difference between mutex and consensus, because of their very definitions mutex does not allow concurrency at the implementation level, whereas consensus does.

Last but not least, let us notice that a recent paper by L. Lamport [40] describes a deconstruction of his famous Bakery mutex algorithm [33] from which is built a distributed state machine as defined in [35] (i.e., any object defined by a sequential specification). This can be seen as an answer to the question posed in the title of this note.

[7] As far liveness properties are concerned, wait-freedom [26] and non-blocking [29] for consensus correspond to starvation-freedom and deadlock-freedom for mutex. Differently obstruction-freedom [27] for consensus has no corresponding liveness property that could be associated with mutex (this is due to the fact that mutex implicitly considers the object to with it is applied as a "physical" object.

References

[1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M., and Shavit N., Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873-890 (1993)

[2] Apt K. R. and Hoare C.A.R. (editors), *Edsger Wybe Dijkstra: his life, work, and legacy*. Association for computing machinery, Morgan & Claypool Publishers, 550 pages (2022)

[3] Auvolat A., Frey D., Raynal M. and Taïani F., Money transfer made simple: a specification, a generic algorithm, and its proof. *Electronic Bulletin of EATCS (European Association of Theoretical Computer Science)*, 132:22–43 (2020)

[4] Ben-Or M., Another advantage of free choice: completely asynchronous agreement protocols. *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC'83)*, ACM Press, pp. 27–30 (1983)

[5] Bracha G. and Toueg S., Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840 (1985)

[6] Brinch Hansen P. (Editor), *The origin of concurrent programming*. Springer, 534 pages (2002)

[7] Castañeda A., Rajsbaum S., and Raynal M., Unifying concurrent objects and distributed tasks: Interval-linearizability. *Journal of the ACM*, 65(6), Article 45, 42 pages (2018)

[8] Castañeda A., Rajsbaum S., and Raynal M., A linearizability-based hierarchy for concurrent specifications. *Communications of the ACM*, 66(1):60–71 (2023)

[9] Censor-Hillel K., Petrank E. and Timnat S., Help!, *Proc. 34th ACM Symposium on Principles of Distributed Computing (PODC'15)*, ACM Press, pages 241–250 (2015)

[10] Chandra T.D., Hadzilacos V., and Toueg S., The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722 (1996)

[11] Chandra T. and Toueg S., Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267 (1996)

[12] Charron-Bost and Schiper A., The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22:49–71 (2009)

[13] Collins D., Guerraoui R., Komatovic J., Monti M., Xygkis A., Pavlovic M., Kuznetsov P., Pignolet Y.-A., Seredinschi D.A., and Tonlikh A., Online payments by merely broadcasting messages. *Proc. 50th IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN'20)*, pp. 26–38 (2020)

[14] Courtois P.J., Heymans F., and Parnas D.L., Concurrent control with readers and writers. *Communications of the ACM*, 14(5):667–668 (1971)

[15] Delporte-Gallet C., Fauconnier H., Guerraoui R. and Kouznetsov P., Mutual exclusion in asynchronous systems with failure detectors. *Journal of Parallel and Distributed Computing*, 65:492-505 (2005)

[16] Delporte-Gallet C., Fauconnier H., and Raynal M., On the weakest information on failures to solve mutual exclusion and consensus in asynchronous crash prone read/write systems. *Journal of Parallel and Distributed Computing*, 153:110–118 (2021)

[17] Dijkstra E.W., Over de sequentialiteit van procesbeschrijvingen (on the nature of sequential processes). *EW Dijkstra Archive (EWD-35)*, Center for American History, University of Texas at Austin (Translation by Martien van der Burgt and Heather Lawrence) (1962)

[18] Dijkstra E.W., Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569 (1965)

[19] Dijkstra E.W., Cooperating sequential processes. In *Programming Languages (F. Genuys Ed.)*, Academic Press, pp. 43-112 (1968)

[20] Dijkstra E.W., Hierarchical ordering of sequential processes. *Acta Informatica*, 1(1):115–138 (1971)

[21] Dijkstra E.W., Guarded commands, non-determinacy and formal derivation of programs. *Communications of the ACM*, 8:453–457 (1975)

[22] Dijkstra E.W., Dahl O.-J., and Hoare C.A.R., Structured programming. *Academic Press*, 220 pages (1972)

[23] Dolev D., Dwork C., and Stockmeyer L., On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97 (1987)

[24] Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382 (1985)

[25] Guerraoui R., Kuznetsov P., Monti M.,Pavlovic M., Seredinschi D.A., The consensus number of a cryptocurrency. *Distributed Computing*, 35:1–15 (2022)

[26] Herlihy M.P., Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149 (1991)

[27] Herlihy M.P., Luchangco V., and Moir M., Obstruction-free synchronization: double-ended queues as an example. *Proc. 23th Int'l IEEE Conference on Distributed Computing Systems (ICDCS'03)*, IEEE Press, pp. 522-529 (2003)

[28] Herlihy M.P. and Moss J.E.B., Transactional memory: architectural support for lock-free data structures. *Proc. 20th ACM Int'l Symposium on Computer Architecture (ISCA'93)*, ACM Press, pp. 289–300 (1993)

[29] Herlihy M.P. and Wing J.M., Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*,12(3):463–492, (1990)

[30] Hoare C.A.R., An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580 (1969)

[31] Hoare C.A.R., Programming: sorcery or science? *IEEE Software*, 1(2):5–16 (1984)

[32] Jones C.B. and Misra J.(editors), *Theories of programming: the life and works of Tony Hoare*, Association for computing machinery, Morgan & Claypool Publishers, 430 pages (2021)

[33] Lamport L., A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455 (1974)

[34] Lamport L., Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811 (1977)

[35] Lamport L., Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565 (1978)

[36] Lamport L., On inter-process communications, part I: basic formalism. *Distributed Computing*, 1(2):77–85 (1986)

[37] Lamport L., On inter-process communications, part II: algorithms. *Distributed Computing*, 1(2):86–101 (1986)

[38] Lamport L., The mutual exclusion problem: Part I- a theory of interprocess communication. *Journal of the ACM*, 33(2): 313–326 (1986)

[39] Lamport L., The computer science of concurrency: the early years (Turing lecture). *Communications of the ACM*, 58(6):71–76 (2015)

[40] Lamport L., Deconstructing the Bakery to build a distributed state machine. *Communications of the ACM*, 65(9):58–66 (2022)

[41] Lamport L., Shostak R., and Pease S., The Byzantine generals problem, *ACM Transactions on Programming Languages and Systems*, 4(3):382–401 (1982)

[42] Lewis H.R. and Papadimitriou C.H., *Elements of the theory of computation*, Prentice Hall Int. Editions, 361 pages (1998)

[43] Lo W.K. and Hadzilacos V., Using failure detectors to solve consensus in asynchronous shared memory systems. *Proc. 8th Int'l Workshop on Distributed Algorithms (WDAG'94)*, Springer LNCS 857, pages 280–295 (1994)

[44] Loui M. and Abu-Amara H., Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, JAI Press Inc. (1987)

[45] Malkhi D. (Editor), *Concurrency: the works of Leslie Lamport.* Association for computing machinery, Morgan & Claypool Publishers, 345 pages (2019)

[46] Mostéfaoui A., Moumen H., and Raynal M., Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *Journal of the ACM*, 62(4), Article 31, 21 pages, (2015)

[47] Mostéfaoui A., Rajsbaum S. and Raynal M., Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6):922–954 (2003)

[48] Pease M., Shostak R., and Lamport L., Reaching agreement in the presence of faults. *Journal of the ACM*, 27:228–234 (1980)

[49] Perrin M., Mostéfaoui and Raynal M., A simple object that spans the whole consensus hierarchy. *Parallel Processing Letters* 28(2), 1850006, 9 pages (2018)

[50] Peterson G.L., Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, 1981.

[51] Peterson G.L., Concurrent reading while writing. *ACM Transactions on Programming Languages and Systems*, 5:46-55 (1983)

[52] Rajsbaum S. and Raynal M., Mastering concurrent computing through sequential thinking. *Communications of the ACM*, Vol. 63(1):78–87 (2020)

[53] Raynal M., *Concurrent programming: algorithms, principles and foundations*. Springer, 515 pages, ISBN 978-3-642-32026-2 (2013)

[54] Raynal M., *Fault-tolerant message-passing distributed systems: an algorithmic approach.* Springer, 550 pages (2018)

[55] Raynal M., and Taubenfeld G., A visit to mutual exclusion in seven dates. *Theoretical Computer Science*, 919:47–65 (2022)

[56] Shavit N. and Touitou D., Software transactional memory. *Distributed Computing*, 10(2):99-116 (1997)

[57] Taubenfeld G., *Synchronization algorithms and concurrent programming*. Pearson Education/Prentice Hall, 423 pages, ISBN 0-131-97259-6 (2006)

[58] Taubenfeld G., Concurrent programming, mutual exclusion. *Springer Encyclopedia of Algorithms*, pp. 421–425 (2016)

The Education Column

BY

Juraj Hromkovič and Dennis Komm

ETH Zürich, Switzerland

juraj.hromkovic@inf.ethz.ch and dennis.komm@inf.ethz.ch

BEATCS no 140

BEBRAS: INSPIRING INFORMATICS EDUCATION ACROSS THE GLOBE

Valentina Dagienė
Vilnius University, Lithuania
valentina.dagiene@mif.vu.lt

Abstract

The Bebras challenge is an international initiative that aims to promote informatics and computational thinking concepts to students in schools. It is designed as a contest where participants solve a set of tasks that require problem-solving skills, logical reasoning, and algorithmic thinking. The challenge is open to students of various age groups, typically ranging from primary school to high school. It provides an opportunity for students to engage with informatics concepts in a fun and interactive way. The tasks are carefully designed to be intellectually stimulating and encourage students to think critically and creatively. The Bebras challenge focuses on concept-based tasks that cover a wide range of informatics topics. These tasks may involve understanding and analyzing information, algorithmic thinking and problem-solving, using computer systems effectively, recognizing patterns and structures, considering social and ethical issues related to technology, and solving puzzles.

The challenge is organized annually in countries all over world, and participating students solve tasks at different levels. The tasks are carefully crafted by teams of educators, researchers, and professionals to ensure their relevance, educational value, and suitability for the target age groups. The tasks are usually based on real-life scenarios or practical situations that require computational thinking skills to solve. Participating in the Bebras challenge offers students an opportunity to develop their computational thinking abilities, improve their problem-solving skills, and gain exposure to various aspects of informatics. It also encourages collaboration, critical thinking, and a deeper understanding of how technology impacts our daily lives. Thus, the Bebras challenge serves as a platform to engage students in computational thinking and foster their interest in informatics, laying the foundation for future studies and careers in computer science and related fields.

1 Introduction

Seven years ago, I published an article "Bringing Informatics Concepts to Children through Solving Short Tasks" about the Bebras challenge in the Bulletin of the EATS [2]. Since then, Bebras challenge has doubled in size: the number of countries now stands at 78, with more than 3 million students participating; see Figure 1.

The Bebras community consists of full members (55 countries) and provisional members (23 countries). Each year new provisional countries are applying and some of them get permission to enter (fulfilling required criteria), e.g., four countries (Azerbaijan, Paraguay, Peru, and Puerto Rico) have qualified to join the challenge this year. The provisional members need to establish the Bebras challenge in their countries by forming networks with schools, involving teachers, translating and adapting Bebras tasks, and promoting informatics education.

The main time of the Bebras challenge is the second week of November each year. Every Bebras member country plans a competition, training, and activities. Several countries in the Southern Hemisphere (Australia, Cambodia, Malaysia, New Zealand, Singapore, and South Korea) hold the main competition in March, when the school year starts, but they usually hold additional rounds in November.

In 2003, the idea of the Bebras competition was proposed. "Bebras" is Lithuanian word for "beaver," a hard-working, intelligent, goal seeking, and lively animal. In the past years, the number of Bebras participants has been notably growing. Over 3 million students from over 70 countries were involved in solving Bebras tasks world-wide each year. Slovenia had the strongest relative participation with over 30 000 students, whereas France had the highest total number of participants, nearly 0.7 million; see Figure 2.

The BETT (British Education and Training Technology), the largest education and technology event for 37 years, took place from March 29–31, 2023. The Bebras challenge was presented as one of the initiatives of Lithuanian researchers; see Figure 3.

In preparation for the BETT exhibition, a team from Vilnius University created a presentation of visual materials such as flyers, task cards and special bookmarks with short tasks; see Figure 4. The app created especially for the exhibition was particularly successful, as it allowed users to solve 10 Bebras tasks and win prizes.

Bebras is not only a contest, but also a platform for learning and discovery, allowing students to develop their problem-solving skills and deepen their knowledge in the field of informatics. This is not only beneficial for students, but also for teachers who can use Bebras tasks and resources in their teaching.

The aim of the Bebras challenge is to stimulate students' interest in computer science, to develop a deeper understanding of technology, to encourage the ability to solve algorithmic and logical problems, to develop critical thinking, programming

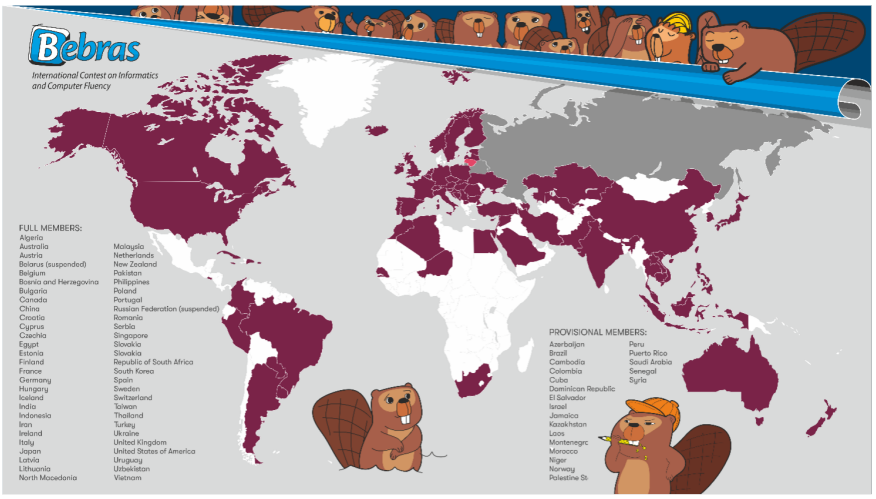


Figure 1: The Bebras challenge covers 78 countries

and computer literacy skills, and to attract more talented young people to study computer science.

Students participating in the Bebras challenge are given a wide variety of tasks, each tailored to their age group and level of logical reasoning. The appeal of Bebras tasks comes from their complexity and variety. The challenges are often complex but very interesting and require not only logical thinking but also creativity. In addition, the tasks often involve the application of various concepts and ideas that are relevant in real-world informatics. All these elements help to increase the attractiveness and relevance of the tasks, as well as to stimulate students' interest and encourage the development of computational thinking and other skills.

The competition gives students the opportunity to test their skills and creativity, as well as to expand their knowledge in the field of computer science.

The famous Finnish educator Pasi Sahlberg has highlighted the significance of playful learning, games, and gamification as factors contributing to the success of Finnish education [9]. Playful learning activities have the ability to capture children's attention and engage them in various subjects. The combination of the joy of discovery and unexpected solutions is a hallmark of such activities.

Bebras is an international initiative aiming to promote informatics among school students of all ages. The challenge is organized annually by each participating country locally. Participants are usually supervised by teachers who may integrate the Bebras tasks into their teaching activities. For running the challenge,

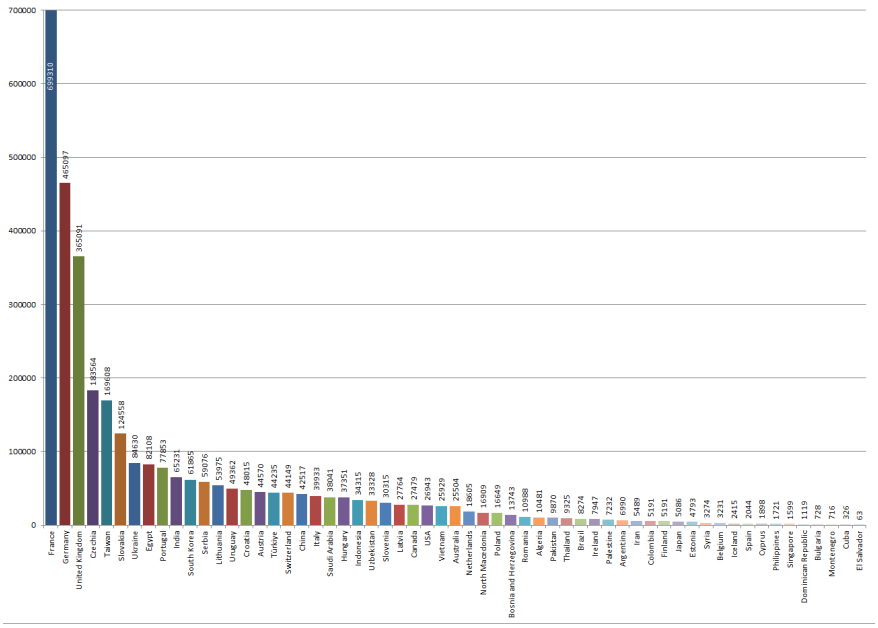


Figure 2: Number of participants in the Bebras Challenge during November 2022 and April 2023

countries use different technologies mainly based on online contest management systems (CMS). Each country chooses tasks from the Bebras task pool approved by the annually-organized international Bebras task workshop.

2 Bebras Tasks

The essence of the Bebras challenge lies in informatics concept-based tasks [1, 8]. Developing a challenging set of tasks is crucial for the success of the challenge. Task developers strive to choose interesting problems that motivate students to engage with informatics and think deeply about its core concepts. There is a need for consensus on task development criteria. Initially, six task topics were proposed: Information comprehension, algorithmic thinking, using computer systems, structures, patterns and arrangements, social, ethical, cultural, international, and legal issues, as well as puzzles [3, 10]. In recent years, a two-dimensional system for categorizing tasks has been elaborated, incorporating both informatics concepts and computational thinking skills [6].

Figure 3: At BETT exhibition: Lithuanian EdTech presents Bebras challenge

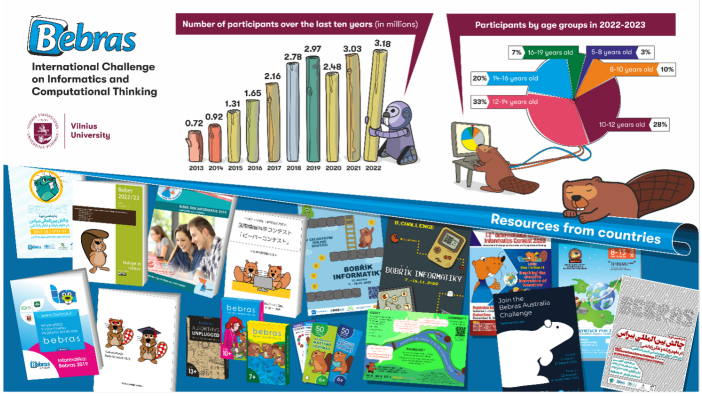


Figure 4: Variety of Bebras didactical material for teaching informatics

An annual international Bebras workshop is organized in different countries, focusing on the creation of concept-based tasks for students of all age groups. The primary objective of these workshops is to develop a set of tasks for the upcoming challenge, facilitate discussions among countries with diverse curricula

and teaching traditions, and reach a consensus on the task selection.

The challenge incorporates various types of tasks to engage participants, including interactive (dynamic) tasks, open-ended tasks, and multiple-choice tasks. The emphasis is on creating context-rich and powerful tasks that motivate and captivate students, encouraging them to delve deeper into informatics concepts. The development and introduction of such tasks pose significant challenges for researchers and educators [4, 7, 11].

Multiple-choice tasks typically feature four distinct and well-defined answer choices, with only one correct solution. Interactive tasks, on the other hand, involve a two-way transfer of information between the user and the computer. These tasks provide a problem specification, requiring students to interact directly with the computer by performing actions such as dragging and dropping objects, clicking on specific areas of pictures, manipulating objects using a keyboard, or selecting elements from a list.

Numerous countries have established networks and teams comprising researchers, teachers, and educators dedicated to the creation and discussion of Bebras tasks. These teams consistently propose new tasks each year. In the following section, four examples of Bebras tasks will be provided and discussed.

Solving short concept-based tasks is a powerful method that can support a pedagogical shift in the classroom and foster pupils' engagement and motivation to learn. Many publications deal with problem-solving methods. Solving short tasks can be one of the strategies that engage and motivate students for deeper learning and foster deeper thinking skills.

The developers of Bebras tasks are seeking to choose interesting tasks (problems) to motivate students to deal with informatics and to think deeper about technology. Also they want to cover as many informatics and computer literacy topics as possible. In informatics, there is also the problem of syllabus. Even if there is an education standard for informatics at school in some countries, until now there is no common agreement on what should be included in an integrated syllabus [5].

Example 1. Beavers vs. Kangaroos (Lithuania 2020, medium for 14–16 years)

While crossing a swamp by using a log path, five beavers meet a group of kangaroos going into the opposite direction; see Figure 5. Nobody wants to become wet or dirty so they stay on the path. The Kangaroos found out that from one specific log it is possible to jump onto a stone next to the log path and jump back to that one log. However, only one kangaroo can stand on the stone at a time.

The kangaroos and beavers don't mind going all the way back, except for Fred,

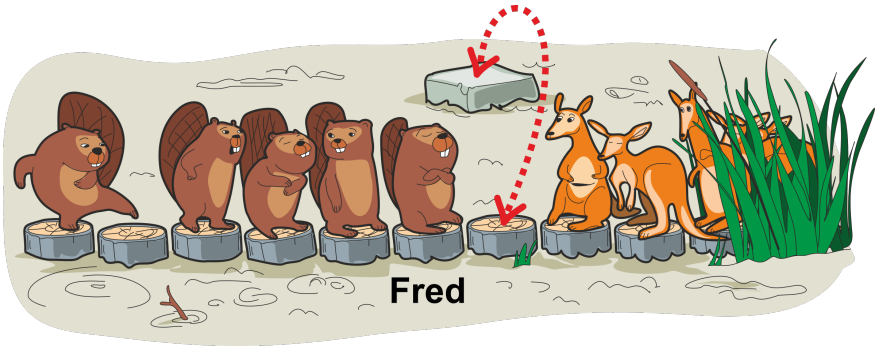


Figure 5

the leading beaver, who is the first to meet the kangaroos. Fred only wants to take a step back 10 times. With Fred's behaving this way, how many kangaroos can pass him without taking a step back?

This task introduces the design of an algorithm as a main concept including a sequence of operations (steps) and repetition, also understanding the concept of variables. Recognition of patterns in algorithms (similar steps that are repeated) can be turned into reusable code for a quick and automatic solution of a problem (as the formulation here). The logs and the stone are like registers in a processor or on a tape drive that can store data.

Example 2: Stickers (Finland 2020, medium for 16–19 years)

Betty Beaver is playing with four kinds of stickers that contain the words `ABBA`, `GAGA`, `GIBB`, and `IGGY`. She creates a word by using the stickers on an empty piece of paper. When a sticker is used at some position, it covers four characters starting from this position; see Figure 6. Betty has many stickers of each kind.

For example, one of the ways to create the word `GIABIGGYGA` would be to use the stickers as follows (asterisks indicate empty positions):

1. `GIBB` at position 1: `GIBB******`
2. `ABBA` at position 3: `GIABBA****`
3. `GAGA` at position 7: `GIABBAGAGA`
4. `IGGY` at position 5: `GIABIGGYGA`

Which of the following four words can be created by Betty's stickers? There may be several correct answers; find them all.

Figure 6

(A) `AGGIBBAGGAGABABGA`

(B) `AGGIBBAGAGGABABGA`

(C) `AGIBBGAGAGGYBAYBB`

(D) `AGGIBBAGAGGYBAGGY`

This is an example of a problem that can be analyzed using "backward induction" and "backtracking," which are common problem-solving methods in informatics. Backward induction is a process of reasoning backwards in time: we start from the end of a problem or situation (in this task, a created word), and try to determine an action (in this task, the use of some sticker) that leads to a feasible preceding situation. This is repeated until the initial situation (in this task, a completely empty state) is reached. In many problems, the process may have several possibilities for selecting actions, and in such cases the backward induction process may need to be applied in a backtracking manner: if the currently selected sequence of actions fails to reach the initial situation, then we may change some previous action selection and try again to proceed towards the initial situation.

Example 3. Strawberry Thief (Switzerland 2021, medium for 8–10 years)

Anja is playing outdoors and makes a design on the ground using four types of objects: acorns, hazelnuts, stones, and strawberries. She then adds sticks to her design according to her **Very Important Rule**:

A stick can go between two objects only if they are of different types.

Anja's completed design is shown in Figure 7. Anja's sister Zoë sees the design and eats the strawberry. To hide what she has done, she tries to replace

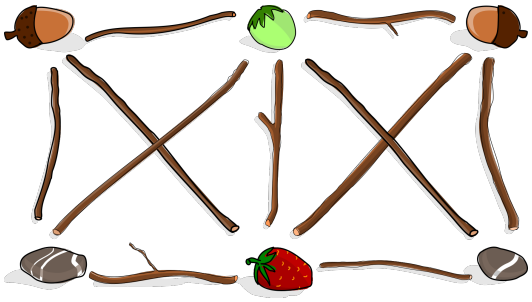


Figure 7

the strawberry with a different type of object. Which object can Zoë replace the strawberry with, without breaking Anja's **Very Important Rule**?

(A) 

(B) 

(C) 

(D) None. Only the strawberry could go there.

The correct answer is Option D. Unfortunately, Zoë is not able to replace the strawberry with a different type of object, without breaking Anja's Very Important Rule. In Anja's original design, the strawberry had sticks between it and two acorns, two stones, and one hazelnut. Changing the strawberry to anything other than another strawberry would force a stick to exist between two objects of the same type.

Anja's design can be called a graph. The objects can be called nodes and the sticks can be called edges. In a graph, edges connect nodes. Two nodes that share an edge are called neighbors. A subset of nodes where each node is a neighbor of every other node in the subset is called a clique. Anja's design contains two cliques: the left half and the right half of the design. Now suppose you wanted to assign the nodes of a graph a color so that no edges connect two nodes of the same color. Of course, the number of colors needed to do this is at least the size of the largest clique. The largest clique in Anja's design has size four, which is one reason why Zoë could not satisfy the **Very Important Rule** using just three types of objects.

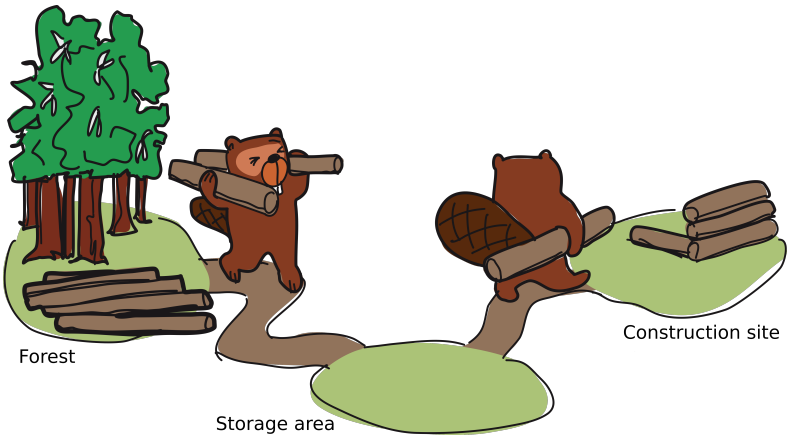


Figure 8

Example 4: Logs (Estonia 2021, medium for 14–16 years)

Jack and Sam are building a log house. Jack is bringing logs from the forest to the storage area. He can move from the forest to the storage area in 5 minutes and drag two logs at the same time. Sam is taking the logs from the storage area to the construction site. He can move from the storage area to the construction site in just 2 minutes, but only carry one log. Both beavers move at the same speed to and from the storage area with or without logs. They are working as follows:

When Jack arrives at the storage area with new logs, he will drop the logs and call out to Sam before returning to the forest; Sam will then stop working at the construction site and take the logs from the storage area.

When Sam takes the last log from the storage area and returns to the house, he will resume doing his work at the construction site; but if there are logs left at the storage area, Sam will drop the log at the house and immediately return to the storage area for more logs.

How many logs will be at most at the construction site 30 minutes after the friends start working? The way the two friends are working is similar to the producer-consumer model of parallel processing in computers. Jack is the producer of logs for Sam, and Sam is the consumer of the logs that Jack has produced.

The storage area acts as a buffer so that Jack does not have to wait until Sam comes to collect the logs; instead, Jack can return to the forest for the next pair of logs immediately and be more productive. Jack calling out to Sam when he adds new logs to the empty storage area is like the signals used in computer systems to allow one program to alert another. This lets Sam do other work instead of just

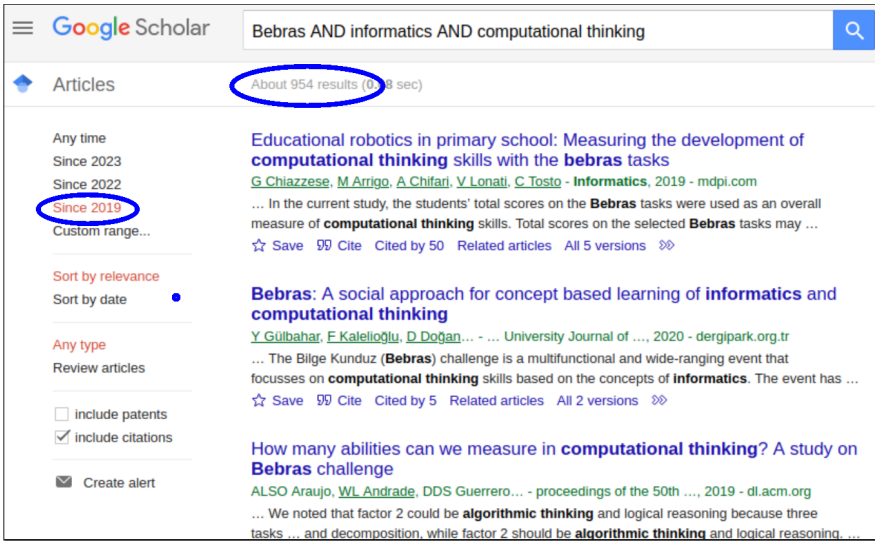


Figure 9: According to Google Scholar, 954 papers from January 2019 to May 2023 were indicated

waiting at the storage area. However, when Jack does call Sam, it takes some time for Sam to go from the construction site to the storage area, causing latency in the movement of logs. A difference of our task from the classical producer-consumer model is that in our case all the logs are considered equal and it is not required for Sam to bring the logs to the construction site in the same order as Jack collected them in the forest.

Tasks are very important both for competitors (students) and task developers (teachers): students have been "pushed" to think on computer science, educators should think about harmonization of syllabus of computer science. Creative, interesting tasks are the main drive for the Bebras contests.

3 Research in Connection with Bebras Activities

The annual Bebras challenge provides a lot of data for making inquiries on how students accept informatics concepts, how they develop computational and algorithmic thinking, what types of tasks help attract and motivate them for further involvement, etc. Some countries started to develop research papers year by year. Other countries have published overviews of tasks with detailed explanations on

Figure 10: Research papers collected at the website `https://bebras.org`

how to solve the tasks and what concepts are behind them. There are articles to promote the Bebras challenge in particular countries, and also articles dealing with particular contest results; Figure 9. The Bebras community collects articles and publishes their list on the Bebras website annually;[1] see Figure 10.

4 Conclusion

Finding the right balance between continuity and innovation is crucial in informatics education. While continuity may seem monotonous, it provides a sense of stability and comfort in our daily lives. Similarly, in teaching informatics, maintaining a consistent framework of lessons, problem-solving exercises, and core concepts can create a comfortable learning environment. Additionally, occasional contests can serve as motivating factors for students.

However, the most significant aspect of introducing informatics in schools is the human connection. Amidst long hours at work and moments of uncertainty, receiving messages from others who share the same interests and struggles can be

[1] https://www.bebras.org/publications.html

incredibly impactful. It is essential to foster a culture of dreaming, searching, and communication, both in everyday life and within informatics education.

To engage students and recognize informatics as a scientific discipline, we should strive for a more successful involvement. Well-organized activities with intriguing and exciting tasks can immerse students in the world of informatics, helping them grasp the core concepts and develop a genuine interest.

References

[1] Carlo Bellettini, Violetta Lonati, Dario Malchiodi, Mattia Monga, Anna Morpurgo, and Mauro Torelli. How challenging are bebras tasks? An IRT analysis based on the performance of Italian students. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2015)*, pages 27–32, ACM, 2015.

[2] Valentina Dagienė. Bringing informatics concepts to children through solving short tasks. *Bulletin of EATCS*, 1(118), 2016.

[3] Valentina Dagienė and Gerald Futschek. Bebras International Contest on Informatics and computer literacy: criteria for good tasks. In *Proceedings of the 3rd International Conference on Informatics in Secondary Schools - Evolution and Perspectives (ISSEP 2008)*, LNCS 5090, pages 19–30. Springer, 2008.

[4] Valentina Dagienė, Gerald Futschek, and Gabrielė Stupurienė. Creativity in solving short tasks for learning computational thinking. *Constructivist Foundations* 14, pages 382–396, 2019.

[5] Valentina Dagienė, Juraj Hromkovič, and Regula Lacher. Designing informatics curriculum for K-12 education: From concepts to implementations. *Informatics in Education* 20(3), pages 333–360, 2021.

[6] Valentina Dagienė, Sue Sentence, and Gabrielė Stupurienė. Developing a two-dimensional categorization system for educational tasks in informatics. *Informatica* 28(1), 23–24, 2017.

[7] Violetta Lonati. Getting inspired by Bebras tasks. How Italian teachers elaborate on computing topics. *Informatics in Education* 19(4), pages 669–699, 2020.

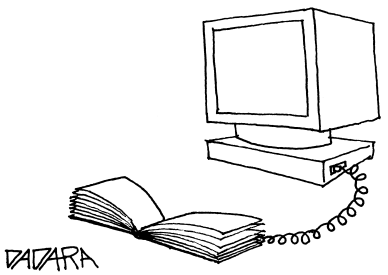
[8] Wolfgang Pohl and Hans-Werner Hein. Aspects of quality in the presentation of informatics challenge tasks. In *Proceedings of the 3rd International Conference on Informatics in Secondary Schools - Evolution and Perspectives (ISSEP 2015)*, LNCS 9378, pages 21–32. Springer, 2015.

[9] Pasi Sahlberg. Finnish lessons 2.0: What can the world learn from educational change in Finland? Teachers College, Columbia University, 2015.

[10] Jiří Vaníček. Bebras informatics contest: Criteria for good tasks revised. In *Proceedings of the 7th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (ISSEP 2014)*, LNCS 8730, pages 17–28. Springer, 2014.

[11] Jiri Vaníček and Vaclav Šimandl. Participants' perception of tasks in an informatics contest. In *Proceedings of the 13th International Conference on Informatics in Schools (ISSEP 2020)*, LNCS 12518, pages 55–65, Springer, 2020.

News and Conference Reports



Report from EATCS Japan Chapter

Yukiko Yamauchi (Kyushu University)

EATCS-JP/LA Workshop on TCS and Presentation Awards

The 21st *EATCS-JP/LA Workshop on Theoretical Computer Science* was held fully face-to-face at Research Institute of Mathematical Sciences, Kyoto University, January 30th to February 1st, 2023. (The details can also be found, although this website is written in Japanese, at
`https://la-symposium2022.github.io/winter_program.html`.)

Every year, we choose the best presenter and the best student presenter. This year, we celebrated the following presentation as the 21st LA/EATCS-Japan Presentation Award:

> "*Hardness Self-Amplification*", Shuichi Hirahara (National Institute of Informatics), **Nobutaka Shimizu** (Tokyo Institute of Technology)

We celebrated the following presentation as the 12th LA/EATCS-Japan Student Presentation Award:

> "*Lipschitz Continuous Algorithms for Graph Problems*", **Soh Kumabe** (The University of Tokyo), Yuichi Yoshida (National Institute of Informatics)

The awards were recognized publicly on the last day, February 1st, 2023.

Congratulations!

This workshop is jointly organized by *LA symposium*, Japanese association of theoretical computer scientists. Its purpose is to give a place to discuss topics on all aspects of theoretical computer science. This workshop is an unrefereed meeting. All submissions are accepted for the presentation. There should be no problem of presenting these papers at refereed conferences and/or journals. This meeting is unofficial, familiar, and widely open for everyone who is interested in theoretical computer science. It is held twice a year (January/February and July/August). If you have a chance, I recommend that you attend it. Check the website `http://www.ecei.tohoku.ac.jp/alg/EATCS-J/` for further details. The list of the presentations is as below; you can see the activity of the Japanese society of theoretical computer science.

Program of the 21st EATCS-JP/LA workshop on TCS (January 30th to February 1st, 2023)

In the following program, "*" indicates non-student speakers, while "**" indicates student speakers. The number [S*xx*] means it is in student session, namely, it is a shorter talk than a regular one.

[1] Groups whose Word Problem is Accepted by an Abelian G-automaton
***Takao Yuyama (Tokyo Institute of Technology)*

[2] Theoretical limits of the stochastic multiprocessor scheduling problem
***Daiki Suruga (Nagoya University)*

[3] Online Job Scheduling with k Servers
***Jiang Xuanke (Kyushu University), Sherief Hashima (RIKEN AIP), Kohei Hatano (Kyushu University / RIKEN AIP), Eiji Takimoto (Kyushu University)*

[4] Computing maximal generalized palindromes
**Mitsuru Funakoshi (Kyushu University), Takuya Mieno (University of Electro-Communications), Yuto Nakashima, Shunsuke Inenaga (Kyushu University), Hideo Bannai (Medical and Dental University), Masayuki Takeda (Kyushu University)*

[5] Reverse Engineering of Right-to-Left Position Heaps
***Koshiro Kumagai, Diptarama Hendrian, Ryo Yoshinaka, Ayumi Shinohara (Tohoku University)*

[6] Quantum Search-to-Decision Reduction for the LWE problem
***Kyohei Sudo (Osaka University), Masayuki Tezuka (National Institute of Technology, Tsuruoka College), Keisuke Hara (National Institute of Advanced Industrial Science and Technology / Yokohama National University), Yusuke Yoshida (Tokyo Institute of Technology)*

[7] Distributed Quantum Interactive Proofs: Parallelization and Application
*Francois Le Gall, **Masayuki Miyamoto, Harumichi Nishimura (Nagoya University)*

[8] Hardness Self-Amplification
*Shuichi Hirahara (National Institute of Informatics), *Nobutaka Shimizu (Tokyo Institute of Technology)*

[9] Overlapping of Lattice Unfolding for Cuboids
***Takumi Shiota (Kyushu Institute of Technology), Tonan Kamata, Ryuhei Uehara (Japan Advanced Institute of Science and Technology)*

[10] On the Security of Chameleon-Hash Functions with Ephemeral Trapdoors
***Kafu Hamada, Yoshida Yusuke, Keisuke Tanaka (Tokyo Institute of Technology)*

[11] Turedo, a novel class of Turing machines for programming RNA co-transcriptional folding
*Daria Pchelina (Université Paris 13), Nicolas Schabanel (ENS Lyon), *Shinnosuke Seki (The University of Electro-Communications), Guillaume Theyssier (Aix-Marseille Université)*

[12] Application of inside and outside judgment method in complex shape
**Satoshi Kodama (International Professional University of Technology in Tokyo)*

[13] Algorithmic Meta-Theorems for Combinatorial Reconfiguration Revisited

[S13] On a spectral lower bound of treewidth
***Kohei Noro, Tatsuya Gima (Nagoya University),Tesshu Hanaka (Kyushu University), Yota Otachi, Hirotaka Ono (Nagoya University)*
[S14] Graph Linear Notations with Regular Expressions
***Ren Mimura, Kyohei Miyabe, Kengo MiyamotoK, Akio Fujiyoshi (Ibaraki University)*
[S15] The Upper Bound on the Minimum Density for Anti-slide Packing Using 2x2x1 Pieces
***Kento Kimura, Kazuyuki Amano (Gunma University)*
[S16] Collecting Balls on a Line by Robots with Limited Energy
***Nicolas Honorato Droguett, Kazuhiro Kurita (Nagoya University),Tesshu Hanaka (Kyushu University), Yota Otachi, Hirotaka Ono (Nagoya University)*
[S17] The ultimate sign of second-order holonomic sequences
*Kawamura Akitoshi, **Hagihara Fugen (Kyoto University)*
[S18] Extension of non-deterministic ZDD by introduction of set difference operation and application for set similarity searching
***Shota Shikama (Kyushu University)*
[S19] Approximation Algorithms for Finding the Myerson Centrality of a Network
***Yuto Kuwabara, Masaaki Matsumoto, Toshinori Yamada (Saitama University)*
[S20] Structural Parameterizations of Vertex Integrity
***Ryota Murai, Tatsuya Gima (Nagoya University), Tesshu Hanaka (Kyushu University), Yasuaki Kobayashi (Hokkaido University), Hirotaka Ono, Yota Otachi (Nagoya University)*
[S21] On nonnegative k-submodular relaxation
***Kotaro Uchida, Yuni Iwamasa (Kyoto University)*
[S22] Cartesian Tree Subsequence Matching on Indeterminate Strings
***Kento Hirose (Kyushu University), Takuya Mieno (The University of Electro-Communications), Yuto Nakashima, Shunsuke Inenaga (Kyushu University)*
[S23] A fast algorithm for finding a maximal common subsequence of multiple strings
***Miyuji Hirota, Yoshifumi Sakai (Tohoku University)*
[S24] Head-or-Tail Bin-Packing Algorithms for Sorted Items
***Rina Atsumi, Hiroshi Fujiwara, Hiroaki Yamamoto (Shinshu University)*

Forthcoming Event

ISAAC 2023

International Symposium on Algorithms and Computation (ISAAC) is intended to provide a forum for researchers working on algorithms and computation. The 34th edition of this symposium will be held in Kyoto from December 3rd to 6th, 2023. In this year, all the accepted papers are expected to be presented on-site by some of the authors. See `https://www.kurims.kyoto-u.ac.jp/isaac/isaac2023/` for more information on ISAAC 2023.

Submission Deadline: June 30, 2023 (Anywhere on Earth)

Notification of Acceptance: September 4, 2023

OPODIS 2023
International Conference on Principles of Distributed Systems (OPODIS) is an open forum for the exchange of state-of-the-art knowledge concerning distributed computing and distributed computer systems. All aspects of distributed systems are within the scope of OPODIS, including theory, specification, design, performance, and system building. The 27th edition of this conference will be held in Tokyo from December 6th to 8th, 2023. See `https://opodis.net` for more information on OPODIS 2023.

WALCOM 2024
The 18th International Conference and Workshops on Algorithms and Computation (WALCOM 2024) will be held at Kanazawa, Japan from March 18th to 20th, 2024. This conference was established to encourage young researchers of theoretical computer science in Asia, especially, India and Bangladesh. Nowadays, there are many participants not only from a wide range of Asia but also from Europe and North America. See `https://www.kono.cis.iwate-u.ac.jp/~yamanaka/walcom2024/` for more information on WALCOM 2024.

New member of EATCS-J (by Ryuhei Uehara)

Professor Yukiko Yamauchi had been a secretary of the Japan chapter of EATCS for long years. Now she has retired, and Professor Yuto Nakashima has joined as a secretary. All members thank Yukiko for her kind support, and welcome Yuto to our team!

EATCS Japan Chapter

Chair:	Ryuhei Uehara
Vice Chair:	Takehiro Ito
Secretary:	Yukiko Yamauchi
email:	eatcs-jp@grp.tohoku.ac.jp
URL:	http://www.ecei.tohoku.ac.jp/alg/EATCS-J/index.html

Report on BCTCS 2023

The 39th British Colloquium for Theoretical Computer Science

3–4 April 2023, University of Glasgow

Ciaran McCreesh

The British Colloquium for Theoretical Computer Science (BCTCS) is an annual forum in which researchers in Theoretical Computer Science can meet, present research findings, and discuss developments in the field. It also provides a welcoming environment for PhD students to gain experience in presenting their work to a broader audience, and to benefit from contact with established researchers.

BCTCS 2023 was hosted by the University of Glasgow and held from 3rd to 4th April 2023. The event attracted 37 registered participants, and featured an interesting and wide-ranging programme. A total of 15 contributed talks – predominantly by PhD students – were presented at the meeting alongside the five keynote speakers. The meeting also featured a special session on the pedagogy of theoretical computer science.

BCTCS 2024 will be hosted by the University of Bath from 3rd–5th April 2024. Researchers and PhD students wishing to contribute talks concerning any aspect of Theoretical Computer Science are cordially invited to do so. Further details are available from the BCTCS website at `www.bctcs.ac.uk`.

Invited Talks

Ruth Hoffmann (University of St Andrews)
Composable Constraint Models for Permutation Patterns and their enumeration

Permutation pattern research started off as investigating which sequences of numbers can be sorted by using a stack. This has now extended into many fields such as using permutations which contain or avoid certain types of patterns when investigating for example Mahonian statistics. Constraint programming is a way of solving combinatorial problems by taking variables, the values they can take and constraints which involve the variables. It then searches for one (all, or the optimal) solution (variable, value assignments) which does not violate the constraints. We will explore different permutation patterns, properties and statistics. While giving you the many definitions we will see how each translates into a constraint model. Having these many models means that we can now easily mix and match them into useful tools to help solve or help investigate permutation problems computationally.

Steve Linton (University of St Andrews)
Three Trips Around the "Virtuous Circle": Theory, Algorithms, Software and Experiments

My thesis in this talk is that there can be a powerful synergy between the study of mathematical and combinatorial structures in the abstract; the theoretical study of algorithms for computing with those structures and their complexity; the development of flexible and usable software implementations of those algorithms; and the gathering of experimental data using that software, which can fuel new conjectures and lead to new mathematical results, completing the circle. I will illustrate this thesis with examples from three areas: permutation groups; transformation monoids; and token-passing networks and pattern classes of permutations.

David Manlove (University of Glasgow)
Models and Algorithms for the Kidney Exchange Problem

A patient who requires a kidney transplant, and who has a willing but incompatible donor, may be able to 'swap' his or her donor with that of another patient, who is in a similar situation, in a cyclic fashion. Altruistic donors can also trigger "chains" of transplants involving multiple recipients together with their willing but incompatible donors. Kidney exchange programmes (KEPs) organise the systemic detection of optimal sets of cycles and chains based on their pools of donors and recipients. There are many examples of KEPs around the world, including the UK Living Kidney Sharing Scheme (UKLKSS). In this talk I will describe integer programming models and algorithms that can be used to solve the underlying optimisation problem involved in a KEP. This includes the algorithms developed at Glasgow that have been used by NHS Blood and Transplant for the UKLKSS since 2008.

Faron Moller (Swansea University)
Technocamps: Transforming Digital Education Throughout Wales

By 2000, it became evident that, in Wales, interest in, knowledge of, and capacity for computing was not keeping pace with the transformational rise of the digital society and economy. Technocamps, the pan-Wales school and community outreach unit established at Swansea University but with a hub in every university in Wales, has throughout this time researched, championed and delivered change in national curricula, qualifications, delivery and professional development in order to foster a sustainable digital skills pipeline in Wales. In this presentation, we highlight the activities and impact of Technocamps, showcasing its wider impact on computing education, practitioners, schools, and learners in Wales, especially with the introduction of the new Curriculum for Wales in September 2022, with its major reform of computer science and cross-curricular digital competencies.

Syed Waqar Nabi (University of Glasgow)
Navigating Pedagogies: Teaching Theory-Heavy Courses to Software Engineering Student

The landscape of teaching pedagogies is a rich one, and not always easy to navigate. While there has been a shift towards student-centered, "constructivist" approaches to teaching, the more traditional teacher-centred approaches like direct-instruction are still prevalent, more so in theory-heavy courses. For this talk, I will use theory-heavy courses I teach to Software Engineering Graduate Apprenticeship students as conduits for exploring this pedagogy landscape, discussing experiences with using a number of teaching instruments along the way. This will lead to the specific pedagogy that I have been converging on called "Competency-Based Education" (CBE), similar to what's called "mastery learning". Based on the outcome of a working group on CBE, I will go a bit more into what CBE is, how it relates to some other pedagogies, how we can draw inspiration from other teaching domains, and what it might mean to use it for computing education. Finally, I will connect this discussion to theory-heavy computing science courses in general, and share some thoughts on how (or if) CBE can work for such courses.

Contributed Talks

Andrew Ryzhikov (University of Oxford)
On Cost Register Automata with Few Registers

Cost register automata (CRA) are an extension of deterministic finite state automata. Instead of accepting or rejecting words, they assign each word a value (which can mean, for example, a cost, probability or duration of an event). This value is computed with a finite set of write-only registers which are updated every time a transition is taken. CRA are tightly related to weighted automata (WA), and natural syntactic restrictions for CRA allow to define new subclasses of functions which are not definable in terms of WA. One such restriction is to bound the number of registers. We show that for CRA with only three registers universality (are the values of all words below/above a certain threshold?) remains undecidable both over the tropical semiring and over the semiring of rational numbers with usual addition and multiplication. In contrast, we show that the zeroness problem (does there exist a word of value zero?) for CRA over the tropical semiring becomes solvable in polynomial time if the number of registers is constant, while it is PSPACE-complete without this assumption.

This is a joint work with Laure Daviaud (City, University of London).

Peter Strulo (University of Warwick)
An Exercise in Tournament Design: When Some Matches Must Be Scheduled

In single-elimination tournaments, players play one-on-one matches with the winner proceeding to the next round until only one player remains. The problem of manipulating the outcome of the tournament by carefully choosing which opponents play each other in each round (the seeding) has been studied extensively. We introduce a new variant of this problem where the aim is to choose a seeding which results in certain desired matches being played, rather than a specific player winning. We obtain both hardness and tractability results: the problem is NP-hard in general but polynomial-time solvable when the input digraph modelling the pairwise results is acyclic.

Marcel De Sena Dall'Agnol (University of Warwick)
Streaming zero-knowledge proofs

We initiate the study of zero-knowledge proofs for data streams. Streaming interactive proofs (SIPs) are well-studied protocols whereby a space-bounded algorithm with one-pass sequential access to a massive stream of data communicates with an all-powerful but untrusted prover to verify a computation that requires large space.

We define the notion of zero-knowledge in the streaming setting and construct zero-knowledge SIPs for the two main building blocks in the streaming interactive proofs literature: the sumcheck and polynomial evaluation protocols. To the best of our knowledge all known streaming interactive proofs are based on either of these tools, and indeed, this allows us to obtain zero-knowledge SIPs for central and well-studied streaming problems, such as index, frequency moments, and inner product. Our protocols are efficient both in terms of time and space, as well as communication: the space complexity is polylog(n) and, after a non-interactive setup that uses a random string of near-linear length, the remaining parameters are sub-polynomial.

En route, we develop a toolkit for designing zero knowledge data stream protocols that may be of independent interest, consisting of an algebraic streaming commitment protocol and a temporal commitment protocol. The analysis of our protocols relies on delicate algebraic and information-theoretic arguments and reductions from average-case communication complexity.

Bruno Pasqualotto Cavalar (University of Warwick)
Constant-Depth Circuits vs. Monotone Circuits

We establish strong separations between the power of monotone and general (non-monotone) Boolean circuits:

- For every $k \geq 1$, there is a monotone function in AC^0 (constant-depth polysize circuits) that requires monotone circuits of depth $\Omega(\log^k n)$. This vastly extends a classical result of Okol'nishnikova (1982) and Ajtai and Gurevich

(1987). Our separation holds for a monotone graph property, which was unknown even in the context of AC^0 versus mAC^0.

- For every $k \geq 1$, there is a monotone function in $\mathsf{AC}^0[\oplus]$ (constant-depth poly-size circuits extended with parity gates) that requires monotone circuits of size $\exp(\Omega(\log^k n))$. This makes progress towards a question posed by Grigni and Sipser (1992).

These results show that constant-depth circuits can be considerably more efficient than monotone circuits when computing monotone functions.

In the opposite direction, we observe that non-trivial simulations are possible in the absence of parity gates: every monotone function computed by an AC^0 circuit of size s and depth d can be computed by a monotone circuit of size $2^{n-n/O(\log s)^{d-1}}$. We show that the existence of significantly stronger monotone simulations would lead to breakthrough circuit lower bounds. In particular, if every monotone function in AC^0 admits a polynomial size monotone circuit, then NC^2 is not contained in NC^1.

Finally, we revisit our separation result against monotone circuit size and investigate the limits of our approach, which is based on a monotone lower bound for constraint satisfaction problems established by Göös et al. (2019) via lifting techniques. Adapting results of Schaefer (1978) and Allender et al. (2009), we obtain a classification of the monotone circuit complexity of Boolean-valued CSPs via their polymorphisms. This result and the consequences we derive from it might be of independent interest.

Nathan Flaherty (University of Liverpool)

On Transposition Distance of Words with Fixed Parikh Vectors

The operation of transposition is a permutation that swaps any two symbols in a word. The Parikh Vector P denotes the number of occurrences of each letter in a given word and the operation of transposition preserves its Parikh Vector. We consider the configuration graph where the set of vertices are words with the same Parikh Vector and edges are defined by transposition operations on these words. The question about the maximal shortest path between two words by transposition corresponds to the estimation of the diameter D in a configuration graph. We show the tight bound on the diameter D which is equal to $n - \max_{i\in[q]} P_i$ where q is the size of a finite alphabet, and $n = \sum_{i\in[q]} P_i$ is the length of considered words. The lower bound is based on the analysis of cyclic covers of auxiliary graph structure and the matching upper bound follows from the direct proof of algorithmic transformation. This is the joint work with Duncan Adamson, Igor Potapov and Paul Spirakis.

Ben Lloyd-Roberts (Swansea University)

Mining Invariants from State Space Observations

The application of model checking to verify railway signalling systems has a long history within academia and is beginning to seen some real applications in the railway sector. One limitation of such model checking is that verification can fail due to over approximation, typically when using techniques such as inductive verification. Here, one solution is to introduce so-called invariants, formal properties satisfied by all states, to suppress false positives. However, automatically generating sufficiently strong invariants to help bound the region of reachable states is complex. In this work, we show it is possible to use machine learning to generate candidate invariants for model checking. Our methodology starts by providing a first formal mapping of state spaces to a reinforcement learning (RL) environment. We then train agents to explore large regions of states spaces while building a dataset of unique state observations. Finally we demonstrate that statistical analysis of state observations gives rise to interesting correlations between variables, allowing proposals for candidate invariant properties.

Matthew McIlree (University of Glasgow)

How can a constraint solver prove it is telling the truth?

A proof log for a problem-solving algorithm provides a verifiable certificate that the result is correct, and also an auditable record of the steps taken to obtain that result. In the field of Boolean satisfiability, proof-logging has become an expected capability of modern solvers, with a standard proof format called DRAT widely accepted for independent verification. In contrast, a similar standard practice has not yet been adopted for Constraint Programming (CP), due to the difficulties of applying DRAT to the more expressive formulations and reasoning present in this paradigm. However, recent work towards "An Auditable Constraint Programming Solver" (Gocht et al. 2022) has shown how a proof system working in a pseudo-Boolean format can certify the reasoning carried out for several important expressive global constraints, offering a strong candidate for a complete, general CP proof logging method. This talk will be an introduction to proof logging in the context of constraint programming. It will summarise the main motivations; the core techniques developed so far; and the reasons for being optimistic about the applicability of the method going forward.

Laura Larios-Jones (University of Glasgow)

Minimising temporal reachability in graphs with uncertainty

Temporal graphs consist of an underlying graph and an assignment of timesteps to edges that specifies when each edge is active. This allows us to model spread through a network which is time-sensitive. Previous work has explored minimising spread by edge deletion for applications such as epidemiology. In reality,

these models cannot be exact. This motivates the introduction of uncertainty to the problem. Our goal is to remove a set of edges in our graph such that the maximum number of vertices reachable from any starting vertex is minimised even when there is uncertainty in our input. We will discuss some preliminary analogous structural and algorithmic results.

Fabricio Mendoza Granada (University of Glasgow)
On finding the b-chromatic number of a tree

Graph colouring is an extensively studied problem in computer science, discrete mathematics and other disciplines. It involves assigning colours or labels to the vertices so that not two adjacent vertices share the same colour. This assignment is called a proper colouring. The problem was originally proposed as a puzzle to colour the map of counties in England in 1878. Its applications arise in the context of scheduling, timetable construction, register allocation and many others. The first graph colouring parameter to be studied was the chromatic number of a graph G, $\chi(G)$, which is the minimum number of colours used by a proper colouring of G. In this talk we will discuss the b-chromatic number $\varphi(G)$ of a graph G, a concept introduced by Irving and Manlove in 1998. The b-chromatic number of a graph is the maximum integer k for which G admits a proper colouring such that for every colour c there exists a vertex v of colour c that is adjacent to at least one vertex of every other colour. Deciding whether $\varphi(G) \geq k$ for a given graph G and integer k was proved to be NP-complete by Irving and Manlove, and this holds even for bipartite graphs. However, they proved that the b-chromatic number of a tree can be computed in polynomial time by describing an algorithm to find a b-chromatic colouring using $\varphi(G)$ colours. In this talk we will present the algorithm for finding a b-chromatic number of a tree in pseudocode form. Futhermore, we show that the algorithm runs in linear time; previously it was only claimed that the algorithm runs in polynomial time. Finally, we will present some experimental results on families of random trees.

Filippos Pantekis (Swansea University)
GPGPUs, Supercomputers, and a Game of Chess

The evolution of General Purpose Graphics Processing Unit (GPGPU) devices, paired with their wide commercial availability, has enabled a broader spectrum of problems to benefit from the superior mathematical capabilities offered by this hardware. Perhaps the biggest obstacle in using GPGPUs to accelerate the solving for all problems, is the restrictive computation flow (regularity) expected by the hardware in order to maximise performance. This talk presents how certain algorithmic choices together with hardware-specific optimisations can transform an irregular algorithm for the N-Queens problem to a competitive solver.

Carlos A. Perez Delgado (University of Kent)
Towards a Physical Fundamental Computational Complexity Theory

In theoretical computer science, the fundamental yardstick of computational cost is left largely undefined. The (time) cost of performing a computation/algorithm is measured in its number of "primitive operations". However, one is allowed to choose the set of primitive operations at will. Big "Oh" notation allows one to do so, while retaining a consistent measure. This allows for an elegantly simple theory that can nevertheless make meaningful statements about algorithms.

The theory is not without its flaws, however. The first is the already mentioned arbitrariness of the yardstick(s). Second, it fails to say anything meaningful when attempting to compare different architectures, or comparing algorithms across them. For instance, computations running on massively parallel architectures, quantum computers, and/or single-core processors cannot be meaningfully compared with one another without introducing extra assumptions.

In this talk I will propose a fundamental theory of computational complexity. This theory uses the physical resource action (that is energy in joules times time in seconds), as the fundamental unit of computation. We will introduce a model of computation that allows us apply this metric, much like Turing machines can be used for (traditional) computational complexity cost. We will discuss how to recover all existing results from computational complexity, and we will discuss benefits of this model in terms of meaningful comparisons that traditional complexity theory cannot make.

Peace Ayegba (University of Glasgow)
Resolving the complexity of variants of stable matching problems.

Matching problems involve the allocation of one set of agents to another set of agents based on preferences, with application in various real-world centralised matching schemes. A common objective is to find a stable matching where no set of agents would rather be matched together than with their current assignment. It is well known that finding a maximum cardinality stable matching for several matching problems such as the Stable Marriage problem with Ties and Incomplete lists (MAX-SMTI), is NP-hard, even with strong restrictions on the input. However, a polynomial-time algorithm exists for a restricted version of MAX-SMTI, where each man's list is of length at most 2 and each woman's list can be of unbounded length. This talk resolves the complexity of other maximum cardinality stable matching problems (e.g., in the context of hospitals-residents with ties, and student-project allocation) under strong restrictions on the input.

Xin Ye (Durham University)
Computing Balanced Solutions for Large International Kidney Exchange Schemes

To overcome incompatibility issues, kidney patients may swap their donors. In international kidney exchange programmes (IKEPs), countries merge their national patient-donor pools. We consider a recently introduced credit system. In each round, countries are given an initial "fair" allocation of the total number of kidney transplants. This allocation is adjusted by a credit function yielding a target allocation. The goal is to find a solution that approaches the target allocation as closely as possible, to ensure long-term stability of the international pool. As solutions, we use maximum matchings that lexicographically minimize the country deviations from the target allocation. We first introduce a novel approach for incorporating credits that has not been proposed in the literature before. Namely, let the solution concepts prescribe a set of target allocations for a credit-adjusted game, where the credits are incorporated into the value function of the game directly. We perform a computational study for a large number of countries, up to fifteen countries. For the initial allocations we extend by also considering the tau value and Banzhaf value, and compare them to previously obtained results, namely the benefit value, contribution value, Shapley value and nucleolus. Our experiments show that using lexicographically minimal maximum matchings instead of ones that only minimize the largest deviation from the target allocation (as previously done) may make an IKEP up to 54% more balanced. This is joint work with Marton Benedek, Peter Biro and Daniel Paulusma.

David Kutner (Durham University)
The TaRDiS and epidemics in temporal graphs

We are interested in the resilience to infection of a population of n individuals who will interact m times, with k of those individuals being initially infectious. In the worst case, the entire population is infected once all the interactions have occured; and this is necessarily the case $k = n$. Our question is then to find the size of the smallest set of infectious individuals which would still infect the entire population.

Temporal graphs (graphs which change over time) offer us a convenient model for this problem. In our case, vertices corresponding to individuals remain constant and edges, each corresponding to an interaction, appear at exactly one (unique) time. We denote this temporal graph $G = (V, E), \lambda : E \to [|E|]$, and say a node v_0 reaches another node v_l (denoted by $v_0 \rightsquigarrow v_l$) if there is a static path $v_0, v_1, \ldots, v_l \in G$ and $\lambda(v_i, v_{i+1})\ \forall i \in \{0, l-1\}$.

Then our problem can be formalized as follows: given a temporal graph G, λ, and integer k, is there a set of vertices $S \subseteq V(G)$ such that for every $v \in V(G)$ either $v \in S$ or $\exists u \in S : u \rightsquigarrow v$? We call this problem Temporal Reachability Dominating Set, or TaRDiS, and present hardness and tractability results for it.

Thomas Karam (University of Oxford)

Lower-order ranks and the structure of the ranges of boolean polynomials on finite prime fields

Let p be a prime integer, and let $1 \leq d < p$ be a positive integer. The *degree-d rank* of a polynomial $P : \mathbb{F}_p^n \to \mathbb{F}_p$ was defined in 2007 by Green and Tao as the smallest nonnegative integer k such that we can find polynomials $P_1, \ldots, P_k : \mathbb{F}_p^n \to \mathbb{F}_p$ with degree at most d and a function $F : \mathbb{F}_p^k \to \mathbb{F}_p$ satisfying $P = F(P_1, \ldots, P_k)$.

As shown by Green and Tao, if $2 \leq d < p$ and P is a degree-d polynomial not approximately uniformly distributed on $\mathbb{F}_p^n$, then P must have bounded degree-$(d-1)$ rank. The literature after that has largely focused on the equidistribution of polynomials, and hence on the notion of degree-$(d-1)$ rank.

Recently, Gowers and the speaker showed that this statement could be extended to boolean polynomials: if P is not approximately uniformly distributed on $\{0,1\}^n$, then P coincides on $\{0,1\}^n$ with a polynomial that has bounded degree-$(d-1)$ rank.

In this talk I shall explain how this extension, which is still based purely on the degree-$(d-1)$ rank, may be used to deduce a description of the range of a polynomial on S^n which uses the other ranks defined by Green and Tao.

Scientific colloquium in honor of former EATCS President Burkhard Monien on the occasion of his 80th birthday

Ulf-Peter Schroeder
Paderborn University
ups@upb.de

On May 5, 2023, a colloquium was held at Paderborn University in honor of Burkhard Monien on the occasion of his 80th birthday. The colloquium paid extensive tribute to Burkhard's approximately 50 years of work and scientific contributions to the theoretical computer science community in general and to Paderborn University in particular. Among the approximately 90 invited guests were many long-time scientific colleagues, his scientific descendants, and also numerous representatives of various national and international scientific organizations (EATCS, GI, DFG, Academy of Sciences). The lecture program was opened by the current Vice Dean of the Paderborn Faculty of Computer Science, Christian Scheideler, who incidentally also succeeded Burkhard on the Chair of Theoretical Computer Science at Paderborn University. In addition to a personal review by Burkhard Monien, the program then included talks on his long-time research interests (Complexity Theory, Parallel Computing, and Algorithmic Game Theory), by scientific descendants (Ewald Speckenmeyer, Oliver Vornberger, and Christian Plessl) and by long-time colleagues (I. Hal Sudborough and Marios Mavronicolas).

As a service to EATCS, Burkhard Monien has held almost every position during his long career. He has been a long-time Secretary, Council member, Vice President, and President of EATCS. Among his more than 200 papers, this includes 12 ICALP papers and two proceedings volumes of ICALP as PC-chair, 1991 and 1996. For his scientific contributions, Burkhard Monien received, among others, the Leibniz Prize of the DFG, various memberships and awards of scientific academies as well as the Test-of-Time Award of the conference series *Workshop on Graph-Theoretic Concepts in Computer Science*. The traces of his scientific work at the computer science location of Paderborn University cannot be enumerated within the scope of this report, so it should only be recalled here that Burkhard was the first appointed computer science professor at Paderborn Uni-

versity in 1977, at that time entrusted with the establishment of the new subject of computer science. In the following years, he was the founder of important scientific institutions at Paderborn University, such as the *Paderborn Center for Parallel Computing* (PC^2) and the *Heinz Nixdorf Institute (HNI)* both of which have since then achieved great national and international reputation.
Incidentally, Burkhard Monien will chair the *Golden Anniversary Session* at the 50th ICALP to be held in Paderborn from July 10-14, 2023. For this, two outstanding keynote speakers, Kurt Mehlhorn and Thomas Henzinger, have been invited. Don't miss this event and register for ICALP 2023!

The Vice Dean of the Faculty of Computer Science at Paderborn University presents the jubilarian with a jersey of his favorite soccer club HSV with the number 80 on the back.

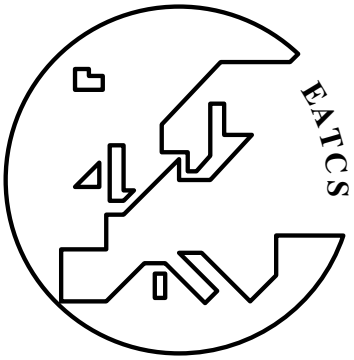
Burkhard Monien presents a personal retrospective of his approximately 50 years as researcher at Paderborn University.

The audience fully enjoyed the invited research talks as well as the *spiritual excursion* by Burkhard Monien on his long-life research career.

Group picture of the speakers at the colloquium: Christian Scheideler (Paderborn University), Ewald Speckenmeyer (University of Cologne), Oliver Vornberger (Osnabrück University), Burkhard Monien (Emeritus of Paderborn University), I. Hal Sudborough (Emeritus of University of Texas at Dallas), Christian Plessl (Paderborn Center for Parallel Computing), Marios Mavronicolas (University of Cyprus).

The Bulletin of the EATCS

European Association for Theoretical Computer Science



E A T C S

201

EATCS

HISTORY AND ORGANIZATION

EATCS is an international organization founded in 1972. Its aim is to facilitate the exchange of ideas and results among theoretical computer scientists as well as to stimulate cooperation between the theoretical and the practical community in computer science.

Its activities are coordinated by the Council of EATCS, which elects a President, Vice Presidents, and a Treasurer. Policy guidelines are determined by the Council and the General Assembly of EATCS. This assembly is scheduled to take place during the annual **I**nternational **C**olloquium on **A**utomata, **L**anguages and **P**rogramming (ICALP), the conference of EATCS.

MAJOR ACTIVITIES OF EATCS

- Organization of ICALP;
- Publication of the "Bulletin of the EATCS;"
- Award of research and academic career prizes, including the EATCS Award, the Gödel Prize (with SIGACT), the Presburger Award, the EATCS Distinguished Dissertation Award, the Nerode Prize (joint with IPEC) and best papers awards at several top conferences;
- Active involvement in publications generally within theoretical computer science.

Other activities of EATCS include the sponsorship or the cooperation in the organization of various more specialized meetings in theoretical computer science. Among such meetings are: CIAC (Conference of Algorithms and Complexity), CiE (Conference of Computer Science Models of Computation in Context), DISC (International Symposium on Distributed Computing), DLT (International Conference on Developments in Language Theory), ESA (European Symposium on Algorithms), ETAPS (The European Joint Conferences on Theory and Practice of Software), LICS (Logic in Computer Science), MFCS (Mathematical Foundations of Computer Science), WADS (Algorithms and Data Structures Symposium), WoLLIC (Workshop on Logic, Language, Information and Computation), WORDS (International Conference on Words).

Benefits offered by EATCS include:

- Subscription to the "Bulletin of the EATCS;"
- Access to the Springer Reading Room;
- Reduced registration fees at various conferences;
- Reciprocity agreements with other organizations;
- 25% discount when purchasing ICALP proceedings;
- 25% discount in purchasing books from "EATCS Monographs" and "EATCS Texts;"
- Discount (about 70%) per individual annual subscription to "Theoretical Computer Science;"
- Discount (about 70%) per individual annual subscription to "Fundamenta Informaticae."

Benefits offered by EATCS to Young Researchers also include:

- Database for Phd/MSc thesis
- Job search/announcements at Young Researchers area

(1) THE ICALP CONFERENCE

ICALP is an international conference covering all aspects of theoretical computer science and now customarily taking place during the second or third week of July. Typical topics discussed during recent ICALP conferences are: computability, automata theory, formal language theory, analysis of algorithms, computational complexity, mathematical aspects of programming language definition, logic and semantics of programming languages, foundations of logic programming, theorem proving, software specification, computational geometry, data types and data structures, theory of data bases and knowledge based systems, data security, cryptography, VLSI structures, parallel and distributed computing, models of concurrency and robotics.

SITES OF ICALP MEETINGS:

- Paris, France 1972
- Saarbrücken, Germany 1974
- Edinburgh, UK 1976
- Turku, Finland 1977
- Udine, Italy 1978
- Graz, Austria 1979
- Noordwijkerhout, The Netherlands 1980
- Haifa, Israel 1981
- Aarhus, Denmark 1982
- Barcelona, Spain 1983
- Antwerp, Belgium 1984
- Nafplion, Greece 1985
- Rennes, France 1986
- Karlsruhe, Germany 1987
- Tampere, Finland 1988
- Stresa, Italy 1989
- Warwick, UK 1990
- Madrid, Spain 1991
- Wien, Austria 1992
- Lund, Sweden 1993
- Jerusalem, Israel 1994
- Szeged, Hungary 1995
- Paderborn, Germany 1996
- Bologne, Italy 1997
- Aalborg, Denmark 1998
- Prague, Czech Republic 1999
- Genève, Switzerland 2000
- Heraklion, Greece 2001
- Malaga, Spain 2002
- Eindhoven, The Netherlands 2003
- Turku, Finland 2004
- Lisabon, Portugal 2005
- Venezia, Italy 2006
- Wrocław, Poland 2007
- Reykjavik, Iceland 2008
- Rhodes, Greece 2009
- Bordeaux, France 2010
- Zürich, Switzerland 2011
- Warwick, UK 2012
- Riga, Latvia 2013
- Copenhagen, Denmark 2014
- Kyoto, Japan 2015
- Rome, Italy 2016
- Warsaw, Poland 2017
- Prague, Czech Republic 2018
- Patras, Greece 2019
- Saarbrücken, Germany (virtual conference) 2020
- Glasgow, UK (virtual conference) 2021
- Paris, France 2022
- Paderborn, Germany 2023

(2) THE BULLETIN OF THE EATCS

Three issues of the Bulletin are published annually, in February, June and October respectively. The Bulletin is a medium for *rapid* publication and wide distribution of material such as:

- EATCS matters;
- Technical contributions;
- Columns;
- Surveys and tutorials;
- Reports on conferences;
- Information about the current ICALP;
- Reports on computer science departments and institutes;
- Open problems and solutions;
- Abstracts of Ph.D. theses;
- Entertainments and pictures related to computer science.

Contributions to any of the above areas are solicited, in electronic form only according to formats, deadlines and submissions procedures illustrated at `http://www.eatcs.org/bulletin`. Questions and proposals can be addressed to the Editor by email at `bulletin@eatcs.org`.

(3) OTHER PUBLICATIONS

EATCS has played a major role in establishing what today are some of the most prestigious publication within theoretical computer science.

These include the *EATCS Texts* and the *EATCS Monographs* published by Springer-Verlag and launched during ICALP in 1984. The Springer series include *monographs* covering all areas of theoretical computer science, and aimed at the research community and graduate students, as well as *texts* intended mostly for the graduate level, where an undergraduate background in computer science is typically assumed.

Updated information about the series can be obtained from the publisher.

The editors of the EATCS Monographs and Texts are now M. Henzinger (Vienna), J. Hromkovič (Zürich), M. Nielsen (Aarhus), G. Rozenberg (Leiden), A. Salomaa (Turku). Potential authors should contact one of the editors.

EATCS members can purchase books from the series with 25% discount. Order should be sent to:

Prof.Dr. G. Rozenberg, LIACS, University of Leiden,
P.O. Box 9512, 2300 RA Leiden, The Netherlands

who acknowledges EATCS membership and forwards the order to Springer-Verlag.

The journal *Theoretical Computer Science*, founded in 1975 on the initiative of EATCS, is published by Elsevier Science Publishers. Its contents are mathematical and abstract in spirit, but it derives its motivation from practical and everyday computation. Its aim is to understand the nature of computation and, as a consequence of this understanding, provide more efficient methodologies.

The Editor-in-Chief of the journal currently are D. Sannella (Edinburgh), L. Kari and P.G. Spirakis (Patras).

ADDITIONAL EATCS INFORMATION

For further information please visit `http://www.eatcs.org`, or contact the President of EATCS:

Prof. Artur Czumaj,
Email: `president@eatcs.org`

EATCS MEMBERSHIP

DUES

The dues are € 40 for a period of one year (two years for students / Young Researchers). Young Researchers, after paying, have to contact `secretary@eatcs.org`, in order to get additional years. A new membership starts upon registration of the payment. Memberships can always be prolonged for one or more years.

In order to encourage double registration, we are offering a discount for SIGACT members, who can join EATCS for € 35 per year. We also offer a five-euro discount on the EATCS membership fee to those who register both to the EATCS and to one of its chapters. Additional € 35 fee is required for ensuring the *air mail* delivery of the EATCS Bulletin outside Europe.

HOW TO JOIN EATCS

You are strongly encouraged to join (or prolong your membership) directly from the EATCS website `www.eatcs.org`, where you will find an online registration form and the possibility of secure online payment. Alternatively, contact the Secretary Office of EATCS:

Mrs. Efi Chita,
Computer Technology Institute & Press (CTI)
1 N. Kazantzaki Str., University of Patras campus,
26504, Rio, Greece
Email: `secretary@eatcs.org`,
Tel: +30 2610 960333, Fax: +30 2610 960490

If you are an EATCS member and you wish to prolong your membership or renew the subscription you have to use the Renew Subscription form. The dues can be paid via paypal and all major credit cards are accepted.

For adittional information please contact the Secretary of EATCS:

Dmitry Chistikov
Computer Science
University of Warwick
Coventry
CV4 7AL
United Kingdom
Email: `secretary@eatcs.org`,