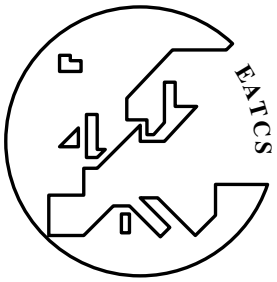
ISSN 0252–9742

Bulletin

of the

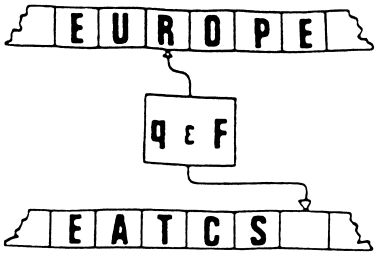
European Association for Theoretical Computer Science

EATCS



Number 139 February 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	Gerhard Wöeginger	The Netherlands

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
Gerhard Wöeginger . . . . . g.j.woeginger@math.utwente.nl

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`.

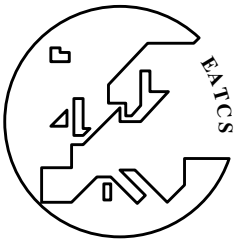
Deadlines for submissions of reports are January, May and September 15th, respectively for the February, June and October issues. Editorial decisions about submitted technical contributions will normally be made in 6/8 weeks. Accepted papers will appear in print as soon as possible thereafter.

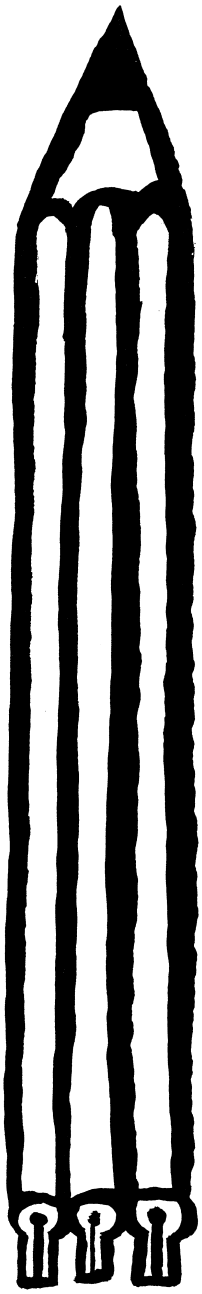
The Editor welcomes proposals for surveys, tutorials, and thematic issues of the Bulletin dedicated to currently hot topics, as well as suggestions for new regular sections.

Table of Contents

News and Conference Reports

EATCS Matters

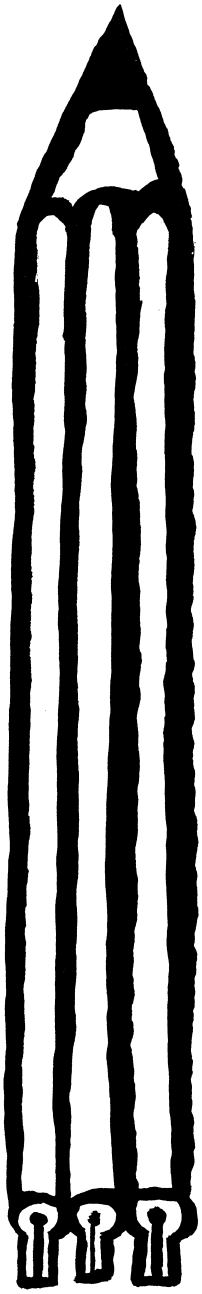




Dear EATCS members,

First of all, let me wish you a very happy 2023. I hope that this will be a healthy and fantastic year for all of us, full of great research advances, exciting conferences and workshops. I look forward to working together with all of you in order to continue promoting the development of theoretical computer science.

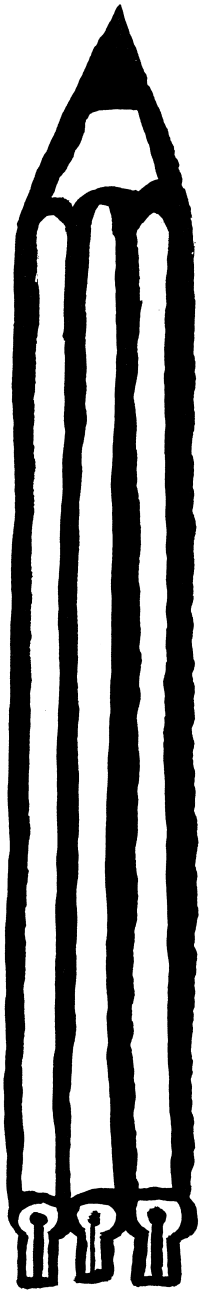
I am especially looking forward to attending the 50th EATCS International Colloquium on Automata, Languages, and Programming (ICALP 2023), the EATCS flagship conference that will be held in Paderborn, Germany, July 10-14, 2023 (https://icalp2023.cs.upb.de/). The PC chairs are Uriel Feige (track A) and Kousha Etessami (track B), and the conference chair is Sevag Gharibian. ICALP 2023 will feature five fantastic invited speakers: Anna Karlin (University of Washington), Rasmus Kyng (ETH Zürich), Rupak Majumdar (Max Planck Institute for Software Systems), Thomas Vidick (Caltech and Weizmann Institute), and James Worrell (University of Oxford). I hope that many of you have submitted your very best work to ICALP 2023 and I expect to see a great scientific program, to be selected by the PC in mid April. As usual, ICALP will be preceded by a series of workshops, which will take place on July 10. ICALP 2023 will be also the occasion to celebrate the 50th times we have the conference. It is fascinating to see how this conference has changed since its establishing in 1972, and also how the field of theoretical computer science has evolved during these years!



Also, please allow me to remind you about three EATCS affiliated conferences that will take place later in summer and fall this year: the 48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023) in Bordeaux, France, August 2023, the 31st Annual European Symposium on Algorithms (ESA 2023, https://algo-conference.org/2023/esa/) in Amsterdam, the Netherlands, September 4-6, 2023, and the 36th International Symposium on Distributed Computing (DISC 2023, http://www.disc-conference.org/wp/disc2023/) in L'Aquila, Italy, October 9-13, 2023.

But there will be some more exciting theory conferences taking place in the summer, where I hope to see strong in-person attendance even if the events are taking place outside Europe. The 55th ACM Symposium on Theory of Computing (STOC 2023, http://acm-stoc.org/stoc2023/) will be held in Orlando, USA, June 20-23, 2023. STOC 2023 is a part of the ACM Federated Computing Research Conference (FCRC, https://fcrc.acm.org/), and it will be run jointly with some other theory conferences, PODC 2023 and SPAA 2023. Another flagship theory conference, the 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2023, https://lics.siglog.org/lics23/), will be held this year in Boston, USA, June 26-29. I expect all these conferences to bring a large in-person attendance and to stimulate fantastic research advances in theory.

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

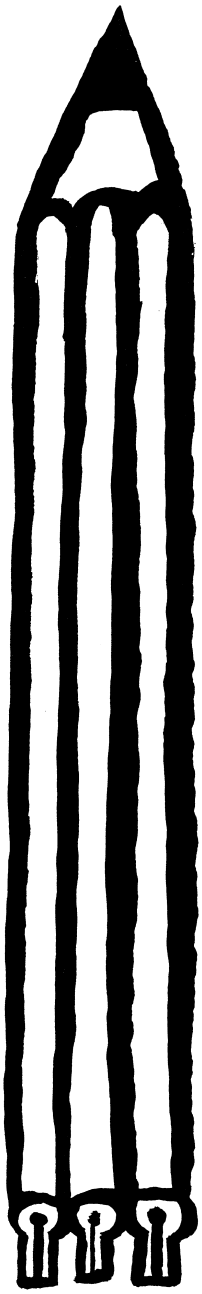


Computer Science community at president@eatcs.org. We will consider all your suggestions and criticisms carefully.

I look forward to seeing many of you around, at ICALP in Paderborn, or during other conferences or workshops that I hope to attend this spring and summer and fall, or maybe only online, and to discussing ways of improving the impact of the EATCS within the theoretical computer science community.

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

February 2023



Dear EATCS community,

I wish you a wonderful 2023!

The new Bulletin features several interesting researchers. Alexandra Silva is our first interviewee this year, and you can read more about her and her advice, e.g., for young researchers or what to do when stuck with a research problem, in the interview column. The Bulletin further includes a guest column by Boaz Barak: in the theory blogs column he tells us about his experience writing on the "Windows in Theory" blog, about his sources of inspiration, about his thoughts on mathematics and computer science education, and much more.

In the complexity column, Ryan Williams is motivated by the observation that we have a lot of information about what the proofs of longstanding open complexity lower bounds cannot look like and investigates the question: what could a proof of a strong lower bound look like? The distributed computing column features Naama Ben-David, winner of the 2022 Principles of Distributed Computing Doctoral Dissertation Award, who revisits the idea of "lock-free locks". The logical column revolves around model theory and Yuri Gurevich discusses the the umbilical cords of finite model theory. The educational column investigates how teaching informatics can contribute to improving education in general.

Last but not least, in the perspectives column, Antoine Amarilli gives an introduction to the Theoretical Computer

Scientists for Future (TCS4F) initiative which aims to make research in theoretical computer science environmentally sustainable.

I'd like to thank all the contributors to this issue.
Enjoy the new Bulletin!

Stefan Schmid, Berlin
February 2023

The Gödel Prize 2023

Call for Nominations

Deadline: March 31st, 2023

The Gödel Prize for outstanding papers in the area of theoretical computer science is sponsored jointly by the European Association for Theoretical Computer Science (EATCS) and the Association for Computing Machinery, Special Interest Group on Algorithms and Computation Theory (ACM SIGACT). The award is presented annually, with the presentation taking place alternately at the EATCS International Colloquium on Automata, Languages, and Programming (ICALP) and the ACM Symposium on Theory of Computing (STOC). The 31st Gödel Prize will be awarded at the 55th ACM Symposium on Theory of Computing (STOC), which will take place in Orlando, Florida, June 20-23, 2023.

The Prize is named in honor of Kurt Gödel in recognition of his major contributions to mathematical logic and of his interest, discovered in a letter he wrote to John von Neumann shortly before von Neumann's death, in what has become the famous "P versus NP" question. The Prize includes an award of USD 5,000.

Award Committee: The 2023 Award Committee consists of 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), and Luca Trevisan (Bocconi University).

Eligibility: The 2023 Prize rules are given below and they supersede any different interpretation of the generic rule to be found on websites of both SIGACT and EATCS. Any research paper or series of papers by a single author or by a team of authors is deemed eligible if:

- The main results were not published (in either preliminary or final form) in a journal or conference proceedings before January 1st, 2010.

- The paper was published in a recognized refereed journal no later than December 31, 2022.

The research work nominated for the award should be in the area of theoretical computer science. Nominations are encouraged from the broadest spectrum of the theoretical computer science community so as to ensure that potential award winning papers are not overlooked. The Award Committee shall have the ultimate authority to decide whether a particular paper is eligible for the Prize.

Nominations: Nominations for the award should be submitted by email to the Award Committee Chair: ronitt@mit.edu. Please make sure that the Subject line of all nominations and related messages begin with "Goedel Prize 2023." To be considered, nominations for the 2023 Prize must be received by March 31st, 2023.

A nomination package should include:

1. A printable copy (or copies) of the journal paper(s) being nominated, together with a complete citation (or citations) thereof.

2. A statement of the date(s) and venue(s) of the first conference or workshop publication(s) of the nominated work(s) or a statement that no such publication has occurred.

3. A brief summary of the technical content of the paper(s) and a brief explanation of its significance.

4. A support letter or letters signed by at least two members of the scientific community.

Additional support letters may also be received and are generally useful. The nominated paper(s) may be in any language. However, if a nominated publication is not in English, the nomination package must include an extended summary written in English.

Those intending to submit a nomination should contact the Award Committee Chair by email well in advance. The Chair will answer questions about eligibility, encourage coordination among different nominators for the same paper(s), and also accept informal proposals of potential nominees or tentative offers to prepare formal nominations. The committee maintains a database of past nominations for eligible papers, but fresh nominations for the same papers (especially if they highlight new evidence of impact) are always welcome.

Selection Process: The Award Committee is free to use any other sources of information in addition to the ones mentioned above. It may split the award among multiple papers, or declare no winner at all. All matters relating to the selection process left unspecified in this document are left to the discretion of the Award Committee.

Recent Winners (all winners since 1993 are listed at `http://www.sigact.org/Prizes/Godel/` and `http://eatcs.org/index.php/goedel-prize`):

2022: Brakerski, Zvika; Vaikuntanathan, Vinod , " Efficient Fully Homomorphic Encryption from (Standard) LWE". SIAM Journal on Computing. **43** (2): 831–871 (preliminary version in Foundations of Computer Science, FOCS 2011). Brakerski, Zvika; Gentry, Craig; Vaikuntanathan, Vinod (2012). "(Leveled) Fully

Homomorphic Encryption without Bootstrapping". ACM Trans. Comput. Theory 6(3): 13:1-13:36 (preliminary version in Innovations in Theoretical Computer Science, ITCS 2012).

2021: Andrei Bulatov, The Complexity of the Counting Constraint Satisfaction Problem. J. ACM 60(5): 34:1–34:41 (2013). Martin E. Dyer and David Richerby: An Effective Dichotomy for the Counting Constraint Satisfaction Problem. SIAM J. Computing. 42(3): 1245–1274 (2013). Jin-Yi Cai and Xi Chen: Complexity of Counting CSP with Complex Weights. J. ACM 64(3): 19:1–19:39 (2017).

2020: Robin A. Moser and Gábor Tardos, *A constructive proof of the general Lovász Local Lemma*, Journal of the ACM (JACM), Volume Issue 2, 2010 (preliminary version in Symposium on Theory of Computing, STOC 2009)

2019: Irit Dinur, *The PCP theorem by gap amplification,* Journal of the ACM (JACM), Volume 54 Issue 3, 2007 (preliminary version in Symposium on Theory of Computing, STOC 2006)

2018: Oded Regev, *On lattices, learning with errors, random linear codes, and cryptography*, Journal of the ACM (JACM), Volume 56 Issue 6, 2009 (preliminary version in Symposium on Theory of Computing, STOC 2005).

2017: Cynthia Dwork, Frank McSherry, Kobbi Nissim and Adam Smith**,** *Calibrating Noise to Sensitivity in Private Data Analysis,* Journal of Privacy and Confidentiality, Volume 7, Issue 3, 2016 (preliminary version in Theory of Cryptography, TCC 2006).

2016: Stephen Brookes, *A Semantics for Concurrent Separation Logic*. Theoretical Computer Science 375(1-3): 227-270 (2007). Peter W. O'Hearn, *Resources, Concurrency, and Local Reasoning*. Theoretical Computer Science 375(1-3): 271-307 (2007).

2015: Dan Spielman and Shang-Hua Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, Proc. 36th ACM Symposium on Theory of Computing, pp. 81-90, 2004; *Spectral sparsification of graphs*, SIAM J. Computing 40:981-1025, 2011; *A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning*, SIAM J. Computing 42:1-26, 2013; *Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems*, SIAM J. Matrix Anal. Appl. 35:835-885, 2014.

IPEC Nerode Prize 2023

Call for Nominations

Deadline: April 15, 2023

The EATCS-IPEC Nerode Prize for outstanding papers in the area of multivariate algorithmics, is presented annually with the presentation taking place at IPEC (International Symposium on Parameterized and Exact Computation). IPEC 2023 is due to take place as part of ALGO 2023 on 4-8 September in Amsterdam, the Netherlands. The Prize is named in honor of Anil Nerode in recognition of his major contributions to mathematical logic, theory of automata, computability, and complexity theory.

Award Committee:

The winning paper(s) will be selected by the EATCS-IPEC Nerode Prize Award Committee. This year's committee consists of the following people.

- Fedor Fomin, chair (University of Bergen, fedor.fomin@uib.no)
- Thore Husfeldt (IT University of Copenhagen, thore@itu.dk)
- Sang-il Oum (IBS and KAIST, sangil@ibs.re.kr)

Deadline for Nominations: 15 April, 2023. Decision: 15 June, 2023.

The Award Committee is solely responsible for the selection of the winner of the award which may be shared by more than one paper or series of papers. The Award Committee reserves the right to declare no winner at all.

Eligibility

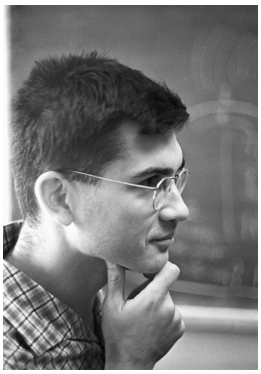
Any research paper or series of research papers by a single author or by a team of authors published in a recognized refereed journal. The research work nominated for the award should be in the area of multivariate algorithms and complexity meant in a broad sense, and encompasses, but is not restricted to those areas covered by IPEC. The Award Committee has the ultimate authority to decide on

the eligibility of a nomination. Papers authored by a member of the Award Committee are not eligible for nomination. Note that the past restrictions that require a certain number of years before/after the publication of the nominated papers have been removed.

Nominations

Nominations may be made by any member of the scientific community including the members of the Award Committee. A nomination should contain a brief summary of the technical content of each nominated paper and a brief explanation of its significance. Nominations are done by email to the Award Committee Chair with copies to the members of the committee. The Subject line of the nomination E-mail should contain the group of words "Nerode Prize Nomination".

Obituary
Yuri Manin



Yuri Manin, one of the best mathematicians of his age, passed away on January 7, 2023. He came up with the idea of quantum computing in his 1980 Russian-language book "Вычислимое и невычислимое" ("Computable and Uncomputable"), published by Советское Радио (Soviet Radio) and untranslated to English as far as I know. He was one of the two people who independently suggested this idea. The other was Richard Feynman, the legendary physicist, who put the idea more forcefully in "Simulating Physics with Computers," International Journal of Theoretical Physics 21 467–487 1982.

A three-page Part 6 of the Introduction to Manin's book is devoted to a quantum-mechanical view on computing. Discussing "a complicated network of flawless bio-chemical transformations" arising in genetic studies, he wrote that it is possible that "to make progress in understanding such phenomena, we need a mathematical theory of quantum automata ... One reason for that is that quantum state space has much bigger capacity than classical state space."

Manin's primary field was algebraic geometry, but the name "Computable and Uncomputable" of his book suggests that Manin was also interested in logic and

foundations, and that is true. He even wrote a textbook "A Course in Mathematical Logic," Springer 1977. The one time I met him was at a 2014 interdisciplinary conference on Philosophy, Mathematics, and Linguistics in St. Petersburg, Russia, https://www.pdmi.ras.ru/EIMI/2014/PhML/index.html. k In his talk, Manin traced "parallelisms between the origins of geometry, of atomism, and of alphabetical writing." We spoke briefly at the conference and corresponded in the following years.

I am not a big fan of Wikipedia, but its article "Yuri Manin", https://en.wikipedia.org/w/index.php?oldid=1133679092, retrieved on Jan. 27 2023, is good.

Yuri Gurevich

Institutional Sponsors

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

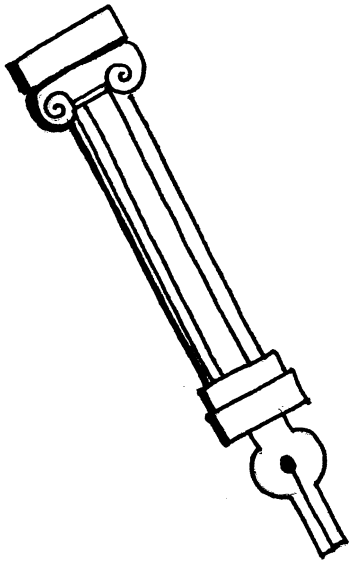
CWI, Centum Wiskunde & Informatica
Amsterdam, The Netherlands

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: Alexandra Silva

Bio: *Alexandra Silva is a theoretical computer scientist whose main research focuses on semantics of programming languages and modular development of algorithms for computational models. A lot of her work uses the unifying perspective offered by coalgebra, a mathematical framework established in the last decades. Alexandra is currently a Professor at Cornell University, and she was previously a Professor at University College London. She was the recipient of an ERC Consolidator in 2020, the Royal Society Wolfson Award in 2019, the Needham Award in 2018, the Presburger Award in 2017, the Leverhulme prize in 2016, and an ERC starting Grant in 2015.*

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?

AS: This is a photo taken at the McGill Bellairs Research Institute in Barbados. Every year since 2012 (and until the pandemic hit), I had the pleasure to join a

workshop there, organized by Prakash Panangaden. That week was always one of the most productive of my year in terms of getting new ideas and also in sparking collaborations! I hope to go back this year, for the first time since 2019!

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

AS: I have a passion for exploring new countries and their food. And when I come home I often try to replicate something I ate on those trips, which means I often travel back with local ingredients in my bag! The pandemic has disrupted the travel part, so I have instead bought myself some new cookbooks. Last year I spent some weeks trying Indian cooking (from the book of Dishoom, a restaurant in London I would often go to).

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

AS: At one point in grad school I was stuck on trying to prove completeness of a process algebra axiomatization and to take a break I decided to read one of the papers on my desk (that my advisor had told me was important!). This was Bart Jacobs' paper *A Bialgebraic Review of Deterministic Automata, Regular Expressions and Languages*. Reading that paper showed me a way to solve not only the problem I was working on but a collection of such problems in a uniform way. I have used the techniques I learned through that paper many times!

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

AS: I am particularly fond of a recent paper (which I presented as an invited talk at MFCS 2019) with my collaborators Steffen Smolka, Tobias Kappé, Nate Foster, Justin Hsu, and Dexter Kozen. This paper appeared at POPL 2020 and contains the foundations of *Guarded Kleene Algebra with Tests*, a fragment of KAT that we have shown can be decided in almost linear time. Nate, Dexter, and I had another paper (with Mae Milano and Laure Thompson) at POPL 2015 in which we had somewhat surprising experimental results on the equivalence check for a special KAT used in network verification (NetKAT). At the time we could not fully understand why the experimental results did not match the known complexity results for deciding equivalence of KAT. Only later, we realized that the programs we were using were only a fragment of NetKAT and this *guarded* fragment had much better properties!

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

AS: When I was in grad school, mornings were my most productive times. Now, with meetings and lectures taking a significant portion of my days I find late afternoons or early evenings are when I can get quiet time to do research (sometimes

after I put my son to bed!). I have been trying to block one day a week with no meetings and spending it in a quiet place, this has been very productive!

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

AS: Failures are part of the process and I always remind myself that a failure is a learning moment: you now have more information on how to attack the problem at hand. I like to take long walks when I am stuck on a problem and sometimes I simply just leave it for a couple of weeks before coming back to it.

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

AS: My first paper was at the Haskell workshop and I attended just before I started graduate school. I did not know anyone there and I was very nervous, as I was the first talk after the invited talk. The person given the invited talk had a level of enthusiasm I had never seen and that just gave me the energy I needed to get up and give my talk (which I was really afraid I would not be able to!). Almost 10 years later when I switched to attending more programming languages conferences that same speaker was there – this was Stephanie Weirich. I took this as a sign I was in the right place!

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

AS: Invest time in reading the classics. My first year in grad school my advisor kept giving me papers from the early days of Computer Science and I could not fully understand it but in hindsight I learned so much from reading those papers. Find problems that make you happy while you are trying to solve them. I have found that my motivation to do research is much higher when I work on problems that make me curious to learn more about their origin, the context, the applications. So even when I am stuck I am still happy I am working on that problem because I want to contribute to the context in which the problem arose.

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

AS: I always look for people who are curious and open to new ideas. I think it is important to understand that what you do in research is not a linear path, during grad school and afterwards if you become an academic. The problems you might work on change with time and the problems the community cares about also change, so it is with an open mind that you can let your research career evolve in a way that you remain happy with the work you do.

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

AS: I think the challenges are in some sense the opportunities: there are new areas emerging that still lack the foundations and as theoretical computer scientists it is important we work together with practitioners to develop the right abstractions.

Please complete the following sentences?

- *Being a researcher...* is never stop questioning your choices, directions, and have an open mind to the most unexpected connections.
- *My first research discovery...* was in functional programming, a subject I then abandoned during graduate school but went back to as a post-doc.
- Surrounding yourself with great collaborators ... *is key to being a happy academic.*
- *Theoretical computer science in 100 years from now...* will continue to be as important as it is nowadays, foundational work is key to progress, though the range of topics will continue evolving with the field.

The Viewpoint Column

BY

Stefan Schmid

TU Berlin, Germany
stefan.schmid@tu-berlin.de

BEATCS no 139

SHOULD CONFERENCES STILL REQUIRE MANDATORY ATTENDANCE?

A COLUMN BY THEORETICAL COMPUTER SCIENTISTS FOR FUTURE (TCS4F)

Antoine Amarilli*
Télécom Paris
antoine.amarilli@telecom-paris.fr

Abstract

Theoretical Computer Scientists for Future (TCS4F) is an initiative aimed at making research in theoretical computer science environmentally sustainable. This article presents TCS4F and gives a perspective on the current question of mandatory attendance at academic conferences.

The issue of climate change has been on our collective mind for decades. Each passing year improves our scientific understanding of the problem, and narrows down our uncertainty about the need to drastically reduce worldwide greenhouse gas emissions. As the window of opportunity is closing, and concrete action is slow to materialize, more and more groups from seemingly unrelated areas find themselves advocating for change.

TCS4F[1] is one such initiative: it is lead by computer scientists, and aims at making research in theoretical computer science environmentally sustainable. It started in 2020 with a manifesto that can be signed by researchers, conferences, and research groups. The pledge taken by signers is to follow a sustainable emissions trajectory: reduce emissions by at least 50% before 2030 relative to pre-2020 levels. The TCS4F manifesto was signed by 199 individual researchers (and counting!), 3 research groups, the 2022 edition of the ICALP conference, and the 3 conferences CSL, STACS, and Highlights of Logic, Games, and Automata.

The contribution of theoretical computer science research to the climate crisis is two-fold. On the one hand, we may be able to improve the situation through our research. For instance, we can improve the efficiency of algorithms and hope

*With help from the TCS4F team: Thomas Colcombet, Thomas Schwentick, Tijn de Vos. Thanks to Louis Jachiet for proofreading and suggestions.

[1] https://tcs4f.org/

to reduce the footprint of the ICT sector — though our efforts may well have the opposite effect because of Jevon's paradox! On the other hand, we should also think about the present impact of our research activities on the environment, and try to adapt our practices to be more sustainable.

It may be unclear at first how theoretical research harms the environment — is it about the consumption of draft paper? Whiteboard markers? In fact, while our activities can emit greenhouse gases in many ways, the main factor in our climate impact is probably long-haul plane trips. Indeed, our research field is structured around international conferences. Their stated aim is to give the community a place to meet, discuss, and exchange new ideas. Prestigious conferences are also the most important means of recognition in our community: they are a must-have on one's CV when applying for research positions. For PhDs and researchers on short-term positions, publishing there is not a choice but has become a vital professional necessity. And, until recently, publishing at international conferences naturally meant that you had to fly across the world and be there.

It is in this context that we launched TCS4F in early 2020. This coincidentally followed Vardi's "Publish and Perish" CACM column [4], which advocated for optional attendance to conferences in the name of environmental sustainability. As we all know, shortly afterwards, the COVID-19 epidemic moved all conferences online almost overnight. This forced experiment gave us a taste of what could be the closest online replacement for traditional conferences — if organized on short notice and by necessity rather than choice. The situation left us yearning for the golden days of in-person conferences and lively bar discussions in exotic locations, and the question of flight-induced climate change was not very pressing while we were stuck at home during lockdowns.

Once the COVID situation improved, many conferences adopted some kind of hybrid format, pragmatically acknowledging the fact that travel was not possible for everyone. These experiments revealed that it is comparatively easy to accommodate remote speakers, and to stream talks to a remote audience, which some conferences already had experience with. However, integrating the in-person and remote worlds proved challenging, especially for coffee breaks and social events. Based on this, some conferences are now back to firm requirements for in-person attendance, and are making explicit what used to be an implicit rule: "all talks are in-person" at ICALP'23, online talks will be for "travel restrictions or other exceptional situations" at ICDT'24... The intent may be to encourage participants to travel so everyone can enjoy a better conference... or to ensure that universities will continue to reimburse trips. Also, a fully in-person conference is of course simpler to organize, and closer to what we are used to.

These rules arguably reveal an inconvenient truth: many conferences are now attracting participants whose main goal is to have their paper published (at a prestigious venue, and on a predictable timeline), and not necessarily to attend the

event. Of course, the general will to travel and meet is still very much alive — as can be seen at events without formal proceedings, such as the Highlights workshop series. But coupling formal publications with an in-person gathering no longer makes sense for everyone.

We argue at TCS4F that decoupling the two is necessary, because plane travel is unsustainable at the scale at which we practice it. Flying across the world to a conference can amount to several tons of CO2-equivalent emissions, exceeding sustainable targets for individual yearly footprints in 2030 [2], and there are no plausible technological pathways for low-carbon intercontinental travel by then. Thus, our position at TCS4F is that, if everyone is to do their part to mitigate climate change, we must fly less — and attend less international conferences in person.

However, I believe that mandatory travel is also a question of diversity and inclusion. In-person conferences are an exclusive club for frequent travelers, and exclude people with insufficient funding to travel, people from countries who cannot easily obtain visas, people with disabilities, and people with caretaking obligations (which disproportionately affect women). For instance, the relative proportion of women participants at the 2020 International Conference on Learning Representations (online) was 20%, versus 15% at ICLR'2019 (in-person) — a 33% increase [3]. Our focus on in-person conferences thus overlooks a silent majority of people for which online attendance is the only feasible way to participate. Further, if prestigious conferences are in-person only, then recognition in our community is reserved to the privileged few who can meet this obligation.

Of course, my point is not that in-person conferences should be eliminated altogether. As we all know from the COVID era, online events are not perfect, and in-person socializing has no known replacement. Traveling to conferences is still important, and can be done responsibly — going there by train if possible, picking geographically closer locations, or simply going there less often. Online and hybrid events can also play a role, as do other forms of online research: online videos[2], online seminars[3], the Theoretical Computer Science Stack Exchange[4], etc. These new formats are especially promising when they do not try to mimic what already exists, but instead leverage features specific to the Internet: asynchronicity, low friction, low-cost, machine interpretability, long-term archival... Overall, it is very challenging to balance the scientific value of international in-person meetings with their environmental impact. But every member of our community should have a say in this choice, and it should be guided by careful deliberation — not simply by reverting to the default 20th-century-style

[2]For instance ScienceCast: `https://sciencecast.org/`

[3]For instance `https://researchseminars.org/`

[4]`https://cstheory.stackexchange.com/`

conference culture.

It is not yet clear how the conference landscape will evolve after COVID: which conferences will settle on a new format in the long run, and which ones will revert to the pre-COVID rule of mandatory participation barring extenuating circumstances. We have tried to survey this at TCS4F [1]. For conferences with optional in-person attendance, it is not clear how much organizers will encourage or discourage participants to travel, and how researchers will respond. These questions should probably be debated in our community, so the system can achieve the best compromise between scientific value, inclusivity, and environmental sustainability. But, specifically for prestigious conferences with formal proceedings, our short-term hope is that future call for papers will allow publication without in-person attendance.

We are interested at TCS4F to hear about the views of the community on this important issue. Should conference publication be conditioned to onsite participation? How should our conference culture change to be sustainable and inclusive? We are interested to hear your views at `contact@tcs4f.org`.

More reading

- Laurent Feuilloley. About mandatory attendance, `https://discrete-notes.github.io/mandatory-attendance`.
- Moshe Vardi. The paradox of choice in computing-research conferences, *Commun. ACM*, 2021, `https://cacm.acm.org/magazines/2021/11/256373-the-paradox-of-choice-in-computing-research-conferences/fulltext`.
- TCS4F blog, `https://tcs4f.org/blog`.
- Flying less in academia: A resource guide, `http://flyinglessresourceguide.info/`.
- ALLEA. Towards climate sustainability of the academic system in Europe and beyond, `https://allea.org/wp-content/uploads/2022/05/ALLEA-Report-Towards-Climate-Sustainability-of-the-Academic-System.pdf`.

References

[1] Antoine Amarilli and TCS4F. How are TCS conferences adapting after COVID-19?, 2022. `https://tcs4f.org/how-are-tcs-conferences-adapting-after-covid-19`.

[2] Tim Gore. Per capita consumption emissions and the 1.5 degrees goal, 2021. https://www.oxfam.org/en/research/carbon-inequality-2030.

[3] Matthew Skiles, Euijin Yang, Orad Reshef, Diego Robalino Muñoz, Diana Cintron, Mary Laura Lind, Alexander Rush, Patricia Perez Calleja, Robert Nerenberg, Andrea Armani, et al. Conference demographics and footprint changed by virtual platforms. *Nature Sustainability*, 5(2), 2022. https://www.nature.com/articles/s41893-021-00823-2.

[4] Moshe Y. Vardi. Publish and perish. *Commun. ACM*, 63(1), 2019. https://cacm.acm.org/magazines/2020/1/241717-publish-and-perish/fulltext.

The Theory Blogs Column

by

Luca Trevisan

Bocconi University
Via Sarfatti 25, 20136 Milano, Italy
L.Trevisan@UniBocconi.it
https://lucatrevisan.github.io

Boaz Barak is a professor of computer science at Harvard, known for a wide range of research interests, from the foundations of cryptography to computational complexity and combinatorial optimization, and for groundbreaking contributions to our field. Boaz has also thought a lot about computer science education: he is a co-author, with Sanjeev Arora, of a textbook on computational complexity and he has been developing a new introductory course in TCS.

In his guest column, Boaz tells us about his experience writing on the "Windows in Theory" blog, about his excellent sources of inspiration, about his thoughts on mathematics and computer science education, and much more.

WINDOWS ON THEORY

A Conversation with Boaz Barak

Q. Boaz, thanks for taking the time to talk about your blog to our readers. When did you start to blog, and what motivated you to start?

In 2012, Omer Reingold started a group blog for the amazing theoretical computer scientists of the Microsoft Research Silicon Valley lab, and called it "Windows on Theory". As a fellow MSR researcher, Omer invited me to join the blog as few months later. Joining a group blog seemed to me like an attractive proposition, since I didn't think I will have something interesting to say on a very regular basis. I liked the idea of explaining technical topics on a blog post, the way you might sketch them on a whiteboard to a colleague. Compared to a survey, where you have to cross all your t's and dot all your i's, and get all references straight, a blog post can be a good way to convey the heart of the matter without doing as much work. Indeed throughout the years, I've been inspired by several blog posts by you, Luca. Your blog is a great example of how to explain technical topics in an informal manner.

Q. Thank you so much for that! You have very broad interests in theoretical computer science, and you blog about a great variety of topics. Have there been instances where writing posts or discussing in the comment section has clarified ideas or lead to a conjecture or otherwise helped with your research?

I do think that my thinking on several questions, including structure vs. combinatorics, quantum skepticism, theory of deep learning, and more, have been shaped by both the process of writing essays and the discussion in comments or outside the blog that ensues. It is a different form of thinking than the typical scientific paper, and often when you sit down to write, it forces you to clarify your thoughts. This is similar to how often the best way to learn a topic is to teach it.

Q. I have followed on your blog your course on methods from theoretical physics and your posts on the foundations of machine learning and AI, and I know you have worked on a new approach to teach computability and complexity. What kind of TCS do you think we should teach to CS undergraduates who are interested in AI?

It's interesting because I think traditionally the critique of courses in theoretical CS was that we are teaching all this math, while students are going to be software developers and they just need to know how to write a website. Now it

turns out that we didn't teach enough math, and to participate in the AI revolution students need to know their gradients and Hessians. It's also the case that Neural networks are really just arithmetic circuits (and backpropagation has been rediscovered several times, including by Baur and Strassen in 1982, where they used it for circuit lower bounds). So I think the tools we teach as theoretical computer scientists are as relevant as ever. I did try to modernize my course, focusing on circuits, which are relevant not just for AI but also for the foundations of both cryptography and quantum computing. I also talk much more about randomness in computation. This means that some other materials, such as automata, need to be reduced or cut, but I think it's a good tradeoff.

Q. On a related note, what do you think that a future satisfactory theory of AI might look like?

As theoretical computer scientists, we are used to being way ahead of practice. For example, people are only starting now to implement the ideas of zero-knowledge and probabilistically-checkable proofs that were put forward by theorists in the 80s and 90s. Dwork and Naor suggested the "proof of work" protocol used by Bitcoin in 1992. (They were also ahead of the curve in another way: proposing to combat "junk email" before most people had access to email and the term "spam email" was even coined.)

In deep learning we are in a different setting: practice is ahead of theory, and people are implementing systems that they themselves don't understand. In that sense, these systems behave more like artifacts that are discovered (or evolved) than like ones that are designed. This forces us to use a different form of theory, and one that relies more heavily on experiments to figure out what are even the right questions to ask. So, we are not in our usual mode where there are easy-to-state but hard-to-prove conjectures, and our goal is to sit down with pen and paper and to prove them. But for me, theoretical computer science was never about the mode of operation but about the mission of understanding computation. So if understanding deep learning means that I needed to re-learn how to code, and rack up large bills for GPU computation, then so be it.

Q. Can you tell us a bit about the plans for changes in California math education and about your involvement in that debate?

Some colleagues in California have alerted me to a proposed change to the way K-12 math is taught there and that this change is part of a national movement. Part of this is the typical tension that always exists between teaching mathematical topics that are foundational (and often a bit more challenging) vs. "practical math". This is something that I mentioned also in the discussion regarding university teaching. In the context of high school, the new version of "practical math" is no longer accounting but "data science". There is also a twist in which it is

claimed that somehow data science is more "equitable", which is something I find offensive, as it tacitly assumes that people from certain groups are inherently incapable of accessing mathematical topics such as algebra and calculus. From my experience in teaching, both at university settings and in Ethiopia and Jamaica, nothing could be further from the truth

Now I am all for teaching students a course in some data literacy, including facility with spreadsheets and understanding the various ways that people can "lie with statistics". It's just not a replacement for math courses.

The truth is that, like at the university level, students need more math these days than ever before. By far the largest growth in job opportunities has been in quantitative fields. When data science is offered as an *alternative* to math, as opposed to complementing them, it basically serves as an "off ramp" that shuts students out of these fields, including, ironically, from careers in data science itself.

Q. In general, what are you thoughts about the role of public intellectuals that theoretical computer scientists could fill, and what are public debates where you would like to see more voices coming from our community?

In our field, we often have the experience of being humiliated by either discovering that our conjecture was wrong or being unable to prove it. I think this is not a bad experience to have had for public intellectuals, and so I would hope that theoretical computer scientists speak up more in the public sphere. Areas including immigration, science funding, open access to publications, and mathematical education are clearly central to our mission to advance science, but I think we can talk about more topics as well. For example, I recently signed an open letter protesting the Israeli government's efforts to weaken the judicial branch and the basic laws on human rights. Scientific progress relies on the ability to collaborate, so free speech and human rights are topics that we should talk about as well.

Q. I would like to ask you to pick one or a couple of your favorite posts, and tell us about it/them/

My first blog post[1] was an exposition of Fully Homomorphic Encryption with Zvika Brakerski. I like that post because we didn't just repeat what's in the papers but used the flexibility of the blog format to focus on optimizing simplicity and intuition as opposed to precision and computational efficiency. I think people have found it useful over the years. Another blog post[2] I am proud of is my post on "Men in Computer Science". I mostly made obvious points in that post, but heard from several women that they appreciated it.

[1] `https://windowsontheory.org/2012/05/01/the-swiss-army-knife-of-cryptography/`
[2] `https://windowsontheory.org/2017/08/16/men-in-computer-science/`

THE DISTRIBUTED COMPUTING COLUMN

Seth Gilbert
National University of Singapore
seth.gilbert@comp.nus.edu.sg

This month, the Distributed Computing Column is featuring Naama Ben-David, winner of the 2022 Principles of Distributed Computing Doctoral Dissertation Award. Her work on concurrent systems both builds critical theoretical foundations, while also addressing practical concerns of real-world systems. Underlying much of her work is a focus on performance: how do we design real concurrent systems that scale better and run faster? Within that context, Naama Ben-David has addressed a wide variety of important questions, such as the use of RDMA (remote direct memory access) memory, the impact of NVRAM (non-volatile random access memories), and how to design high-performance Byzantine agreement.

In this column, Naama Ben-David revisits the idea of "lock-free locks," an approach to concurrency that realizes many of the benefits of both lock-based and lock-free algorithms. Lock-free locks provide the same guarantee to a programmer as a typical blocking lock, while at the same time allowing for stronger progress guarantees, e.g., lock-freedom and wait-freedom. (This seemingly impossible combination is made possible by requiring that the critical section have the property of "idempotence," meaning that it can safely be executed more than once.)

Naame Ben-David gives an overview of the state-of-the-art for lock-free locks, and discusses several interesting open questions that remain. It is clear both that the approach is quite promising, and at the same time there is much work left to do!

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 139

LOCK-FREE LOCKS

Naama Ben-David
(VMware Research)

1 Locks

Modern systems make use of multiple processes to speed up tasks that can be parallelized. However, inevitably, when multiple processes run simultaneously in the same system and are accessing the same resources, they sometimes need to synchronize. More specifically, on modern multicore architectures, processes must coordinate accesses to the same shared memory to avoid overwriting each other's work and causing inconsistencies. This is the main challenge addressed in the study of concurrent programs; how do we ensure safe coordination among processes?

Perhaps the most common way to do this is through the use of *locks*. A lock is a primitive that protects a prespecified section of memory, and allows only one process to access that memory at a time. This allows processes to safely modify that memory without worrying about potential interference from other processes. The problem solved by locks is called *mutual exclusion*, first introduced by Dijkstra [19, 20]. In its simplest form, mutual exclusion specifies three sections of code that a process might be in; the entry section, the critical section, and the exit section. Intuitively, processes in the entry section are competing to acquire the lock, a process in the critical section is holding the lock, and processes in the exit section have just released the lock. Mutual exclusion guarantees that at any point in time, at most one process can be in the critical section, and that if there are processes in the entry section, eventually there will be a process in the critical section.

Since its introduction in the 1960s, the mutual exclusion problem has unsurprisingly garnered a lot of attention, with a lot of research into how to design mutual exclusion algorithms with better guarantees [36, 46], and fitting the requirements of new architectures [14, 18, 26, 32, 33], as well as several surveys on the topic [13, 45].

Use of locks in practice. Locks are used in many practical systems, including transactional systems [50], file systems [31], databases [16], and concurrent data

structures [8] to allow for increased parallelism without risking the safety of program logic. When designing a system or data structure using locks, an important decision must be made: at what *granularity* should the locks be used? In other words, how much memory should a single lock protect? This decision exposes a difficult tradeoff; it is simplest to write code when locks are coarse grained, meaning that each lock protects a large portion of the application's memory, since this usually means that only one lock must be acquired per operation. However, the more memory a single lock protects, the more likely it is that other processes will *contend* on that lock, therefore causing more sequential bottlenecks. Often, systems opt to use locks in a *fine-grained manner*. That is, rather than having a single global lock that protects the entire system, many locks are defined, protecting small pieces of memory. For example, in many transactional systems, one lock is assigned per data item [35,50,52]. This means that when executing a transaction on several data items at a time, the locks for all of them must be acquired before any change is made on any of the memory.

Downside of locks. While locks provide a simple abstraction for safely synchronizing concurrent processes, they suffer from a major drawback: when one process holds the lock, it blocks all others from accessing the memory that the lock protects. This may sound inevitable, since after all, preventing concurrent execution on that piece of memory is the goal. However, the manner in which it is done can in fact be quite problematic in practice, because for many different reasons, processes in a system often operate at very different speeds. For example, a process may be scheduled out by the system for a long period of time, during which it does not execute any code for the program, while others continue their execution. Another reason for stalling is caching issues, or architectural features that place some processes further away from parts of the memory than others. With all of these factors coming into play, bottlenecks can often form when a slow process is holding and not releasing a lock. This is especially when there is high contention, since then many other processes are waiting for this process to finish, causing a lot of wasted CPU cycles.

2 Lock-Freedom

Lock-free algorithms avoid this drawback of locks; they guarantee that progress is made in the system even if some process fails or stalls for an arbitrarily long time. Lock-free algorithms achieve this by carefully reasoning about the semantics of a program or data structure, and designing algorithms that can use just small atomic primitives that are provided by the hardware, like compare-and-swap (CAS), to synchronize processes. Lock-free data algorithms have been the topic of extensive

study, and many efficient lock-free data structures have been designed, including BSTs [9,11,21,43], queues [34,41,42], hash tables [40,44,48], priority queues [2, 7, 37, 49, 55], and linked-lists [28, 47]. However, lock-freedom comes at the cost of increased programming effort; the elegant abstraction that locks provide, which allows programmers to write sequential code and be guaranteed that it will be safe in a concurrent setting is no longer available.[1]

Various research efforts have been made to ease the design and implementation of lock-free programs. One approach has been to observe that many lock-free algorithms have a similar structure, and exploit that structure to extend and optimize many lock-free programs in a general way. For example, this was done in the definition of the *normalized form* of lock-free data structures [51], which was then used in several works to add useful properties to any normalized data structure [5, 15, 51]. In a similar vein, recent works have shown how to add range queries to a large class of lock-free search data structures [54], and how to make a different class of lock-free tree data structures persistent in an efficient manner [24]. Another approach to ease the design of lock-free algorithms has been to present helpful primitives that can be used instead of just individual word-sized CASes. For example, some work has introduced lock-free ways to extend CAS to affect two words at a time [25], or several words at a time [23, 27, 29, 38]. Similar primitives have also been introduced with a focus on making commonly recurring constructs in lock-free data structures simpler to implement [10, 12].

3 Lock-Free Locks: Best of Both Worlds?

So far, we discussed two approaches to synchronizing concurrent processes; locks, which are easy to use but can cause processes to *block* others from making progress, and lock-freedom, which does not block, but requires careful design and is difficult to generalize. We briefly surveyed a few efforts to make lock-free code easier to program through the design of useful lock-free primitives.

Lock-free locks offer an abstraction that achieves the best of both worlds, and can perhaps be seen as the most general extension of the above-mentioned trend of presenting easy-to-use lock-free primitives. More precisely, lock-free locks give the same interface and safety guarantee as a regular blocking lock (namely, that at any given time, at most one process holds the lock and that process can execute a critical section of code atomically on the protected memory), but provide a lock-free progress guarantee. At a high level, this is achieved by having a process holding a lock leave a *descriptor* of its critical section for others to see. Other

[1] While there are some *universal constructions* that can be used to generate general-purpose lock-free data structures [1, 22, 30], these constructions sequentialize all accesses to the data structure, and are thus inefficient.

processes contending on a lock then *execute the critical section of the current lock holder*. Once this is done, the contending processes can safely release the lock from the ownership of the previous process and take it for themselves.

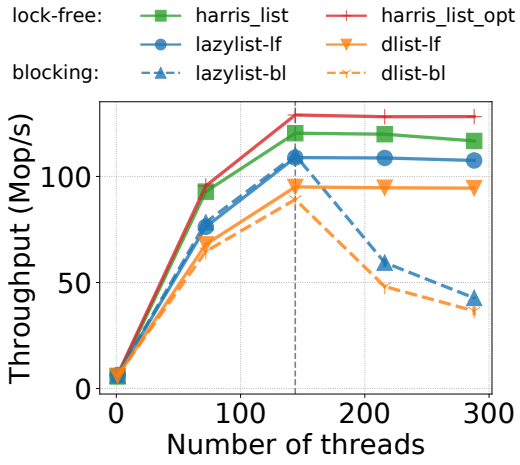
However, this must be done carefully, since having several processes potentially execute the same code simultaneously may result in that code having a different effect than intended, thereby breaking safety guarantees. Thus, for lock-free locks to work, the critical section that is run must be *idempotent*, i.e. it must have the same effect whether it is run just once or several times concurrently.

Origins of Lock-Free Locks. The idea of lock-free locks was first introduced in the 1990s by Barnes [3] and independently by Turek et al [53]. Both papers present similar ways of taking a fairly general class of code and converting it to be idempotent, thereby allowing lock-free locks to be used for any critical sections that fall within that class. However, the idempotent constructions presented in these early works required inefficient mechanisms that made them impractical. In particular, these papers achieved idempotence through a *context-saving* approach, in which after every instruction in the critical section, the entire context of the program (including program counters and registers) must be saved to allow others to continue the execution from that point. This heavy-handed approach would not only be slow, but would also require a special-purpose compiler to implement in practice. Lock-free locks have therefore been written off as impractical, and not been studied much further for about 30 years.

Renewed Interest. Recent work [6] has renewed interest in lock-free locks by introducing a new approach that makes them much more practical. At a high level, this new approach converts any critical section to an idempotent version of itself through a *log-based* mechanism rather than a context-saving one. This mechanism introduces a log that is shared among all helpers of a given critical section, and ensures that they all observe the same shared memory values for each instruction in the code by logging the first value observed by any process that ran the code. This approach does away with inefficient context-saving, instead requiring only one extra log access per instruction. Additionally, it allows the approach to be implemented in a simple library, thus enabling code written for blocking locks to be easily converted to its lock-free counterpart. In the next section, we describe this idempotence construction in more detail.

Practical potential. Ben-David et al. implemented their idempotence construction and the resulting lock-free lock abstraction in a library, demonstrating the potential that lock-free locks have [6]. In particular, they converted several lock-based data structures into lock-free ones by simply applying the library, and ran

experiments in various environments comparing the performance of lock-based data structures to that of lock-free ones. While their experiments were much more thorough, we show a representative plot from their paper here, in Figure 1.



(a) 100 keys, 5% up., $\alpha = 0.75$

Figure 1: Throughput of singly and doubly linked lists. The 'bl' and 'lf' suffixes represent the blocking and lock-free version of the lock algorithm of [6], respectively.

The figure shows the empirical performance of various implementations of linked-lists (both singly and doubly linked). In particular, two hand-tuned lock-free implementations from the literature are shown (Harris's lock-free singly linked list [28] (`harris_list`), and an optimized version of Harris's list [17] (`harris_list_opt`)). Furthermore, two simple lock-based linked-list algorithms, one singly linked (`lazylist`) and the other doubly linked (`dlist`) are implemented, and converted to be lock-free using the lock-free lock library of [6]. Both their blocking and their lock-free version are shown. In the plot, blocking algorithms are represented with a dotted line. The plot shows the algorithms' scalability as the number of threads increases. A workload of 5% updates, split evenly between inserts and deletes, and 95% lookups is run, and keys are chosen according to a zipfian distribution with parameter 0.75. The experiment was run on a machine with 72 physical cores, each with two-way hyperthreading, so the number of parallel processes possible at any one time on the machine is 144.

Unsurprisingly, none of the algorithms scale much once the physical limit of parallelism is hit. However, it is interesting to note that the blocking algorithms' performance drastically degrades once that limit is hit, and the system becomes

oversubscribed (i.e., uses more processes than are available on the machine). This nicely demonstrates the downside of blocking locks; in an oversubscribed system, it is much more likely that a process holding a lock will be scheduled out by the system and will stall for long periods of time. The plot clearly shows that lock-free locks fix this problem. Importantly, in practice oversubscription may be difficult to avoid, since most machines are used for running several independent applications at a time, and those applications are unaware of each other's resource usage.

It is also clear that while the two lock-free algorithms that employ the lock-free lock library scale similarly to their hand-designed lock-free competitors, and much better than their blocking counterparts, they are still slower than the hand-designed versions. This is also unsurprising. Using a general methodology almost always implies giving up possible optimizations. However, the performance of the library-based versions is still competitive, and shows the potential that lock-free locks have.

The rest of this article. It is clear from the above discussion that lock-free locks have the potential to make lock-free algorithms easy to design and implement efficiently. However, there is still plenty of room to improve both their practical performance and their theoretical guarantees. In the rest of this article, we overview the current state of the art for lock-free locks; we discuss the notion of idempotence in more detail and present the idempotence construction of [6], and then discuss an algorithm that makes lock-free locks guarantee the stronger *wait-free* progress in a scalable manner [4]. After presenting these algorithms, we conclude the article with a brief discussion of the many open directions left to explore in this space. However, before delving into what is known and unknown about lock-free locks in more detail, we first briefly discuss realistic expectations about what lock-free locks can offer, and what they cannot.

3.1 What Lock-Free Locks Aren't

While we believe that the idea of lock-free locks carries a lot of potential, we must also recognize the limitations of this approach. In particular, there are various causes for slowdowns and bottlenecks in concurrent systems that cannot be fixed by replacing locks with their lock-free counterparts, regardless of how efficient the lock-free lock constructions can get. We now briefly discuss two such potential bottlenecks, where fixing them can be crucial for the performance of a system, but the solution will not be found by delving deeper into the lock-free lock approach.

Buggy critical sections. In this article so far, we discussed some reasons that processes may be slow while executing their critical section, for example, if they get scheduled out by the operating system. However, another potential reason for slowdowns experienced by a process holding a lock is bugs; if the critical section code runs into an infinite loop, for example, that process may never release the lock. The lock-free approach discussed here does not address this issue. In fact, employing lock-free locks in this situation may make matters worse; instead of having one process stuck trying to execute a buggy critical section, we may have multiple processes stuck executing that same code when trying to help. To address this source of slowdowns in lock-based system, an entirely different approach must be taken. Indeed, entire fields are dedicated to testing, debugging, and verification of software. When applying lock-free locks more broadly in practice, it would be good to combine their use with methods that ensure the correctness of the critical sections being helped.

Speeding up critical sections. Note that the lock-free lock approach has several processes redundantly (though safely) executing the same code. While this redundancy might be negligible when critical sections are short, this repeated helping can cause a lot of wasted work (CPU cycles). In some applications, the critical section that a process may want to run can be lengthy and slow, even without any performance bugs. Ideally, if many processes are all spending cycles trying to execute that critical section, one may think that that combined effort could be used to speed up the code. However, that is not what lock-free locks do. Instead, the different helping processes are each executing the entire critical section independently, racing to complete it in its entirety. This may in fact *slow down* the critical section further, since the processes may interfere with each other's cache locality. To speed up the execution of a critical section with more processes, the critical section must be carefully analyzed to find and expose its potential parallelism; this direction, while it may be very beneficial in some applications, is not explored in the study of lock-free locks.

4 Idempotence

The notion of idempotence appears in many different fields, including in linear algebra, networking, and recently, persistent memory. In all these settings, the meaning of idempotence is always intuitively the same; an operation is idempotent if applying it multiple times has the same effect as applying it just once. For concurrent programs, the definition of idempotence is a little bit more involved, since it must account for not only applying an operation multiple times sequentially, but also for the possibility that several concurrent processes executed the

same operation at the same time. It is thus surprisingly non-trivial to define. Here we briefly describe the definition of concurrent idempotence presented recently in [4] and discuss the intuition behind it. We note that similar notions have been used in the past – including in the early works surrounding lock-free locks [3,53], in the definition of normalized lock-free algorithms [51], and for persistent memory constructions [5] – but were never explicitly defined as idempotence.

Definition. To capture concurrent idempotence, we must first understand what a concurrent execution, or *history*, can look like. We model concurrency through a sequence of *steps*, where each step is an instruction executed by some process. Each process executes a sequence of steps that is dictated by the code it is running, and the steps of different processes are interleaved to form a concurrent *history*. When we discuss idempotence, we must refer explicitly to the code that generated these steps, to be able to determine whether this code is idempotent. Below is the definition of concurrent idempotence taken from [4].

> A piece of code C generates a sequence of steps S_C, which can depend on the state of memory. A step $s \in S_C$ is said to be *a step for* C, regardless of which process executes it. A *run* of a piece of code C is the sequence of steps taken by *a single process* to execute or help execute C. The runs for C can be interleaved. An *instantiation* of a piece of code C is a subsequence of the steps for C in a history H, possibly from many different runs, such that those steps are consistent with a single run of C.
>
> **Definition 4.1** (Idempotence [4])**.** *A piece of code C is idempotent if in any valid history H, there exists a valid instantiation H' of C that is a (possibly empty) subsequence of all operations from runs of C in H, such that*
>
> 1. *if there is a finished run of C (response on C), then the last step of the first such finished run must be the end of H', and*
> 2. *all steps for C in any of its runs in H that are not in H' have no effect on the shared memory.*

Intuitively, this definition allows the possibility that many different processes are executing *runs* of code C, but there is some way to combine the runs of many different processes into a subsequence of steps that were possibly executed by different processes, but together look as if they form a single run. That single run, or *instantiation* of C is intuitively 'the one that counts', and all other steps taken for C have no effect.

Log-Based Construction. We now discuss the construction of [6], which takes any piece of code C implemented from reads, writes, compare-and-swap, and memory allocation/de-allocation instructions, and constructs a version of it which satisfies the above notion of idempotence.

Recall that to use idempotence for safe lock-free locks, a process p must make a descriptor available in which it specifies its critical section. The idempotence construction presented in [6] takes advantage of the fact that there is already a descriptor shared by all processes wanting to execute this critical section and adds to its a *log* that is shared as well. The log's length corresponds to the number of instructions in the critical section, that is, there is one entry in the log per instruction in the critical section code. When a process (either p or another process contending on the lock) executes p's critical section, it uses a compare-and-swap to try to write the result of the ith instruction of its critical section execution into the ith slot of p's log. If the CAS fails, this means that another process has already executed this instruction; the process adopts the result written in the log as its own result, and proceeds from there. Intuitively, this ensures that all processes executing p's critical section read the same values (either directly from memory if they were the first to do so, or from the log otherwise), and therefore, assuming the critical sections are deterministic and processes do not rely on their private register values when executing a critical section, also write the same values.[2] Thus, overall, all executions of p's critical section have exactly the same effect. This construction is quite general. It works not only for reads and writes, but also if the critical section includes CAS instructions, and memory allocations and de-allocations.

The overhead that is introduced by this approach is easy to analyze. For each instruction in the original critical section, an extra CAS on the log is introduced. Note that if the lock is not highly contended and there is just one process executing this critical section at any point in time, then this overhead is minimal, since the log is likely cached (or mostly cached) for that process. However, if there is contention, i.e., at least two processes competing for this lock and concurrently executing this critical section, then each log access may constitute an extra cache miss, as cache coherence may move the log from the cache of one process to the cache of the other. In this case, it is likely that each non-log shared-memory access also causes a cache miss for the same reason. Thus, the log accesses approximately double the cache misses incurred by the program when there is contention. However, we note that this may constitute far more cache misses than the original process would have incurred were it to run its code using a traditional blocking lock, since cache coherence would not be a major factor in that scenario. As the

[2] These assumptions are fairly general. The second can be guaranteed by having the initiating process write its private register values in the descriptor along with the critical section code, so that all helpers can use the same values.

experimental results of Ben-David et al. show, this cost can be non-negligible, but may still pay off if processes are likely to be stalled for other reasons [6].

5 Wait-Freedom

Regardless of how they achieve idempotence, all lock-free lock constructions we discussed so far operate in the same manner: each lock has a descriptor pointer, which is `null` when the lock is free. When a process p wants to acquire a given lock ℓ, it tries to swing ℓ's descriptor pointer from `null` to its own descriptor using a CAS. If it succeeds, then p has now acquired the lock. Otherwise, this means that some other process p' has acquired the lock. p then helps p' run its critical section, which the descriptor specifies, in an idempotent manner, and then tries again to swing ℓ's descriptor pointer to its own descriptor.

This simple approach guarantees lock-free progress; as long as some process wants to acquire lock ℓ, some process will acquire lock ℓ and complete its critical section. However, there is no guarantee that any one specific process will succeed in its own acquisition of the lock as long as others are contending on it. In particular, in the description above, when p tries again to swing the pointer to its own descriptor, it may fail because a new process p'' did so first. This could continue forever, leaving p to help others continuously but never make progress for itself.

This phenomenon is by no means unique to lock-free locks. Many lock-free algorithms exhibit similar behavior; while global progress is guaranteed for the system, no individual process is guarantees to make progress. A stronger notion of progress is *wait-freedom*. A wait-free algorithm guarantees progress for each individual process within a finite number of its own steps. There's another advantage to wait-freedom: it allows us to easily discuss the complexity of an algorithm in terms of the number of steps that a process must take in the worst case to execute its operation. This is much more difficult to do in algorithms that are lock-free but not wait-free, since that number may be infinite.

This leads to a natural question: can we make lock-free locks have a *wait-free* progress guarantee? If so, how many steps does a process p need to take to acquire a lock?

First cut: a queue-based solution. One potential solution to this question would be to employ a wait-free queue per lock; processes contending on the lock can enqueue themselves onto the queue, and then help everyone ahead of them before acquiring the lock for themselves. A similar approach is commonly used in the implementation of *fair* solutions to the mutual exclusion problem (except that processes wait for their turn without helping in the case of traditional blocking locks) [32,39,46]. This solution could work quite well for lock-free locks as well.

The number of steps each process would take to acquire the lock in this case would be proportional to the contention it encountered (i.e., how many other processes were ahead of it in the queue) and the number of steps it takes to help each process, plus some overhead to enqueue itself at the beginning. If an efficient queue is used, this solution can be quite efficient.

However, there is another consideration to take into account. As discussed in Section 1, many practical lock-based systems employ locks at a fine granularity. In particular, this means that processes are likely to acquire not one lock, but several locks simultaneously before being able to execute their critical sections. In this setting, the queue-based approach can quickly lose its good step complexity guarantees. Consider, for example, a scenario in which each process in the system wants to acquire at most 2 locks, and each lock has at most 2 processes contending on it at any given time. This relatively low-contention setting should intuitively allow each process to complete its execution quickly, within a constant number of its own steps. However, long dependency chains can form; if a process p wants to acquire lock ℓ_1, which has process p_1 already on its queue, p must help p_1 complete its critical section first. If p_1 must first acquire lock ℓ_2 before it can execute its own critical section, then process p must help it acquire ℓ_2 as well. However, before doing so, it might need to help a process p_2 acquire ℓ_2, which may then need to acquire ℓ_3 as well. In this way, the number of processes that p must help before executing its own critical section could blow up to include the total number of processes in the system, despite only facing a small constant number of contenders on its own lock.

A randomized approach. Recent work has addressed this problem and presented a randomized protocol for wait-free locks in which the expected number of steps to acquire a set of locks does not depend on the size of the entire system [4]. In more detail, the algorithm considers *tryLock attempts* in which a process specifies a subset of the locks in the system that it wants to simultaneously acquire, and the critical section it wants to run if successful. A tryLock attempt may fail, in which case the locks are not acquired, the critical section is not run, and the process may try again in a new attempt. The algorithm guarantees that each attempt has a *fair* chance to succeed; if each attempt aims to acquire at most L locks, and each lock has at most C processes contending on it at any time, then the attempt has a chance of at least $1/CL$ to succeed. Furthermore, the number of steps a process takes per attempt is $O(L^2 \cdot C^2 \cdot T)$, where T is the maximum length of a critical section.

At a high level, the algorithm works by assigning a random priority to each contending process, and processes only help those who have a higher priority than their own. Each tryLock attempt gets a single priority that it uses for on all

its locks for this attempt. Each lock has a 'competing set' that can be thought of as the equivalent of its queue in the queue-based approach; this set keeps track of the processes currently contending on the lock. When starting an attempt, a process p does the following: (1) it adds its descriptor to the competing set of each lock in its desired lock set, but without specifying its competing priority. (2) it checks all of the competitors on its competing set, helps the one with the highest priority on each lock, and 'kills' all the others. If some competitor doesn't have a priority, it is skipped (not helped and not killed). That is, for any attempt in the competing set of some lock, if its priority exists but is not the highest in this set, then p sets its state to 'aborted'. No aborted attempts will be helped in the future. (3) Now p finally chooses a random priority and updates its descriptor accordingly. Process p now repeats the second step, this time with the possibility that it will be the winner.

The random priorities help avoid the long chains that prevented the queue-based approach from scaling; rather than having to help a process until it succeeds in its critical section, a process p helping a process p_1 may now abort p_1 if p_1 does not have the top priority on its other locks. This is the key idea that helps the randomized algorithm achieve its good step complexity bounds.

Subtleties and downsides of the algorithm. When analyzing randomized concurrent protocols, an *adversary* is used to model the system scheduler to capture worst case executions. It is generally assumed that the adversary does not know the future, so does not know the results of future coin flips or random decisions, but can know what has happened so far in the execution. This adversary can be quite powerful in skewing the probability that a process p will succeed. For example, in the algorithm of [4], p goes through a first round of helping before choosing its own priority and starting to compete to avoid effects that the adversarial scheduler could have. In particular, if p were to choose its priority and competes immediately, an adversary that wants to decrease p's probability of success can wait until it sees that p's competitors currently have relatively high priorities, and only let p compete at that point. This would inherently skew p's chances of success.

The algorithm of [4] achieves bounds that depend on the maximum number of locks requested in each attempt, the maximum amount of contention per lock, and the maximum length of a critical section. These bounds must be known in advance to the algorithm, as it in fact makes use of these bounds explicitly. In particular, it injects an artificial *delay* before choosing p's priority during the execution of an attempt, making p potentially take extra useless steps just to waste time, to ensure that p always takes the same number of steps between the beginning of its attempt and the point at which its priority is revealed to the adversary. This is done to

prevent the adversary from using the number of steps p takes during its execution to skew its probability of success. These delays are fairly unsatisfying, but are required to achieve the guaranteed bounds.

6 Open Problems

Since their introduction in the 1990s, lock-free locks have been mostly dismissed in the literature as impractical, and their study has only been renewed recently. This leaves our understanding of lock-free locks in its infancy, with many open problems, both theoretical and practical, left to be resolved. To wrap up this article, we briefly outline some of these promising future directions.

Reducing the overhead of idempotence. Lock-free locks must inherently introduce some overhead as compared to their blocking counterparts, since they must ensure that the code that is run in their critical sections is idempotent. The potential of lock-free locks was only understood when an idempotence construction with relatively low overhead was introduced last year [6]. However, it may be possible to improve this overhead, thereby immediately improving the performance and practicality of lock-free locks, as well as other applications of idempotence (see Section 4 for a brief discussion of such applications).

The goal of improving the overhead of idempotence constructions can be approached from several angles. The most immediate one from the discussion in this article is to find a construction that is as general as the one presented in [6], but more efficient. As discussed in Section 4, the construction of [6] already only introduces constant overhead in theory, making it difficult (though maybe not impossible) to improve the theoretical overhead. However, there may be plenty to do to improve its overhead in practice. In particular, this construction exhibits poor cache locality due to coherence issues when sharing the log among concurrent processes. It would be interesting to study whether an idempotence construction can be found that suffers less from coherence misses. Furthermore, we note that aside from the coherence issues, the construction of Ben-David et al. forces each new helping process to execute the entire critical section from the beginning, potentially wasting a lot of redundant work. This may be fine for short critical sections, but can become unacceptable when critical sections are long. Another direction for improving this construction's overhead is to find a way to allow helpers to skip ahead to where the most advanced process has reached in the critical section code. This is something that is achieved by the context-saving approaches discussed in Section 3, but at too great a cost. Is there a way to avoid having each process re-execute the entire critical section without resorting to saving the entire context of each process after each instruction?

On a similar vein, another way to optimize the construction would be to reduce its space overhead. The log used in the construction of Ben-David et al. is as long as the number of instructions in the critical section. This may be ok overhead for short critical sections, but may become prohibitive if critical sections are longer. Is there a way to reduce the space overhead required for idempotence?

A different approach to improving idempotence overhead can also be taken. Namely, rather than sticking with an idempotence construction that can be applied to extremely general code, one can ask whether there are some types of critical sections that are naturally more amenable to becoming idempotent with minimal overhead. Of course, an immediate answer is yes – some critical sections may be idempotent to begin with, therefore requiring no overhead at all. This opens up the possibility that one could define classes of code that are in the middle; not already idempotent as-is, but may easily become so. Understanding how general such classes of code may be could open up the potential to employ lock-free locks in various real-world applications almost for free. Interestingly, this direction ties in nicely with the very active research area on lock-free primitives and ways to make lock-freedom more easily applicable in general (see Section 2 for a brief discussion of these works), as such studies also aim to identify general types of code that are both useful in many applications and easy to make lock-free.

Improving theoretical guarantees of wait-free locks. We know that wait-free locks can be achieved in a fine-grained lock system in time proportional to the number of locks each process may acquire at once and the amount of contention each lock may have at any given time. These bounds are quite good for many systems, since they don't depend on the total number of locks in the system or the total number of processes in the system, both of which could be huge compared to the local contention or the size of each individual operation.

However, the bounds we have leave a lot to be desired. Firstly, it's possible that the dependencies on L, the maximum number of locks per attempt, and C, the maximum contention per lock, could be decreased. Currently, for a process to successfully acquire its locks, it needs $O(L^3 \cdot C^3 \cdot T)$ steps in expectation, with T being the maximum length of a critical section. Is it possible to reduce this to only a linear dependence? Perhaps more interestingly, recall that for these bounds to hold, L, C, and T must be known to the algorithm in advance. The algorithm loses its complexity guarantees if these bounds are surpassed at any point. This is somewhat unsatisfying; many systems may not know exact bounds on contention and size of operations in advance, and may experience workloads in which there are long periods of low contention and small operations, with intermittent busy periods where these figures are much higher. Finding an algorithm that can adapt to the *actual* amount of contention and size of operations in an execution is therefore

an important future direction.

Finally, we also note that the algorithm presented in [4] is randomized, and therefore all bounds are given in expectation rather than in the worst case. Recall that the queue-based approach described in Section 5 is deterministic, but its worst case step complexity bounds depend on the total number of processes in the system. Is it possible to find a *deterministic* wait-free lock algorithm that does not depend on the total size of the system?

References

[1] Yehuda Afek, Dalia Dauber, and Dan Touitou. Wait-free made fast. In *ACM Symposium on Theory of Computing (STOC)*, pages 538–547, 1995.

[2] Dan Alistarh, Justin Kopinsky, Jerry Li, and Nir Shavit. The spraylist: A scalable relaxed priority queue. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 11–20, 2015.

[3] Greg Barnes. A method for implementing lock-free shared-data structures. In *Proceedings of the fifth annual ACM symposium on Parallel algorithms and architectures*, pages 261–270, 1993.

[4] Naama Ben-David and Guy E Blelloch. Fast and fair randomized wait-free locks. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 187–197, 2022.

[5] Naama Ben-David, Guy E Blelloch, Michal Friedman, and Yuanhao Wei. Delay-free concurrency on faulty persistent memory. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*.

[6] Naama Ben-David, Guy E Blelloch, and Yuanhao Wei. Lock-free locks revisited. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (SPAA)*, pages 278–293, 2022.

[7] Anastasia Braginsky, Nachshon Cohen, and Erez Petrank. Cbpq: High performance lock-free priority queue. In *European Conference on Parallel Processing*, pages 460–474. Springer, 2016.

[8] Nathan G Bronson, Jared Casper, Hassan Chafi, and Kunle Olukotun. A practical concurrent binary search tree. In *Symposium on Principals and Practice of Parallel Programming*, 2010.

[9] Trevor Brown. A template for implementing fast lock-free trees using htm. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 293–302, 2017.

[10] Trevor Brown, Faith Ellen, and Eric Ruppert. Pragmatic primitives for non-blocking data structures. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 13–22, 2013.

[11] Trevor Brown, Faith Ellen, and Eric Ruppert. A general technique for non-blocking trees. In *ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, pages 329–342, 2014.

[12] Trevor Brown, William Sigouin, and Dan Alistarh. Pathcas: an efficient middle ground for concurrent search data structures. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 385–399, 2022.

[13] Peter A Buhr, David Dice, and Wim H Hesselink. High-performance n-thread software solutions for mutual exclusion. *Concurrency and Computation: Practice and Experience*, 27(3):651–701, 2015.

[14] Irina Calciu, Dave Dice, Yossi Lev, Victor Luchangco, Virendra J Marathe, and Nir Shavit. Numa-aware reader-writer locks. In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 157–166, 2013.

[15] Nachshon Cohen and Erez Petrank. Automatic memory reclamation for lock-free data structures. *ACM SIGPLAN Notices*, 50(10):260–279, 2015.

[16] Alexander Conway, Abhishek Gupta, Vijay Chidambaram, Martin Farach-Colton, Richard Spillane, Amy Tai, and Rob Johnson. {SplinterDB}: Closing the bandwidth gap for {NVMe}{Key-Value} stores. In *USENIX Annual Technical Conference (USENIX ATC)*, pages 49–63, 2020.

[17] Tudor David, Rachid Guerraoui, and Vasileios Trigonakis. Asynchronized concurrency: The secret to scaling concurrent search data structures. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[18] David Dice, Virendra J Marathe, and Nir Shavit. Lock cohorting: a general technique for designing numa locks. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pages 247–256, 2012.

[19] Edsger W Dijkstra. Cooperating sequential processes. In *The origin of concurrent programming*, pages 65–138. Springer, 1968.

[20] Edsger W Dijkstra. Solution of a problem in concurrent programming control. In *Pioneers and Their Contributions to Software Engineering*, pages 289–294. Springer, 2001.

[21] Faith Ellen, Panagiota Fatourou, Eric Ruppert, and Franck van Breugel. Non-blocking binary search trees. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 131–140, 2010.

[22] Panagiota Fatourou and Nikolaos D Kallimanis. A highly-efficient wait-free universal construction. In *ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 325–334, 2011.

[23] Steven Feldman, Pierre LaBorde, and Damian Dechev. A wait-free multi-word compare-and-swap operation. *International Journal of Parallel Programming*, 43(4):572–596, 2015.

[24] Michal Friedman, Naama Ben-David, Yuanhao Wei, Guy E Blelloch, and Erez Petrank. Nvtraverse: In nvram data structures, the destination is more important than the journey. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 377–392, 2020.

[25] George Giakkoupis, Mehrdad Jafari Giv, and Philipp Woelfel. Efficient randomized dcas. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1221–1234, 2021.

[26] George Giakkoupis and Philipp Woelfel. Randomized abortable mutual exclusion with constant amortized rmr complexity on the cc model. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 221–229, 2017.

[27] Rachid Guerraoui, Alex Kogan, Virendra J Marathe, and Igor Zablotchi. Efficient multi-word compare and swap. In *International Symposium on Distributed Computing (DISC)*, 2020.

[28] Timothy L Harris. A pragmatic implementation of non-blocking linked-lists. In *International Symposium on Distributed Computing (DISC)*.

[29] Timothy L Harris, Keir Fraser, and Ian A Pratt. A practical multi-word compare-and-swap operation. In *International Symposium on Distributed Computing (DISC)*, pages 265–279. Springer, 2002.

[30] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(1):124–149, 1991.

[31] William Jannen, Jun Yuan, Yang Zhan, Amogh Akshintala, John Esmet, Yizheng Jiao, Ankur Mittal, Prashant Pandey, Phaneendra Reddy, Leif Walsh, et al. {BetrFS}: A {Right-Optimized}{Write-Optimized} file system. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 301–315, 2015.

[32] Prasad Jayanti, Siddhartha Jayanti, and Sucharita Jayanti. Towards an ideal queue lock. In *Proceedings of the 21st International Conference on Distributed Computing and Networking*, pages 1–10, 2020.

[33] Prasad Jayanti, Siddhartha Jayanti, and Anup Joshi. A recoverable mutex algorithm with sub-logarithmic rmr on both cc and dsm. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 177–186, 2019.

[34] Alex Kogan and Erez Petrank. Wait-free queues with multiple enqueuers and dequeuers. In *Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, pages 223–234, 2011.

[35] Hsiang-Tsung Kung and John T Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, 6(2):213–226, 1981.

[36] Leslie Lamport. A new solution of dijkstra's concurrent programming problem. *Communications of the ACM*, 1974.

[37] Yujie Liu and Michael Spear. A lock-free, array-based priority queue. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 323–324, 2012.

[38] Victor Luchangco, Mark Moir, and Nir Shavit. Nonblocking k-compare-single-swap. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 314–323, 2003.

[39] Peter Magnusson, Anders Landin, and Erik Hagersten. Queue locks on cache coherent multiprocessors. In *Proceedings of 8th International Parallel Processing Symposium*, pages 165–171. IEEE, 1994.

[40] Maged M Michael. High performance dynamic lock-free hash tables and list-based sets. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 73–82, 2002.

[41] Maged M Michael and Michael L Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 267–275, 1996.

[42] Gal Milman, Alex Kogan, Yossi Lev, Victor Luchangco, and Erez Petrank. Bq: A lock-free queue with batching. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 99–109, 2018.

[43] Aravind Natarajan and Neeraj Mittal. Fast concurrent lock-free binary search trees. In *Proceedings of the 19th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 317–328, 2014.

[44] Jesper Puge Nielsen and Sven Karlsson. A scalable lock-free hash table with open addressing. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–2, 2016.

[45] Michel Raynal and Gadi Taubenfeld. A visit to mutual exclusion in seven dates. *Theoretical Computer Science*, 919:47–65, 2022.

[46] Michael L Scott and William N Scherer. Scalable queue-based spin locks with timeout. In *Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 44–52, 2001.

[47] Niloufar Shafiei. Non-blocking doubly-linked lists with good amortized complexity. In *19th International Conference on Principles of Distributed Systems (OPODIS 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[48] Ori Shalev and Nir Shavit. Split-ordered lists: Lock-free extensible hash tables. *Journal of the ACM (JACM)*, 53(3):379–405, 2006.

[49] Håkan Sundell and Philippas Tsigas. Fast and lock-free concurrent priority queues for multi-thread systems. *Journal of Parallel and Distributed Computing*, 65(5):609–627, 2005.

[50] Adriana Szekeres, Michael Whittaker, Jialin Li, Naveen Kr Sharma, Arvind Krishnamurthy, Dan RK Ports, and Irene Zhang. Meerkat: Multicore-scalable replicated transactions following the zero-coordination principle. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–14, 2020.

[51] Shahar Timnat and Erez Petrank. A practical wait-free simulation for lock-free data structures. In *Proceedings of the 19th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 357–368, 2014.

[52] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. Speedy transactions in multicore in-memory databases. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 18–32, 2013.

[53] John Turek, Dennis Shasha, and Sundeep Prakash. Locking without blocking: making lock based concurrent data structure algorithms nonblocking. In *Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 212–222, 1992.

[54] Yuanhao Wei, Naama Ben-David, Guy E Blelloch, Panagiota Fatourou, Eric Ruppert, and Yihan Sun. Constant-time snapshots with applications to concurrent data structures. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 31–46, 2021.

[55] Martin Wimmer, Jakob Gruber, Jesper Larsson Träff, and Philippas Tsigas. The lock-free k-lsm relaxed priority queue. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 277–278, 2015.

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 139

How Teaching Informatics Can Contribute to Improving Education in General

Juraj Hromkovič and Regula Lacher
Department of Computer Science, ETH Zurich
juraj.hromkovic@inf.ethz.ch, regula.lacher@inf.ethz.ch

Abstract

Current education focuses on teaching facts, models, methods, and skills–and applying them to solve different tasks. Due to computer science, most of these tasks can be automized nowadays. As a result, future education has to change its focus on supporting students in exploring their creativity and on training critical thinking. In this article, we first explain how one can support the development in this direction, and then show by some examples that well-designed computer science textbooks can take a pioneering role in this process.

1 Concept for Teaching in the 21st Century

The main goal of education is the holistic development of the students' personalities (taking into account their individual talents) with the aim of maximally supporting the development of their potential, thus ultimately contributing to the development of society. To learn means to build one's own new concepts in context (abstraction and modeling), to think critically, and to be creative. Schools are expected to prepare their graduates for future careers–and more broadly for the future world.

However, in the 21st century, no one can reliably predict what kind of competences will be asked in professions in 20 years. All we know is that everything that is understood to some degree will be automated. Learning to operate equipment, apply predefined methods, or train certain skills will have less and less educational value in the future. Digital technologies already make it possible to perform these activities more accurately, faster, and more reliably. The only advantage humans have over man-made technologies is critical thinking, creativity, and the associated ability to improvise and redesign. Therefore, schools in the 21st century must focus primarily on the emotional and intellectual development of students through activities that foster their creativity and critical thinking.

Almost all educational systems to a high extent remain stuck in the paradigm of educational requirements from the time of the industrial revolution of the 19th century. In that time, the labor market needed experts who were able to apply complex procedures (methods, algorithms) to solve various tasks. However, these tasks are now largely automated. This also corresponds to the content of today's textbooks, which were written with the aim of imparting knowledge and practicing certain skills (following operating instructions and executing algorithms). In this sense, this is only a higher form of memorization. It is not enough for the school to teach developed concepts, teaching must encourage and enable the students to walk on the paths of discoveries and innovations to rediscover basic concepts, notions, methods, and technologies.

The current state of the educational system cannot be changed in the short term. It is necessary to set in motion an evolutionary process that, in a longer sequence of steps, "gently" transforms the current educational model, focused on teaching prefabricated knowledge and certain skills, into an education that develops the potential of students in dimensions that will not be replaced by automation and technology. The motto for teaching and textbook design must be:

> Do not teach the products of science (facts, theorems, methods, models, technologies) and the skillful use of them, but the processes of their discovery and creative development.

This approach is in line with the current international trend of acquiring competence. Competence is not the ability to solve routine problems according to a learned method. Competence is the expertise with which we are able to act meaningfully and originally in new situations based on our knowledge and experience.

Graduates of such training are truly innovative, with an intrinsic motivation to discover, understand, and create. They do not believe any statement presented to them, but they question those statements independently, examining their genesis and rationale.

Criteria For the Creation and Evaluation of Textbooks

The classical, common criteria of technical correctness and comprehensibility remain the basic prerequisites for good teaching. However, these criteria are not sufficient to achieve the above goals. The key to initiate the process of innovation in teaching is to formulate criteria that will help teams of authors to develop textbooks with the above goals in mind. First and foremost, the quality of any textbook—and thus teaching—should be measured by the extent to which it can support the achievement of the following goals:

1. Decomposition of the learning content and of its discovery process into a sequence of small steps, so that students have a realistic chance to go through the individual steps independently and thus to rediscover and master knowledge largely on their own.

2. Detailed and easy-to-understand explanations that allow for a mostly independent study or a study with minimal assistance. Each student, according to individual needs, can work through the topic any number of times at any individual pace. The examples and solutions to all tasks include not only the results and a brief description of the approach, but also the thoughts that clarify why the solver proceeded in the manner presented. If there are several possible ways of solving the problem, alternative solutions should be presented.

3. A motivating introduction to each topic to cognitively activate the class.

4. Support each student individually. An easy-to-understand approach to the basic knowledge and to the activities that must be mastered by all, as well as challenging topics for gifted and motivated students. The textbook must allow the teacher to differentiate the objectives in accordance with the abilities of each student.

5. The students should learn to a high degree through creative activities. The goal is to train the students to analyze the properties, functionality, and applicability of the products of their own work–and, in this context, to motivate them to improve their products.

6. Guide the students in conducting experiments and in analyzing the results of the experiments in order to acquire new knowledge.

7. Motivate and strengthen the will to try to solve problems and to learn from failed attempts (mistakes) that did not lead to the set goal(s).

8. Carefully cultivate precise terminology and to practice precise wording in all forms of communication.

9. Increase the sustainability of the newly acquired knowledge by placing it in the context of already available knowledge.

10. Promote teamwork with intensive communication while solving tasks.

11. Offer instruments enabling students to measure their progress independently.

All of the above goals taken together should "guarantee" the sense of accomplishment that is the most effective teaching method. The final objective must be the increase of self-confidence of the students, and their willingness to solve problems and to create new products on their own.

Writing textbooks which achieve the above goals requires a high level of expertise from the writing team in several areas–more specifically:

- In-depth subject knowledge in a broader and deeper context including the genesis of the development of the scientific discipline under consideration;
- pedagogical experience from teaching in the specific age group;
- and knowledge of subject didactics of the scientific discipline under consideration.

When creating textbooks, we recommend the use of the following elements:

- *Elements of cognitive activation for the introduction to the subject*–e.g., puzzles, surprising observations, seemingly contradictory statements, or attractive applications;
- *examples with presentations of approaches* and reflections on them;
- *learning tasks* with the aim of discovering new relationships (context) or finding a way to solve a problem;
- *analysis of and learning from failures*;
- *design of experiments, execution thereof, and analysis of results*;
- *project tasks* to solve problems in teams and to create products with the desired properties;
- *classic exercises* to consolidate what has been learned;
- *historical and social remarks* explaining the genesis of the studied topic in a scientific and social context;
- *questions* to verify the correct use of technical terminology;
- *frequent summaries* of the newly acquired knowledge in the context of the already established knowledge;
- and *test items* to independently test the acquired subject knowledge.

Each textbook should be accompanied by a teacher's guide, which should also be accessible to the students' parents. This methodological material should ensure the following:

- Provide teachers the knowledge transfer related to the subject to be taught (in a broader and deeper context);
- show teachers in detail how the lessons can be implemented, explaining exactly the objective of each element of the textbook and how to verify the success of the corresponding learning process;
- explain how the textbook can be used to guide students individually in the learning process;
- provide a clear specification of objectives and instruments for measuring the learning progress;
- give detailed solutions to the exercises including didactic comments;
- provide advice and recommendations on how to respond to unexpected original approaches proposed by students.

2 Teaching Informatics As a Pattern For Teaching Other Subjects

Why can teaching informatics take a crucial role in transforming education in the direction proposed in the first section? Let us consider *constructionism* of Seymour Papert [20–22], where the main idea can be expressed by the genius sentence "Learning by getting things to work." Starting by "Learning as building knowledge structures" of Jean Piaget [23], and continuing by "Learning by doing" of Aristotle [1] and Comenius [5], students construct some products (programs, secret codes, number representations, data organization, hashing functions, algorithms, etc.). But finishing the construction of their products is not the end of their activity, but rather the beginning of their learning process. The immediate next step is to investigate the functionality and the properties of their own products, not only to correct them if required, but especially in order to get new ideas on how to improve them or how to extend their functionality. This brings students on a new starting line of their creative activity with the goal to construct something better than they did before. This is exactly how humans live, work, and learn since the dawn of mankind: an infinite loop of improvements and motivations, each step enabling a deeper understanding and increased expertise. This is the only way to properly build up competences.

In what follows, we discuss only two concepts related to teaching programming and data security. For a large number of examples of such teaching sequences we, refer to the textbooks [2, 3, 8, 9, 12, 13, 15, 19] for students and textbooks for teachers [4, 6, 7, 10, 11, 14, 16, 18], covering all ages from kindergarten to high school.

Programming

Why do we program? Because we want to explain some activity to a computer or to a robot in such a way that the technical system becomes able to execute this activity autonomously. In these terms, a computer program is a description of an activity in a programming language that is understandable for the machine. Developing a program is a very creative and constructive activity that fits our main goals because:

- One has to find a strategy of how to reach a given goal. Thus, problem solving is the key issue. Students can first train to correctly interpret the problem description, then to develop their own descriptions of the problem by using concepts such as tables, graphs, equations, etc., and finally build experience by trying to solve concrete problem instances. After developing a solving strategy, they can test its functionality by searching for problem instances for which their strategy does not work, or look for arguments as to why the strategy works properly for all instances of the problem. This is like testing a model by experiments.

- After having designed an algorithm (solving strategy), students have to describe the algorithm by a program in a programming language. A program as a product of the students' own work can be analyzed with the focus on its functionality. This is one of the reasons why tasks for programming novices deal with moving in the plane or drawing a geometric shape in order to enable to visually study the execution of the program. If the program does not work properly, the children have to try to fix a logical error in the program description or revise their strategy. If the program works properly, the students can think about extending its functionality or about formulating more complex goals and use the program developed as a module (building block) for developing a new program for the more complex task.

- Advanced students can analyze their programs with respect to their computational complexity, compare the efficiency of different programs, and then start searching for an algorithm or an implementation, respectively, that is more efficient than those developed up to now.

- Trying to reach the competences of a good programmer, it is not sufficient to develop programs for different purposes. Students must be trained to analyze programs of others, to correct them, and to modify them for some new purposes. All of this again means a lot of testing (experimental work) in order to understand the functionality of the program investigated.

- A program is a syntactically correct text in a programming language. A computer can interpret and execute a program only if there is no syntactical (grammatical) error in the program. A program is a precise and complete description of an activity, and the program as a text has only one unique representation. Thus, writing programs is good training in the consequent application of the syntactic rules as well as in semantically correctly expressing what one would like to describe.

We see that the goals 1, 2, 5, 6, 7, and 10 of good teaching listed in Section 1 are implicitly included in well-designed programming courses. In addition, all the remaining goals can be achieved as well when the details of teaching sequences are designed properly. We did consider the first goal by using the historical method, i.e., following the genesis of programming languages. Students start to learn programming by using a very small vocabulary. We teach them how to create new words and explain their meaning to the computer. In this way, students take part in the development of the programming language, improve its expressibility, and test this product of their own work by programming.

Cryptography

Cryptography as design and analysis of secret writings is about 4000 years old, and it is the kernel of data security. Following the first goal of good teaching we again propose to apply the historical method. The main goal is not to show some examples of secret writings or cryptosystems and to learn to use them, but to present students some concepts for designing and attacking secret writings in order to encourage them to develop their own, original secret writings and to try to break systems developed by others.

One can start to teach this topic very early (starting already in 3rd class) due to the fact that the ancient rule of secrecy was that the description of secret writings must be so simple that anyone can learn it by heart, i.e., the description does not need to be saved in written form on a medium. Following [3, 6, 7, 11, 12, 15, 16] one can see that the secret of the first secret writings based on transpositions can be described by simple pictures, and so the students can develop secret writing by describing them by simple pictures as well. The same is true for substitution-based secret writing. In this case students design new secret alphabets in a funny way and explore their fantasy.

The study of the properties (quality) of the designed cryptosystems is then done by attacking them. First, students can learn to break the cryptosystem CAESAR by analyzing the letter frequencies in given cyphertexts, and later they can learn to break any monoalphabetic cryptosystem by the analysis of the distribution of letter frequencies. Trying to make their own cryptosystems secure against the letter frequency analysis, students can reinvent the cryptosystem VIGENÈRE. For designing small steps in this direction, one can again follow the historical development, in which the first step was to increase the key (shift in the alphabet) of CEASAR by 1 after ciphering each single letter. VIGENÈRE is nothing else than jumping in the *tabula recta* following a special pattern. Another simple step towards VIGENÈRE is to use different CEASAR keys for letters at even and odd positions in a given plaintext. One can again let students design their own polyalphabetic cryptosystems before approaching VIGENÈRE. Breaking VIGENÈRE can also be done in a sequence of small steps [3] and can be the invitation for moving from context-free cryptosystems to context-sensitive (block-based) ones.

Here is again a lot of freedom to design plenty of own cryptosystems that cannot be broken by the analysis of VIGENÈRE. We stop at this point, because the description how to make modern cryptography understandable for high-school students requires much more space than we have available here. For a concise introduction of public-key cryptography in high schools, we recommend the paper by Keller et al. [17]. But the key point here is that we can teach secret writings in such a way that the students can develop a large part of crucial concepts by themselves, or one can present the development of other ideas understandable in small portions. Another reason why cryptography is very attractive to a class is that on the beginning of each new special topic one can cognitively activate the class by funny, challenging puzzles.

3 Conclusion

We presented two key computer science topics to show that they are predestined to be mastered by reaching the set educational goals and offer patterns of how to teach for other disciplines. It is important to note that all parts of informatics can be taught this way. What are the main activities of computer scientists?

- Developing object representations and codes with required properties (short, secret, self-verifying, efficient to handle, etc.);

- designing efficient algorithms for different purposes (solving problems, translating codes and languages, etc.);

- and controlling technology by programming and designing IT systems that offer numerous services.

In all these cases we can get expertise exactly in the way proposed. One designs and develops a product that can be analyzed with respect to the properties, functionality, and suitability in different applications under different conditions. The historical method can be applied to make all the knowledge available in small steps based on activities of the learners.

Acknowledgements

We thank Peter Borovansky, Martin Curuc, Dennis Komm, Viliam Kratochvil, Daniel Masarovič, Maria Smerkova, Gabriel Thullen, and Michal Winczer for interesting discussions and for their comments and on previous draft versions of this paper.

References

[1] Aristotle. *A New Translation of the Nichomachean Ethics of Aristotle.* J. Vincent. Oxford, 1835

[2] Jarka Arnold, Cédric Donner, Urs Hauser, Matthias Hauswirth, Juraj Hromkovič, Tobias Kohn, David Maletinsky, and Nicole Roth. *Informatik, Programmieren und Robotik (in German).* Klett und Balmer. Baar, 2021.

[3] Michael Barot, Britta Dorn, Ghislain Fourny, Jens Gallenbacher, Juraj Hromkovič, and Regula Lacher. *Informatik, Data Science und Sicherheit (in German).* Klett und Balmer. Baar, 2021.

[4] Michelle Barnett, Juraj Hromkovič, Anna Laura John, Regula Lacher, Pascal Lütscher, and Jacqueline Staub. *Einfach Informatik 1 / 2, Spielerisch Programmieren mit Robotern (in German).* Klett und Balmer. Baar, 2021.

[5] John A. Comenius. *Didactica magna.* 1675.

[6] Urs Hauser, Juraj Hromkovič, Petra Klingenstein, Regula Lacher, Pascal Lütscher, and Jacqueline Staub. *Einfach Informatik 1 / 2, Rätsel und Spiele ohne Computer (in German).* Klett und Balmer. Baar, 2021.

[7] Heinz Hofer, Juraj Hromkovič, Regula Lacher, Pascal Lütscher, and Urs Wildeisen. *Einfach Informatik 3 / 4. Programmieren und Rätsel lösen, teachers guide (in German).* Klett und Balmer. Baar, 2021.

[8] Heinz Hofer, Juraj Hromkovič, Regula Lacher, Pascal Lütscher, and Urs Wildeisen. *Einfach Informatik 3 / 4. Programmieren und Rätsel lösen, textbook (in German).* Klett und Balmer. Baar, 2021.

[9] Juraj Hromkovič. *Einfach Informatik 5 / 6. Programmieren, textbook (in German).* Klett und Balmer. Baar, 2018.

[10] Juraj Hromkovič. *Einfach Informatik 5 / 6. Programmieren, teacher's guide (in German).* Klett und Balmer. Baar, 2019.

[11] Juraj Hromkovič and Regula Lacher. *Einfach Informatik 5 / 6. Lösungen finden, teacher's guide (in German).* Klett und Balmer. Baar, 2019.

[12] Juraj Hromkovič and Regula Lacher. *Einfach Informatik 5 / 6. Lösungen finden, textbook (in German).* Klett und Balmer. Baar, 2019.

[13] Juraj Hromkovič. *Einfach Informatik 7–9. Strategien entwickeln, textbook (in German).* Klett und Balmer. Baar, 2018.

[14] Juraj Hromkovič. *Einfach Informatik 7–9. Strategien entwickeln, teacher's guide (in German).* Klett und Balmer. Baar, 2018.

[15] Juraj Hromkovič. *Einfach Informatik 7–9. Daten darstellen, verschlüsseln, komprimieren, textbook (in German).* Klett und Balmer. Baar, 2018.

[16] Juraj Hromkovič. *Einfach Informatik 7–9. Daten darstellen, verschlüsseln, komprimieren, teacher's guide (in German).* Klett und Balmer. Baar, 2018.

[17] Lucia Keller, Dennis Komm, Giovanni Serafini, Andreas Sprock, and Björn Steffen. Teaching public-key cryptography in school In *Proc. of the 4th International Conference on Informatics in Secondary Schools: Evolution and Perspectives (ISSEP 2010)*, volume 5941 of Lecture Notes in Computer Science, Springer-Verlag 2010, pages 112–123.

[18] Tobias Kohn and Juraj Hromkovič. *Einfach Informatik 7–9. Programmieren, teacher's guide (in German).* Klett und Balmer. Baar, 2018.

[19] Tobias Kohn and Juraj Hromkovič. *Einfach Informatik 7–9. Programmieren, textbook (in German).* Klett und Balmer. Baar, 2018.

[20] Seymour Papert. *Mindstorms: Children, Computers and Powerful Ideas.* Basic Books, Inc. Publ. New York, 1980.

[21] Seymour Papert. *The Children's Machine: Rethinking School in the Age of the Computer.* Basic Books, Inc. Publ. New York, 1993.

[22] Seymour Papert and David Harel. *Constructionism.* Ablex publishing. New York, 1991.

[23] Jean Piaget and David Harel. *The Psychology of Intelligence.* Cambridge, Harward University Press, 1950.

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 139

The Power of Constructing Bad Inputs

Ryan Williams*
CSAIL & EECS
MIT, Cambridge, Massachusetts, USA
rrw@mit.edu

Abstract

A *lower bound*, showing that a function f cannot be computed by some class C of algorithms, necessarily shows that for every algorithm A in C, there must exist a "bad" input x such that $f(x) \neq A(x)$. We consider the computational complexity of generating such bad inputs for a given f and class C, and we study how the complexity of this task relates to existing (and major open problems in) lower bounds.

1 Introduction

Out of decades of thought on how to prove complexity lower bounds (and failing), one maxim repeatedly emerges: strong complexity lower bounds are hard for us to prove. There are many formal "barriers" known to proving complexity lower bounds, such as the relativization barrier [BGS75], Razborov-Rudich natural proofs [RR97], algebrization [AW09, IKK09] (see also [AB18, AB19]), and locality [Yao89, CHO+20]. For instance, relativization tells us that we cannot rely on any generic "black-box" arguments for proving strong complexity lower bounds, but many fundamental proof techniques in complexity theory are generic in exactly this sense. The algebrization barrier generalizes the relativization barrier, showing that the non-relativizing methods behind theorems like $\mathsf{IP} = \mathsf{PSPACE}$, querying polynomials that represent computations, are not sufficient in themselves to prove (for example) $\mathsf{P} \neq \mathsf{NP}$, $\mathsf{P} \neq \mathsf{PSPACE}$, $\mathsf{EXP} \neq \mathsf{ZPP}$, $\mathsf{NEXP} \neq \mathsf{BPP}$, $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$, et cetera. I've often summarized the state of affairs as: *not only can we not prove lower bounds, but we can prove that we cannot prove lower bounds*.

*Partially supported by NSF CCF-2127597. Part of this work was completed while the author was visiting the Simons Institute for the Theory of Computing, participating in the *Meta-Complexity* program.

So, we have apparently a lot of information about what the proofs of longstanding open complexity lower bounds **cannot** look like, in that we know a variety of limitations on how such proofs must proceed. The starting point of this article is to ask the question:

Q1: *What* **could** *a proof of a strong lower bound look like?*

Intuitively, complexity barriers tell us what techniques we should try to avoid, if we wish to separate complexity classes. Can we identify *obligations* that strong lower bounds must obey, properties that such lower bound proofs (of even "easier" separations, like $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$, separating exponential time with an NP oracle from randomized polynomial time) must possess? Rather than studying what is **not sufficient** for lower bounds, could we get a handle what is **necessary**?

Please keep in mind that we are not starting from a blank slate: it is not that there are no lower bounds whatsoever. When one reads introductions like this, one might get the impression that there are essentially no complexity lower bounds in the literature. This is not really true. There are many areas within complexity theory, for which researchers have managed to establish hosts of interesting and strong limitations and no-go theorems. One of the most successful of these areas is *communication complexity* [KN97, RY20] which has aided complexity lower bounds in VLSI circuit design, streaming algorithms, and Turing machines, to take three examples. Nevertheless, it is felt that there is a gap (or maybe even a chasm) between what kinds of lower bounds can currently be proved, and what we'd like to prove. Perhaps a better question then, is:

> Q2: *What properties are missing from the lower bounds that we know how to prove, which we will have to include in a proof of (say)* $\mathsf{NEXP} \neq \mathsf{BPP}$*?*

What are the missing ingredients in our lower bound toolkit? In this article, I will highlight one type of answer, from a recent paper co-authored with Lijie Chen, Ce Jin, and Rahul Santhanam [CJSW21]. I will discuss an interesting way in which efficient algorithms will have to be central to resolving major complexity lower bound questions. Let me be clear that I do not claim to have fully answered the above question Q2, in any sense. If I have brought your attention to the question, and if I have gotten you to think about it on your own for ten minutes, I will consider my job successful. (Even if you think my answer to the question is terrible.)

2 Lower-Bounding as Finding Bad Inputs

To get a handle on question Q2, we start by looking very carefully at what it means to prove a lower bound against a class of algorithms.

Let $f : \{0,1\}^\star \to \{0,1\}$ be a decision problem, and let $\mathcal{A}$ be a class of algorithms. In particular, we stipulate that each $A \in \mathcal{A}$ has a finite-length description, and each A takes in arbitrary-length Boolean inputs, outputting a single bit. An algorithm A is said to compute f if for all but finitely many inputs x, $A(x) = f(x)$. (We permit "all but finitely many", because if there were only finitely many inputs on which A and f differ, then this finite set could be "hard-coded" directly, into another larger but still finite algorithm that computes f everywhere.)

Therefore, a lower bound that "f is not in $\mathcal{A}$" is a claim of the form:

$$(\forall A \in \mathcal{A})(\exists^\infty n)(\exists x_n \in \{0,1\}^n)[A(x_n) \neq f(x_n)].$$

That is, for every candidate algorithm A for computing f, someone has to "respond" with a bad input x_n on which A does not compute f correctly. In fact, in the given setup, we should construct infinitely many bad x_n inputs (if there were only finitely-many bad inputs, they could be hard-coded into the algorithm).

Now fix a function f, and fix an algorithm A from a class $\mathcal{A}$. We ask:

Q3: *What is the complexity of constructing a bad input x_n of length n?*

Suppose you are provided 1^n, the string of n ones. We are asking: how difficult is it for you to construct an input $x_n \in \{0,1\}^n$ such that $f(x_n) \neq A(x_n)$? (You can output whatever n-bit string you want, if n is not the length of some bad input; in principle, there may be only infinitely many bad n.)

2.1 Two Types of Lower Bounds

Roughly speaking, the literature on complexity lower bounds gives two types of answers to question Q3. That is, the known proofs of lower bounds yield two different types of answers to Q3:

1. **"Random" or non-constructive ways of choosing bad inputs.** In general, such lower bound proofs rely on counting/information-theoretic arguments. For example, many proofs of time-space lower bounds for explicit problems in P (*e.g.,* [Hen65, Maa84, BC82, Bea91, BJS01, Ajt02, BSSV03, MW19]) work by choosing a random x_n from some distribution of n-bit inputs, and arguing that the randomness in the string can confound the algorithm A into making a mistake.

   For a simple example of such a lower bound, one can prove that no deterministic finite automaton (DFA) computes the language of palindromes $\{xx^R \mid x \in \{0,1\}^\star\}$, by arguing that we could "compress" an *arbitrary* string x by storing the state q_x of the DFA in which the last bit of x is read, and the length $|x|$, which takes only $O(\log |x|)$ bits. Then, if the DFA recognizes

palindromes, there should be a unique path of length $|x|$ from q_x to a final state, and the string x^R must be encoded along the path. Thus we could encode every x with a description of length only $O(\log |x|)$. However, a random string x requires a description of length at least $|x| - 1$ with nonzero probability, so we have a contradiction.[1] This argument generalizes to show that any DFA that recognizes all palindromes of length n requires at least $2^{\Omega(n)}$ states. At any rate, the argument shows that a *random* x is a bad input with decent probability.

2. **"Efficient" ways of choosing bad inputs.** Many proofs of lower bounds based on diagonalization arguments provide an efficient method for generating the bad input.

   For example, the standard proof by diagonalization that the Halting Problem is undecidable can be modified to have this property. In particular, given the code of a Turing machine that claims to solve the Halting Problem, one can efficiently produce a "bad" input on which the Turing machine fails to correctly decide the Halting Problem.[2] Similarly, the old-school proof of the Time Hierarchy Theorem, where one constructs a hard function by simulating different time-bounded Turing machines on different inputs (and flipping the answer), also has this property.

The idea of focusing on constructing bad inputs, when reasoning about impossibility results in computing, is in fact quite old. For instance, the celebrated Myhill-Nerode theorem [Ner58] says that a set of strings S is not regular if and only if there exists an infinite *distinguishing set* for S: an infinite set of "bad" strings that effectively forces every prospective DFA to need infinitely many states.

Mulmuley [Mul10] has suggested that to make progress on separating P and NP, one must search for **algorithms** which can efficiently find counterexamples for any algorithms claiming to solve the conjectured hard language. This view has been dominant in the GCT approach towards the VNP vs. VP problem [Mul07, Mul12, IK20]. One can think of our present article as confirming Mulmuley's intuition in a broader sense than what was known before.

[1] This is the first theorem one learns on Kolmogorov complexity. See Chapter 6.4 of Sipser [Sip06].

[2] For completeness, we sketch the proof. Given the code of some Turing machine H, let D_H be code for the "diagonal" machine which on an input x, flips the answer of $H(\langle x, x\rangle)$. Then H cannot terminate with a correct output on the input $\langle D_H, D_H\rangle$.

3 Starting Point: The Work of Gutfreund, Shaltiel, and Ta Shma

The starting point of our work was thinking about a beautiful paper of Gutfrend, Shaltiel, and Ta Shma [GST07]. They showed:

> *If* $\mathsf{P} \neq \mathsf{NP}$*, then bad inputs for every claimed poly-time algorithm for SAT can be constructed in poly-time.*

In particular, the following theorem can be derived from their arguments.

Theorem 3.1 (Follows from [GST07])**.** *Assume* $\mathsf{P} \neq \mathsf{NP}$*. For every* n^k*-time decision algorithm A, there is an algorithm* R_A *such that, for infinitely many n,* $R_A(1^n)$ *outputs a formula* F'_n *of length n such that* F'_n *is satisfiable if and only if* $A(F'_n)$ *outputs "UNSAT". Furthermore,* $R_A(1^n)$ *runs in* $n^{O(k^2)}$ *time.*

That is, R_A runs in polynomial time, and prints formulas on which A fails to determine SAT correctly. We say that the algorithm R_A is a *refuter*, since its job is to refute the claim that A solves the SAT problem by producing bad inputs for A.

As their theorem is very important to our work, we will carefully outline a proof of the theorem.

3.1 Refuter for Algorithms Trying to Print SAT Assignments

Let's start by assuming $\mathsf{P} \neq \mathsf{NP}$, and derive a refuter for any n^k-time algorithm A that attempts to print a SAT assignment to its input formula, whenever a SAT assignment exists. That is, our algorithm A first attempts to print a SAT assignment to its input formula, and A determines "UNSAT" if the printed assignment fails to satisfy, otherwise A determines "SAT".

Under this setup, since it's forced to print a SAT assignment, the algorithm A will always correctly output "UNSAT" when it's given an unsatisfiable formula. Therefore, under this setup, every bad input for A must be a satisfiable formula on which A determines "UNSAT". (Furthermore, since we assume $\mathsf{P} \neq \mathsf{NP}$, there must be infinitely many such formulas.)

This refuter is a bit easier to describe, yet it already captures the key idea: *if an algorithm A claims to solve SAT, then exploit its claimed ability to find its own counter-examples.*

We define the refuter as follows.

$R_A(1^n)$: Use the Cook-Levin reduction to construct a 3CNF formula F_n that is satisfiable if and only if the following holds:

$$(\exists G : |G| = n)(\exists a)[G(a) = 1 \wedge A(G) \text{ outputs "UNSAT"}].$$

Run A on F_n. If A prints a satisfying assignment (G', a') to F_n, then output G'. If A outputs "UNSAT" instead, then output F_n.

In more detail, since A is a poly-time algorithm, the property

$$(\exists G : |G| = n)(\exists a)[G(a) = 1 \wedge A(G) \text{ outputs "UNSAT"}]$$

can be checked in NP. Therefore, applying the Cook-Levin reduction, there is a 3CNF formula F_n such that $F_n(G, a)$ is true if and only if $G(a) = 1 \wedge A(G)$ outputs "UNSAT". Since A runs in time n^k, the formula F_n output by the Cook-Levin reduction has size at most $n^{O(k)}$.

What does R_A do? The refuter R_A is asking A to print its own counterexamples! That is, if A outputs a valid SAT assignment (G', a') on F_n, then G' is a formula of length n that is a "bad input" for A, by definition of F_n. Now we have two cases.

Case 1: If A outputs a valid SAT assignment G' for F_n for infinitely many n, the proof is complete: for infinitely many n, $R_A(1^n)$ outputs a formula F_n of length n such that F_n is satisfiable if and only if $A(F_n)$ outputs "UNSAT".

Case 2: The alternative is that for all but finitely many n, A outputs "UNSAT" on F_n. In this case, $R_A(1^n)$ outputs F_n on all but finitely many n. But since $\mathsf{P} \neq \mathsf{NP}$, the formula F_n is actually satisfiable for infinitely many n. Therefore, for infinitely many n, $R_A(1^n)$ still outputs a formula that is satisfiable, yet A reports "UNSAT".[3]

3.2 Refuter for Algorithms Trying to Decide SAT

Gutfreund, Shaltiel, and Ta Shma actually give a refuter for every poly-time algorithm that attempts to *decide* SAT as well. This refuter works by exploiting the well-known search-to-decision reduction for SAT. In particular, given an algorithm A that only decides SAT, we can produce an algorithm B that can print SAT assignments (when they exist) making only polynomially many calls to A, by plugging in values of variables into the formula and calling A to check if the reduced formula is still satisfiable. Now, the refuter has the following form, for $i = 1, 2, 3$:

[3] A minor detail: in this case, R_A is (infinitely often) outputting a formula of length $L = n^{O(k)}$ on the string 1^n, so it does not meet our original specification, where we want to output a string of length n on 1^n. However, a modified algorithm R'_A which on the input string 1^L, determines n and runs $R_A(1^n)$, does meet the specification.

$R^i_A(1^n)$: Use the Cook-Levin reduction to construct a 3CNF formula F_n that is satisfiable if and only if the following holds:

$$(\exists G : |G| = n)(\exists a)[G(a) = 1 \wedge A(G) \text{ outputs “UNSAT”}].$$

Use search-to-decision on A to search for a SAT assignment to F_n. (Call A on F_n; if it reports “UNSAT” then abort. Try setting a variable x to 0, in F_n. If A reports “SAT” then continue with another variable. If A reports “UNSAT” then flip the value of x to 1. If A still reports “UNSAT” then abort. If A reports “SAT” then continue with another variable.)
There are three possible outcomes:

1. $A(F_n)$ = “UNSAT”. In this case, output F_n.
2. A finds a SAT assignment (G', a'). In this case, output G'.
3. In the search-to-decision reduction, A reaches a subformula F''_n such that $A(F''_n)$ = “SAT”, but when a variable x of F''_n is set 0 (yielding $F''_n[x = 0]$), or set 1 (yielding $F''_n[x = 1]$), A reports “UNSAT” in both cases and we abort.
   Then, A must be wrong on at least one of the three. In this case, $R^1_A(1^n)$ reports F''_n, $R^2_A(1^n)$ reports $F''_n[x = 0]$, and $R^3_A(1^n)$ reports $F''_n[x = 1]$.[4]

Similarly to the previous refuter against algorithms printing SAT assignments, there are a few cases to check.

Case 1: If A outputs a valid SAT assignment on F_n for infinitely many n, we are done.

Case 2: If A reports “UNSAT” on F_n for all but finitely many n, we are done (same analysis as the refuter from the previous subsection).

Case 3: In the remaining case, A only outputs a SAT assignment on F_n for finitely many n, yet A reports “SAT” on F_n for infinitely many n. Thus there are infinitely many n such that A reports the wrong answer on at least one of

$$F''_n, \quad F''_n[x = 0], \quad F''_n[x = 1].$$

Since R^1_A always reports F''_n, R^2_A always reports $F''_n[x = 0]$, and R^3_A always reports $F''_n[x = 1]$, there must be an $i \in \{1, 2, 3\}$ such that for infinitely many n, $R^i(1^n)$ outputs a formula on which A is incorrect!

This concludes the proof of Theorem 3.1.

A Universal Refuter. In fact, for separations like that of SAT against n^k-time algorithms that print SAT assignments, one can construct a *single* $n^{O(k^2)}$-time refuter R which works against all n^k-time algorithms A, infinitely often. The idea is to simply perform "Levin search" [Lev73]: we consider a "meta-algorithm" A' which on a formula F of length n, runs each of the first $\log(n)$ algorithms with n^k-time alarm clocks, and if *any* of the first $\log(n)$ algorithms outputs a SAT assignment to F, then A' outputs it. Now, the refuter $R_{A'}$ for A' is a refuter for all n^k-time algorithms.

3.3 An Aside: Finding Hard Instances for Practical SAT Solvers

Although the above theorem is rather complexity-theoretic in nature, we believe that the ideas could be useful in finding "hard instances" for practical SAT solvers. Let t and m, n be positive integer parameters. For any given solver S, in principle one can build a SAT instance $F_{S,n,t}$ which is satisfiable if and only if there exists a 3CNF G with m clauses and n variables such that G is satisfiable, yet S does not conclude that G is satisfiable within t decisions/backtracks/seconds (whatever notion of time is easiest to encode). If S can solve $F_{S,n,t}$ (indeed, if *any* solver can solve $F_{S,n,t}$) then the solution will produce a hard instance. If no solver can solve $F_{S,n,t}$, then the formula $F_{S,n,t}$ is itself a good candidate for a hard instance. One can imagine holding a "tournament" between a host of practical SAT solvers, feeding various formulas $F_{S',n,t}$ into various solvers S'', to produce many interesting hard instances.

4 Constructive Separations

Theorem 3.1 of the previous section showed that it's possible to efficiently produce "hard inputs" for claimed SAT solvers, assuming $\mathsf{P} \neq \mathsf{NP}$. To generalize this notion, we propose the following definition, letting f be a decision problem and $\mathcal{A}$ be a class of algorithms.

Definition 4.1. We say there is a **P-constructive separation of** $f \notin \mathcal{A}$ if for all algorithms $A \in \mathcal{A}$, there is a polynomial-time algorithm R_A such that, for infinitely many n, $A(R_A(1^n)) \neq f(R_A(1^n))$.

Thus, a P-constructive separation means that for every "weak" algorithm, we can concoct a polynomial-time algorithm that produces bad inputs for the weak algorithm. Theorem 3.1 can then be expressed as:

If $\mathsf{P} \neq \mathsf{NP}$*, then there's a* P*-constructive separation of SAT* $\notin \mathsf{P}$.[5]

[5]Here, we are conflating the class of polynomial-time algorithms with the class of decision problems solvable in polynomial time. My apologies if this bothers you.

So, we know that proving $\mathsf{P} \neq \mathsf{NP}$ will also require us to be able to efficiently construct hard SAT instances for polynomial-time algorithms. A natural question arises:

> *Which complexity lower bound problems* **require** *constructive separations, and which do not?*

Our question is an algorithmic one about the nature of a lower bound. What algorithms are implied by complexity lower bounds? In our paper [CJSW21], we try to make a case that constructive separations are a key to proving major separations between complexity classes. We prove that:

1. Essentially all major separation problems regarding polynomial time will require constructive separations.
2. Making many known lower bounds constructive requires resolving *other* major lower bound problems.

That is, we believe that the property of constructivity (the ability to efficiently refute weak algorithms) lies in the "gap" between lower bounds we know how to prove, and major lower bounds that we'd like to prove. Constructivity is a property we *want* of lower bounds; it is in a sense the opposite of a barrier.

In the next two subsections, we explain our results in more detail.

4.1 Major Complexity Class Separations Will Require Constructive Separations

One of our main theorems is that for *many* choices of complexity classes $\mathcal{C}$ and $\mathcal{D}$, a separation $\mathcal{C} \neq \mathcal{D}$ implies a *constructive* separation of $f \notin \mathcal{C}$ for some function $f \in \mathcal{D}$.

Theorem 4.2 (Informal, see [CJSW21] for details)**.** *For all classes $\mathcal{C} \in \{\mathsf{P}, \mathsf{ZPP}, \mathsf{BPP}\}$ and all classes $\mathcal{D} \in \{\mathsf{NP}, \Sigma_2\mathsf{P}, \mathsf{PP}, \mathsf{PSPACE}, \mathsf{EXP}, \mathsf{NEXP}, \mathsf{EXP}^{\mathsf{NP}}\}$, if $\mathcal{C} \neq \mathcal{D}$, then there is a $\mathcal{C}$-constructive separation of $f \notin \mathcal{D}$, for a "natural" function $f \in \mathcal{D}$.*

The above theorem is informal, in that (a) we have not defined "natural" (but the properties needed hold of most $\mathcal{D}$-complete problems), and (b) we have not defined what it means to be ZPP-constructive or BPP-constructive (but it is a natural randomized notion of constructive separation; see the paper for details).

The above Theorem 4.2 generalizes Theorem 3.1 to hold for many different classes $\mathcal{C}$ and $\mathcal{D}$. A couple of these other cases were known prior to our paper [GST07, DFG13], but most were not. See our paper [CJSW21] for more details.

Theorem 4.2 says that, if we manage to prove a good separation against (randomized) polynomial time, then we are also going to get a constructive separation (in which we can efficiently produce bad inputs). So, we might as well think about constructive methods for proving lower bounds!

In Section 5, we will give one particular example of such a result, proving that if $\mathsf{P} \neq \mathsf{PSPACE}$, then there is a P-constructive separation that true quantified Boolean formulas (TQBF) is not in P.

4.2 Making Known Lower Bounds Constructive Implies Strong Circuit Lower Bounds

In the second part of the paper [CJSW21], we show how constructive separations for several different well-known lower bounds (based on information-theoretic arguments) would turn out to imply breakthrough lower bounds. That is, "constructivizing" any one of many known lower bounds would have actually have rather significant lower bound consequences. The three regimes we consider are:

- (randomized) streaming lower bounds,
- query complexity lower bounds, and
- superlinear-time one-tape Turing machine lower bounds.

Streaming lower bounds and query complexity lower bounds are generally considered to be well-understood, and certain superlinear-time lower bounds against one-tape Turing machines have been known for decades [Hen65, Maa84]. Surprisingly, we show in [CJSW21] that making these separations constructive would imply breakthrough separations such as $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$, or even $\mathsf{P} \neq \mathsf{NP}$ (if the algorithm producing bad inputs is restricted enough). Here, we briefly outline our results in more detail.

Constructivizing Streaming Lower Bounds Implies Breakthroughs. In the streaming algorithm setting, an algorithm (storing little space) must pass through all bits of the input stream exactly once, and output a good answer when the stream ends. A host of problems are well-known to be *unconditionally* hard for randomized streaming algorithms that use a small amount of working space, and these lower bounds typically follow from communication complexity lower bounds. For a canonical example, in the Set-Disjointness (DISJ) communication problem, Alice is given an n-bit string x, Bob is given an n-bit string y, and the goal is to determine whether or not the inner product $\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$ is nonzero, with minimal communication. We can think of DISJ as a function that takes (x, y) and outputs a Boolean value, in the natural way. The randomized communication complexity lower bounds for DISJ [KS92, Raz92, BJKS04] directly imply that

no $n^{1-\varepsilon}$-space randomized streaming algorithm can correctly decide the simple language

$$L_{\mathsf{DISJ}} = \{xy \in \{0,1\}^\star \times \{0,1\}^\star \mid |x| = |y|, \mathsf{DISJ}(x,y) = 1\}.$$

Therefore, every (randomized) streaming algorithm using only $n^{o(1)}$ workspace must fail to correctly decide L_{DISJ} on some inputs. We show that efficient refuters against streaming algorithms attempting to solve *any* NP problem (not just L_{DISJ}) would imply a breakthrough separation against *general* randomized algorithms.

Theorem 4.3 (Informal)**.** *For every language $L \in \mathsf{NP}$, a P^{NP}-constructive separation of L from uniform randomized streaming algorithms using $O(\log n)^{\omega(1)}$ space implies $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$.*

Essentially every lower bound proved against streaming algorithms in the literature holds for some problem whose decision version is in NP. Theorem 4.3 shows if *any* such lower bound can be made constructive, even if it takes P^{NP} to produce the bad inputs, then $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{BPP}$ follows, a longstanding (embarassing) open problem in complexity theory. And if we could replace "P^{NP}-constructive" with "P-constructive", we would prove that $\mathsf{EXP} \neq \mathsf{BPP}$. The upshot is that the counterexample inputs printed by any such refuter algorithm must encode a function that is actually hard for *general* randomized algorithms.

Constructivizing Lower Bounds for One-Tape Turing Machines Implies Breakthroughs. It has been known at least since Maass [Maa84] that nondeterministic one-tape Turing machines require $\Omega(n^2)$ time to decide even simple problems such as PALINDROMES. However, the lower bounds (and others like it) are proved by non-constructive counting arguments. (One can also use $\Omega(n)$ lower bounds on the nondeterministic communication complexity of the EQUALITY function, to prove such lower bounds.) Similarly to the previous setting, we show that if there is a P^{NP} refuter that can produce bad inputs for a given one-tape (nondeterministic) Turing machine, then E^{NP} ($2^{O(n)}$ time with an NP oracle) requires exponential-size circuit complexity. This in turn would imply $\mathsf{BPP} \subseteq \mathsf{P}^{\mathsf{NP}}$, a breakthrough simulation of randomized polynomial time. The theorem we prove is very general, and applies to many more problems than just PALINDROMES:

Theorem 4.4. *For every language L computable by a nondeterministic $n^{1+o(1)}$-time RAM, a P^{NP}-constructive separation of L from nondeterministic $O(n^{1.1})$-time one-tape Turing machines implies $\mathsf{E}^{\mathsf{NP}} \not\subset \mathsf{SIZE}[2^{\delta n}]$ for some constant $\delta > 0$.*

Let us demonstrate how Theorem 4.4 works, with the specific example of PALINDROMES. Our approach can be readily generalized to a proof of Theorem 4.4.

Proof of Theorem 4.4. (Sketch) Let $\varepsilon > 0$ be arbitrarily small. Our goal is to construct a nondeterministic one-tape Turing machine M that takes $N^{1+c\varepsilon}$ time for a universal constant $c \geq 1$, such that M has the following properties:

- For every n, M **accepts** every palindrome $xx^R \in \{0,1\}^{2N}$ of length $2N = 2^{n+1}$ such that x, construed as the length-2^n truth table of a function from n bits to 1 bit, has circuit complexity at most $2^{\varepsilon n}$.
- M **rejects** every string $y \in \{0,1\}^{2N}$ that is not a palindrome and has circuit complexity at most $2^{\varepsilon n}$, when construed as a function from $n+1$ bits to 1 bit.
- M **rejects** every string of odd length (for simplicity, we only consider even-length palindromes, of the form xx^R).

That is, M correctly decides PALINDROMES on all strings of circuit complexity at most $2^{\varepsilon n}$. This Turing machine M exists unconditionally: its correctness does not rely on any assumptions, and we will describe how to construct M later.

Given that such an M exists, for sufficiently small $\varepsilon > 0$, M runs in time $o(N^2)$. Therefore it must fail to correctly solve PALINDROMES on infinitely many inputs. Consider any P^{NP} algorithm R that, on the input 1^N, prints an input $z_N \in \{0,1\}^N$ on which M fails to decide PALINDROMES, for infinitely many N.

The properties of M imply that the string z_N does *not* have circuit complexity at most $2^{\varepsilon n}$. Therefore, there is a P^{NP} procedure R that, on infinitely many 1^N, prints the truth table of a function with circuit complexity greater than $2^{\varepsilon n}$. We can produce a function $f \in \mathsf{E}^{\mathsf{NP}}$ with circuit complexity greater than $2^{\varepsilon n}$, as follows.

> $f(x) :=$ Let $n = |x|$. Run $R(1^{2^n})$, producing a string z_{2^n} of length 2^n.
> Let $y_1, \ldots, y_{2^n}$ be the list of all n-bit strings in lex order.
> Output the i-th bit of z_{2^n}, where $x = y_i$.

Observe that the truth table of f, on inputs of length n, is precisely the string z_{2^n}. Therefore, f requires circuit complexity greater than $2^{\varepsilon n}$, for infinitely many n.

Now we turn to the construction of the desired nondeterministic Turing machine M; here, we just sketch how it works. First, M rejects its input z immediately if $|z|$ is odd; this can be easily checked in linear time in a standard way. Note that M can also compute $|z|$ directly in $N \cdot \mathrm{poly}(\log N)$ time, by "dragging along" an $O(\log n)$-bit counter on its single tape as it streams through the bits of z, using a larger alphabet to store the content of the counter. Next, given that $|z| = 2N = 2^{n+1}$, M nondeterministically guesses a circuit C of size at most N^{ε}, using $O(N^{\varepsilon} \log N)$

bits of nondeterminism. The intention is that $z = xx^R$ and C is a circuit such that $C(i)$ outputs the i-th bit of x. The machine M can check this as follows. First, by "dragging along" the description of C as it reads the bits of x, M can verify that $C(i)$ outputs the i-th bit of x. Evaluating C on an input takes no more than $O(N^{c\varepsilon})$ time per bit of x for a fixed constant $c \geq 1$, which is in total $O(N^{1+c\varepsilon})$ time over all bits of x. Finally, M can verify that z is a palindrome by using C to check that the first bit of z matches the last bit, the second bit of z matches the next-to-last bit, and so on, using evaluations of C to check the first half of z. All this takes no more than $O(N^{1+c\varepsilon})$ time, and M accepts if and only if all bit checks pass.

It is easy to see that if z has circuits of size at most N^{ε}, and z is a palindrome, then the computation path of M that guesses a small circuit correctly will indeed accept z. However, if z is not a palindrome, then no computation path of M will accept z. □

Constructivizing Certain Query Lower Bounds Implies Breakthroughs. Even obtaining efficient refuters for query lower bounds on the "coin problem" [BV10] would imply strong lower bounds. We define the problem $\mathsf{Promise\text{-}MAJORITY}_{n,\varepsilon}$ for a parameter $\varepsilon < 1/2$, as follows:

> $\mathsf{Promise\text{-}MAJORITY}_{n,\varepsilon}$: *Given an input* $x \in \{0,1\}^n$*, letting* $p = \frac{1}{n}\sum_{i=1}^{n} x_i$*, distinguish between the cases* $p < 1/2 - \varepsilon$ *or* $p > 1/2 + \varepsilon$*.*

It is well-known that every randomized query algorithm needs $\Theta(1/\varepsilon^2)$ queries to solve $\mathsf{Promise\text{-}MAJORITY}_{n,\varepsilon}$ with constant success probability (uniform random sampling is the best one can do). That is, any randomized query algorithm making $o(1/\varepsilon^2)$ must make mistakes on some inputs, with high probability. We can show that constructing efficient-enough refuters for this simple sampling lower bound would imply $\mathsf{P} \neq \mathsf{NP}$. Please see the paper [CJSW21] for details.

Theorem 4.5 (Informal)**.** *A "uniform* AC^0*" constructive separation of the problem* $\mathsf{Promise\text{-}MAJORITY}_{n,\varepsilon}$ *from all randomized query algorithms that make only* $o(1/\varepsilon^2)$ *queries and run in* $\mathrm{poly}(1/\varepsilon)$ *time, implies* $\mathsf{P} \neq \mathsf{NP}$*.*

Finally, we also show that constructive separations for the Minimum Circuit Size Problem (MCSP) against AC^0 circuits would also imply unexpected breakthrough lower bounds. (Informally, the Minimum Circuit Size Problem (MCSP) is the problem of determining the circuit complexity of a given 2^n-bit truth table.) As above, it is known that MCSP does not have small AC^0 circuits [ABK+02], and the question is whether there is a *constructive* separation. We refer the reader to the paper [CJSW21] for details.

5 Proving Polynomial-Time Lower Bounds for TQBF Requires Constructivity

To give one example of how Theorem 3.1 can be extended beyond $\mathsf{P} \neq \mathsf{NP}$ to other major open problems about polynomial time, I will demonstrate here how $\mathsf{P} \neq \mathsf{PSPACE}$ implies a P-constructive separation of TQBF $\notin \mathsf{P}$. That is, assuming $\mathsf{P} \neq \mathsf{PSPACE}$, given any poly-time algorithm A that attempts to decide the PSPACE-complete problem TQBF (true quantified Boolean formulas), we can *efficiently* find QBFs on which A fails. Notice, since we are now talking about a lower bound on a PSPACE-complete problem, there is no way we can definitively *check* all answers to the algorithm A in polynomial time: unless $\mathsf{NP} = \mathsf{PSPACE}$, we cannot even give short proofs that a QBF is true or false. Nevertheless, we can still locate bad inputs for A. I have chosen the particular example of refuters for $\mathsf{P} \neq \mathsf{PSPACE}$ with TQBF, because one can apply similar ideas as in the refuter for $\mathsf{P} \neq \mathsf{NP}$ with SAT. In the paper [CJSW21], we use more sophisticated ideas to obtain refuters for many more complexity class separation problems.

In particular, assume $\mathsf{P} \neq \mathsf{PSPACE}$, and let A be an n^k-time that attempts to decide TQBF by outputting "true" or "false" on every encoding of a Boolean formula. For a quantified Boolean formula (QBF) F, let "F" denote its encoding in binary. Say that A is *inconsistent* on F if:

- Either $F = (\forall x)G(x)$ for a QBF G with one free variable, and $A(\text{“}F\text{”}) \neq A(\text{“}G(0)\text{”}) \wedge A(\text{“}G(1)\text{”})$,
- or $F = (\exists x)G(x)$ for a QBF G with one free variable, and $A(\text{“}F\text{”}) \neq A(\text{“}G(0)\text{”}) \vee A(\text{“}G(1)\text{”})$.

Of course, $(\forall x)G(x)$ is true if and only if $G(0)$ and $G(1)$ are true, and $(\exists x)G(x)$ is true if and only if $G(0)$ or $G(1)$ is true. Thus, A is inconsistent when it fails to satisfy this basic property of quantifier semantics, and A must be incorrect on at least one of three QBFs.

We can define a refuter against A, as follows. Let $i \in \{1, 2, 3\}$.

> $R_A^i(1^n)$: Construct a formula F_n encoding the property
>
> $$(\exists \text{ QBF “}G\text{”}, |G| = n)[A \text{ is inconsistent on } G].$$
>
> If $A(F_n)$ outputs "false", then output F_n.
> Otherwise, $A(F_n)$ outputs "true". As in Theorem 3.1, use search-to-decision to try to construct a QBF G on which A is inconsistent.
> If the search-to-decision fails (we reach a formula that A declared "true" but its

two subformulas are declared "false"), then as in Theorem 3.1, we have a set of three QBFs such that A is incorrect on at least one of them; R^i will output the ith such formula.
If the search-to-decision succeeds, then A is inconsistent on G, and we obtain three QBFs (G and two other "subformulas") such that A is incorrect on at least one of them; R^i will output the ith such formula.

Analogously as in Theorem 3.1, we can argue that there is always some $i \in \{1, 2, 3\}$ such that for infinitely many n, $R^i_A(1^n)$ outputs a QBF on which A is incorrect.

6 Conclusion

We hope this article has encouraged the reader to think more about how algorithmic methods are actually necessary for proving strong complexity lower bounds.

Our work leaves open several interesting directions. For example, it is not entirely clear how to extend our results to separations with complexity classes within P. For example, let L be a decision problem which is complete for P under logspace reductions. If L is not decidable in $\mathsf{LOGSPACE}$, does a "logspace-constructive" separation of $L \notin \mathsf{LOGSPACE}$ follow? What about constructive separations for non-uniform complexity classes, such as $\mathsf{P/poly}$? Should we expect a constructive separation of $\mathrm{SAT} \notin \mathsf{P/poly}$, and if so, what properties should the algorithms/circuits have?

Finally, in this invited article, I have not provided an overview of all relevant prior work. Such an overview can be found in our paper [CJSW21].

Acknowledgments

I am grateful to my co-authors, Lijie Chen, Ce Jin, and Rahul Santhanam, for collaborating with me on this project. I also thank Michal for inviting me to write about this work, and his patience with me while I was finishing this article.

References

[AB18] Baris Aydinlioglu and Eric Bach. Affine relativization: Unifying the algebrization and relativization barriers. *ACM Trans. Comput. Theory*, 10(1):1:1–1:67, 2018.

[AB19] Baris Aydinlioglu and Eric Bach. Corrigendum to affine relativization: Unifying the algebrization and relativization barriers. *ACM Trans. Comput. Theory*, 11(3):16:1, 2019.

[ABK+02] Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM J. Comput.*, 35(6):1467–1493, 2006. Preliminary version in FOCS'02.

[Ajt02] Miklós Ajtai. Determinism versus nondeterminism for linear time rams with memory restrictions. *Journal of Computer and System Sciences*, 65(1):2–37, 2002.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, 2009.

[BC82] Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11(2):287–297, 1982.

[Bea91] Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991.

[BGS75] Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the P =?NP question. *SIAM J. Comput.*, 4(4):431–442, 1975.

[BJKS04] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

[BJS01] Paul Beame, Thathachar S Jayram, and Michael Saks. Time-space tradeoffs for branching programs. *Journal of Computer and System Sciences*, 63(4):542–572, 2001.

[BSSV03] Paul Beame, Michael Saks, Xiaodong Sun, and Erik Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *JACM*, 50(2):154–195, 2003.

[BV10] Joshua Brody and Elad Verbin. The coin problem and pseudorandomness for branching programs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 30–39, 2010.

[CHO+20] Lijie Chen, Shuichi Hirahara, Igor Carboni Oliveira, Ján Pich, Ninad Rajgopal, and Rahul Santhanam. Beyond natural proofs: Hardness magnification and locality. In *11th Innovations in Theoretical Computer Science Conference, ITCS*, pages 70:1–70:48, 2020.

[CJSW21] Lijie Chen, Ce Jin, Rahul Santhanam, and R. Ryan Williams. Constructive separations and their consequences. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 646–657. IEEE, 2021.

[DFG13] Shlomi Dolev, Nova Fandina, and Dan Gutfreund. Succinct permanent is *NEXP*-hard with many hard instances. In *Algorithms and Complexity, 8th International Conference, CIAC 2013. Proceedings*, volume 7878 of *Lecture Notes in Computer Science*, pages 183–196. Springer, 2013.

[GST07] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. If NP languages are hard on the worst-case, then it is easy to find their hard instances. *Computational Complexity*, 16(4):412–441, 2007.

[Hen65] F. C. Hennie. Crossing sequences and off-line turing machine computations. In *6th Annual Symposium on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, USA, October 6-8, 1965*, pages 168–172. IEEE Computer Society, 1965.

[IK20] Christian Ikenmeyer and Umangathan Kandasamy. Implementing geometric complexity theory: on the separation of orbit closures via symmetries. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 713–726. ACM, 2020.

[IKK09] Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. An axiomatic approach to algebrization. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC*, pages 695–704. ACM, 2009.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

[KS92] Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Math.*, 5(4):545–557, 1992.

[Lev73] Leonid Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[Maa84] Wolfgang Maass. Quadratic lower bounds for deterministic and nondeterministic one-tape turing machines (extended abstract). In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 401–408. ACM, 1984.

[Mul07] Ketan Mulmuley. Geometric complexity theory VI: the flip via saturated and positive integer programming in representation theory and algebraic geometry. *CoRR*, abs/0704.0229, 2007.

[Mul10] Ketan Mulmuley. Explicit proofs and the flip. *CoRR*, abs/1009.0246, 2010.

[Mul12] Ketan Mulmuley. The GCT program toward the *P* vs. *NP* problem. *Commun. ACM*, 55(6):98–107, 2012.

[MW19] Dylan M. McKay and Richard Ryan Williams. Quadratic time-space lower bounds for computing natural functions with a random oracle. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 56:1–56:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Ner58] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[Raz92] Alexander A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

[RY20] Anup Rao and Amir Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020.

[Sip06] Michael Sipser. *Introduction to the theory of computation (2nd edition)*. Thomson Course Technology, 2006.

[Yao89] Andrew Chi-Chih Yao. Circuits and local computation. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 186–196. ACM, 1989.

THE LOGIC IN COMPUTER SCIENCE COLUMN

BY

YURI GUREVICH

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

BEATCS no 139

THE UMBILICAL CORD OF FINITE MODEL THEORY

Yuri Gurevich
Computer Science & Engineering, University of Michigan

Abstract

Model theory was born and developed as a part of mathematical logic. It has various application domains but is not beholden to any of them. A priori, the research area known as finite model theory would be just a part of model theory but didn't turn out that way. There is one application domain — relational database management — that finite model theory had been beholden to during a substantial early period when databases provided the motivation and were the main application target for finite model theory.

Arguably, finite model theory was motivated even more by complexity theory. But the subject of this paper is how relational database theory influenced finite model theory.

This is NOT a scholarly history of the subject with proper credits to all participants. My original intent was to cover just the developments that I witnessed or participated in. The need to make the story coherent forced me to cover some additional developments.

1 Prelude

Q[1]: How come finite model theory is computer science while model theory is mathematics?

A[2]: Let me think. Model theory is a part of mathematical logic which was born and developed as a part of foundations of mathematics. Finite model theory (FMT) was developed much later. It did not exist as a separate research area before the 1970s. FMT was developed primarily by computer scientists. It was — and is — much influenced by complexity theory which is viewed traditionally as theoretical computer science (though mathematicians often view it as mathematics). Also, during a substantial early period FMT was much influenced by (relational) database theory (DBT), bona fide computer science. I doubt that FMT would have become a recognizable research area without that DBT link. In that sense, the DBT link is the umbilical cord of FMT.

Q: Tell me about that influence of DBT on FMT.

A: I'll try my best. But let me make two reservations. I am not a DBT expert and, more importantly, I am a wrong person to render a scholarly history of the subject complete with proper credits to all participants. It takes a special talent to give a scholarly account of a research field; I don't have such talent. I will tell you primarily about the developments that I witnessed or participated in.

[1] Quisani is my former student.
[2] The author

Q: One last question before you start your story. I presume that logicians studied finite models before the 1970s?

A: Yes, they did. Let me start with that logic prehistory of FMT.

2 Prehistory

Finite structures were relevant to humanity long before the notion of structure was introduced, in fact before the notion of theorem was introduced. Consider for example the fact that a finite set can be counted in any order, always giving the same result. The special cases of this fact — for 2-element sets, 3-element sets, etc. — must have been known very early. The Greeks probably knew the general fact before the notion of theorem was introduced.

While model theory was studied (e.g. by Skolem and Löwenheim) in the beginning of the 20th century, finite model theory arguably became a separate research area in the 1970s, and the term "finite model theory" was coined only toward the end of the century.

As the rest of this section shows, finite model theory was studied and used before it became a separate research area.

2.1 Boris A. Trakhtenbrot

In the paper [45] Boris Trakhtenbrot proved that finite satisfiability is not decidable. There is no algorithm that, given a first-order sentence φ, decides whether φ has a finite model.

It follows that finite validity is not recursively enumerable, that is the set of finitely valid (valid on all finite models) first-order sentences is not recursively enumerable. Indeed, let S be the set of finitely valid sentences and S' the complement of S. Sentences in S' have finite counterexamples. It follows that S' is recursively enumerable. If S were recursively enumerable as well, then the following (not very feasible) algorithm would decide, given a sentence φ, whether it is in S or in S'. Keep enumerating a list of S sentences and a list of S' sentences until φ occurs in one of the lists.

The failure of recursive enumerability is important. It means that there is no reasonable deductive system for finite validity. This contrasts with the situation for general validity (validity in arbitrary, not necessarily finite, structures) where there is such a deductive system.

In the paper [46] published in 1953, Trakhtenbrot strengthened his result. The set of finitely satisfiable sentences and the subset of unsatisfiable sentences are recursively inseparable: There is no recursive set that contains one of the two sets and is disjoint from the other.

Since FMT is typically considered as a part of theoretical computer science, it may be fitting to note that Trakhtenbrot was also a pioneer of computer science and had many good students (Figure 1 shows him with a few of them). See also [2] in this connection.

Figure 1: Boris Trakhtenbrot with some of his former students in 1992. Left to right: Vladimir Sazonov, Janis Barzdins, Rusins Freivalds, Alexandre Dikovsky, Boris Trakhtenbrot, Mars Valiev, Miroslav Kratko, and Valery Nepomnyaschy.

I first met Trakhtenbrot in the mid-1960s in the Novosibirsk Academgorodok [50] where he was running a lively seminar on the nascent computer science. In spite of the difference in age, we became friends. I used to come to the Akademgorodok from time to time to give a talk at the Algebra & Logic seminar run by academician Anatoly Maltsev. On the same occasions, I would attend Trakhtenbrot's seminar and talk to Trakhentbrot. I remember how he introduced time and space complexity (by means of so-called "signalizing functions") independently of his Western colleagues. I could have become a computer scientist much earlier than I did. But, at the time, I was moving from algebra to logic and it was logic that fascinated me.

2.2 Dana Scott, Patrick Suppes, and William W. Tait

(a) Dana S. Scott

(b) William W. Tait

(c) Patrick C. Suppes

It is easy to see that a purely universal sentence, that is a sentence of the form

$$\varphi = \forall x_1 \forall x_2 \ldots \forall x_n \Phi(x_1, x_2, \ldots, x_n)$$

where Φ is quantifier-free, is preserved by submodels: If $A \models \varphi$ and B is a submodel of A then $B \models \varphi$. According to the Łoś-Tarski theorem [10, Theorem 5.2.4], there is a converse: Any sentence that is preserved by substructures is equivalent to a universal sentence.

In the 1958 paper [41], Dana Scott and Patrick Suppes conjectured that the Łoś-Tarski theorem remains valid when only finite structures are taken into account. In the 1959 paper [42], William W. Tait disproved their conjecture. In retrospect, his counterexample is rather simple. Essentially it is a first-order sentence expressing that a linear order with a first element has also a last element. Every finite model has the property, and so the property is preserved by submodels of finite models. But an infinite model may violate the property in spite of being a submodel of one that has the property; the smallest example is the linear order ω of natural numbers as a submodel of $\omega + 1$.

But there was more in the Scott-Suppes article than the conjecture mentioned above. We will return to their paper later in this section.

2.3 Yuri Glebsky and his students

In the 1966 International Congress of Mathematicians in Moscow, Yuri Glebsky announced an unexpected zero-one law for first-order logic [23].

Consider a finite set V of relation symbols and restrict attention to models of vocabulary V with the universe of the form $\{1, 2, \ldots, n\}$. For each first-order sentence φ in vocabulary V, let $B(n)$ be the total number of n-element models and $A(n)$ the number of such models satisfying φ. The zero-one law states that, for every V and φ as above, $\lim_{n\to\infty} A(n) \,/\, B(n)$ exists and is equal to either 0 or 1.

Figure 3: Yuri V. Glebsky

In 1969, Glebsky and three of his students — Dmitry I. Kogan, Mark I. Liogonky, and Vladimir A. Talanov — published a proof of the zero-one law [24]. In 1972, the paper was duly translated to English but wasn't noticed in the West. By chance, I knew paper [24]. This wasn't because I was so scholarly; that has never been my forte. But, in 1970, I served as an official opponent at the public defense of the PhD thesis of Liogonky. In the thesis, Liogonky proved that the 0-1 law remains valid if the structures are counted up to isomorphism

In 1972, Ron Fagin announced the 0-1 law, which he had discovered independently in the meantime [15]. He gave an elegant proof of the law in his thesis [16] and published it in [21]. A number of years later, I met Fagin and told him about Glebsky. Fagin was surprised, but he found the English translation of the 1969 paper, and the matter was settled.

In 1977 Glebsky drowned in the Volga trying to save his son. His students admired him, and his untimely death devastated them [39].

2.4 Other relevant results

Q: To what extent is the collection of relevant "prehistoric" results above exhaustive?

A: It is definitely not exhaustive. There are many pre-1970 results that are relevant to some degree. For example, the recursive inseparability result of Trakhtenbrot provoked more subtle recursive inseparability results in algebra and logic. In the book [8], an appropriate reduction theory is presented, and it is proven that, for many natural classes K of first-order sentences, the set of finitely satisfiable K sentences and the set of unsatisfiable K sentences are recursively inseparable.

Also there is at least one very relevant result that is missing above. Commenting on a draft of this paper, Erich Grädel reminded me of the Büchi-Elgot-Trakhtenbrot Theorem [5, 11, 47], which he views as the first descriptive complexity result. The theorem states that a set of strings in any finite alphabet is accepted by a finite-state automaton if and only if it is definable in monadic second-order logic (MSO). It is certainly the first descriptive complexity result that I know.

3 Database theory

Edgar Frank "Ted" Codd wouldn't get into a model theory hall of fame. His famous theorem, known simply as Codd's Theorem [52], is good model theory, but it wasn't real news to logicians as we explain below. Yet he is a hero of our story. He invented the relational model for database management, which facilitated the birth of finite model theory as a separate research area.

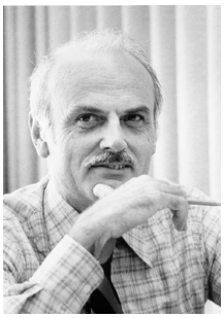


Figure 4: Edgar F. Codd

It may be hard for a theorist with no practical experience to appreciate Codd's contribution. Codd observed that databases can be adequately represented by (first-order) structures [13, 14]. But structures had originally been designed to represent data, or knowledge.

Q: Designed by whom?

A: By architects of what is now called model theory, starting from Löwenheim and Skolem, including Gödel and Tarski, especially Tarski; see [43] for example.

Q: Mathematical logic was developed within the framework of foundations of mathematics. Only mathematical data might have been of interest to logicians.

A: I don't know what you mean by mathematical data. Logicians certainly did not restrict attention to numerical data. Any data whatsoever may be involved in a mathematical problem and thus become "mathematical data" of interest to logicians.

Besides, logicians have been interested in problems beyond foundations of mathematics. The Scott-Suppes paper mentioned in §2.2 is on the foundations of theories of measurements. The authors foresaw relational databases:

> "From an abstract standpoint, a set of empirical data consists of a collection of relations between specified objects ... we treat sets of empirical data as being (finitary) relational systems" [41, §1]. ◁

In article [14], Codd defined a variable-free *relational algebra* and a *relational calculus*. The latter is a form of first-order logic, with propositional connectives and quantifiers, which is convenient for his purposes. He also proved a theorem, Codd's Theorem, establishing that relational algebra has essentially the expressive power of first-order logic.

Q: Why essentially and not literally?

A: Database queries should be safe in the appropriate technical sense [1, §5.3]. For example, no query should express the complement of a finite subset of an infinite domain. Codd's Theorem establishes that relational algebra has the expressive power of the safe fragment of first-order logic.

But Codd's Theorem wasn't real news for logicians either. An algebraic form of first-order logic, using so-called cylindric algebras, had been developed earlier by Tarski and his collaborators [12, 44, 32].

Q: If logicians knew all that, then what made Codd's work important?

A: In one word, engineering. There is a huge difference between treating sets of empirical data as relational systems and practical database management. Codd's approach was practical, and the result was a model of database management superior to all competitors. He received the Turing Award in 1981.

Q: What are the benefits of the relational database management?

A: One is simplicity; all information is represented by data values in addition to a few relation symbols. Another important benefit is clear and unambiguous semantics. Yet another important benefit is that the relational approach admits high level query languages.

Q: If we stick to theory, rather than practice, was there anything new in Codd's approach?

A: Yes, there was. Normally, in logic and in mathematics in general, structures are static. You study a particular structure, e.g. the field of real numbers, or a class of structures, e.g. commutative groups. But databases evolve and in that sense they are dynamic. For example, consider the salary database for some organization. New employees are hired, some employees leave the organization, some get raises, etc. To incorporate evolution, the language of databases should allow us to update databases in various ways.

Q: Did you know Codd?

A: No; I wish I did. Codd wrote his PhD thesis at the University of Michigan, but that was in the 1960s, long before I arrived there. But I heard plenty about Ted Codd from his advisor John Holland. John told me that Codd compromised his health trying to

promote the relational approach at IBM. His fight at IBM is described in many places, even in the Wikipedia article [51].

4 The beginning

Q: After all these preliminaries are you ready to discuss FMT per se? If yes, start with the beginning.

A: I am ready to discuss FMT but starting with the beginning is a challenge. Until 1982, when I joined the University of Michigan and officially became a computer scientist, I was a mathematician and very different things were on my mind, primarily model theory. So, preparing for this discussion, I exchanged letters with experts. In particular, I posed two questions to Ron Fagin:

1. What do you count as the beginning of FMT? Is it your thesis?
2. Who coined the term "finite model theory"? And when?

"Although Trakhtenbrot's Theorem was the first important result," replied Ron on Nov. 9 2022, "[I] do feel that my Ph.D. thesis launched FMT as a field. Since I am biased, I called Moshe [Vardi] (whom I am cc-ing). He says he completely agrees that Trakhtenbrot had the first major result and that my thesis started FMT as a field."

A couple of days later, Moshe sketched for me his view of the early history.

> "In the 1970s, there are three independent lines of work:
>
> a. 1970: Codd's paper — start of relational DBT.
> b. 1973: Fagin's dissertation — 'Contributions to the model theory of finite structures', where he proved Σ^1_1 = NP, the 0-1 Law, and that reachability is not definable in [existential] MSO.
> c. 1980: Immerman's dissertation — 'First order expressibility as a new complexity measure.'
>
> The work in the early 1980s built on these three legs, even though it was not explicitly FMT. For example, the 1982 papers by Immerman and by Vardi built on these legs, but did not emphasize the finiteness feature. Your papers in the 1980s raised the issue of finiteness and helped FMT become a well-defined field." — Moshe Vardi Dec. 17 2022

Q: Does this sound reasonable to you?

A: Totally. Ron's thesis [16] was truly pioneering; the results are published in five papers: [17, 18, 19, 20, 21]. I have already mentioned his elegant proof of the 0-1 law [21]. Another major result is that connectivity of undirected graphs is not expressible in existential monadic second-order logic [18]; since disconnectedness is expressible in the logic, the collection of expressible properties is not closed under complementation. The result known as Fagin's Theorem asserts that the expressive power of existential second-order logic on finite structures is exactly that of the NP complexity class [17]. I mentioned above that Büchi-Elgot-Trakhtenbrot Theorem was the first descriptive complexity result. But Fagin's Theorem started descriptive complexity theory in earnest. It wasn't about finite automata but about one of the most important complexity classes, NP.

As far as relational database theory is concerned, the timing of Ron's thesis was perfect. Codd's 1970 and 1972 papers [13] and [14] respectively laid foundations for the relational model of database management, which provided motivation and application for the nascent finite model theory. In a different letter, of Dec. 17 2022, Ron wrote to me: "[W]hen I transferred from IBM Watson to IBM San Jose in 1975, Ted Codd became my mentor, which led me to write papers explicitly on relational data base theory."

In the 1970s and early 1980s there was much DBT work of relevance to FMT. I know several experts in DBT who are also experts in FMT — Serge Abiteboul, Phokion Kolaitis, Victor Vianu, and Moshe Vardi; they can tell you all about that. The book [1] is a good DBT textbook for logically inclined. One DBT article of influence on FMT that will play a role later in our story is the article [9] by Ashok Chandra and David Harel.

In 1980 Neil Immerman defended his PhD thesis [33]; the results were published in [34, 35]. I asked Neil what led him to FMT.

> "I was inspired by Fagin's characterization of NP and his proof that connectivity is not in monadic SOE [existential monadic second-order logic]," replied Neil, "I felt that lower bounds on full arity SO was going to be too hard, but I thought that first-order lower bounds would be more tractable. This was what inspired me to work in my thesis area developing what I called "First-Order Expressibility". Later Hartmanis [Neil's advisor] suggested, "Descriptive Complexity" as a more appealing name." — Neil Immerman Dec. 20 2022

In STOC 1982, Immerman and Vardi presented their versions of what became known as the Immerman-Vardi Theorem; they showed that, on ordered structures, the least fixed point extension of first order logic captures polynomial time, widely viewed as the most important complexity class [36, 48]. Immerman went on to become a champion, maybe I should say the champion, of descriptive complexity theory, as both a contributor and an evangelist; see his book [37] on the subject.

Q: What about your contributions that Vardi mentioned?

A: I'll speak more about that in the next section. Here let me mention the very first result: Under a natural interpretation over finite domains, a function is primitive recursive if and only if it is logspace computable, and it is general recursive if and only if it is polynomial time computable [27].

Q: Did Fagin reply to your second question?

A: Yes. He wrote: "I don't know (and neither does Moshe) who coined the term 'finite model theory'. Moshe says he will look into that." Moshe wrote to us the same day and surprised me. "The first paper that I found that used FMT in its title is [30]. Ron followed up with [22]. So it seems that Yuri deserves the credit for coining the term[3]."

Q: Did you coin the term?

A: Not consciously. Contrary to Moshe and Ron, I did not belong to the database community. I came from model theory, and the term "finite model theory" suggests itself to model theorists working with finite models. And it is possible of course that the term was used earlier though not in the titles of papers.

[3] I took the liberty of replacing URLs with references to the bibliography

5 The importance of being finite

Q: How did you arrive to relational databases?

A: At FOCS 1982, Moshe Vardi presented "On decomposition of relational databases" where he made essential use of Beth's Definability Theorem [49]. I worried that the theorem may not survive the restriction to finite structures and I asked him whether his databases may be infinite. He said yes.

Q: Infinite databases! This is crazy.

A: And a little expensive.

Q: Right.

A: I thought that first-order logic may not be the best fit for databases.

Q: What gave you this idea?

A: My prior experience with ordered abelian groups (OAGs). In my PhD thesis [25], I proved the decidability of the first-order theory of OAGs, solving a problem of Tarski. But the OAG algebra is rarely first-order. So I extended the decidability to the monadic second-order theory of OAGs where the set variables range over convex subgroups [26]. The expanded OAG theory essentially subsumed the OAG literature of the time while the decidability proof became simpler. Obviously first-order logic wasn't the best fit for OAGs.

Returning home after the FOCS conference, I constructed counterexamples for some famous classical logic theorems, including Beth's Definability, in the finite case, that is the case where only finite structures are taken into account [28]. In the process, I discovered that William Tait had given the counterexample for the Łoś-Tarski theorem described in §2.2.

Q: Was there a classical theorem that you expected to fail in the finite case but couldn't find a counterexample?

A: Yes, Lyndon's theorem [38]. It states that a first-order sentence $\varphi(P)$ is monotone in P if and only if it is equivalent to a sentence $\psi(P)$ where P has only positive occurrences. Eventually Miki Ajtai and I proved that it fails in the finite case [4].

Q: Why did Codd strive to achieve the expressivity of first-order logic.

A: That was rather natural, I guess. First-order logic is relatively expressive. A great many relational queries are expressible in first-order logic. Besides, first-order logic is the logic of standard logic textbooks. What else was there? Propositional logic is way too weak while second-order logic is way too expressive.

Q: But first-order logic was developed in the framework of foundations of mathematics, I understand, where infinite structures are indispensable.

A: This is true. Also, in the foundational framework, it is important that first-order logic admits an adequate deductive system, where a sentence is provable if and only it is valid. In fact, first-order logic was introduced as a deductive system. It was obvious that all provable sentences are valid. Gödel's completeness theorem established that all valid sentences are provable. As we saw in §2.1, finite validity is not recursively enumerable. Thus there is no logic calculus where a sentence is provable if and only it is valid on all finite structures.

Q: In reality, had first-order expressibility proved insufficient?

A: Yes. There is no first-order query (or even existential monadic second-order query) that, given an arbitrary relation R returns the transitive closure of R [18], and transitive closure is needed in database management [53].

Q: If first-order expressivity isn't sufficient, what can you do? Maybe you can carve out an appropriate fragment of second-order logic.

A: Yes, but it is more convenient to think in different terms, like transitive closure or least fixed point operators [3].

Q: But what guides you?

A: Ah, this is the key point. The answer is computational complexity.

Q: What's the connection exactly?

A: This is a long story. Let me refer you to the paper [28].

6 Metafinite structures

Q: You said that you didn't coin the term "finite model theory" consciously. Have you ever consciously coined a term?

A: Sure, more than once. One of those terms is relevant to our current conversation: metafinite.

Q: Hmm, you can't be half pregnant or for that matter half infinite. You don't claim that databases are metafinite, I presume, as you claimed a minute ago that they are finite.

A: I do claim that databases are metafinite. Let me explain. A database contains only finitely many records (tuples in relations) at any given moment. In that sense it is perfectly finite. It is also dynamic as we saw in §3. But implicitly databases often involve static infinite structures.

Q: We considered a salary database in §3. Does it involve an infinite structure?

A: Yes, a query to a salary database may return a number, say the average salary, that does not appear in the database. Where does this new number come from? From the infinite structure of rational numbers.

Q: That involvement of an infinite structure, what is it mathematically?

A: A good question. Erich Grädel and I worked on it [31]. In our formalization, a metafinite structure consists of

1. a primary part, which is a finite structure,
2. a secondary part, which is a (usually infinite) structure, e.g. the arithmetic of real numbers, and
3. "weight" functions from the primary part to the secondary.

Q: I guess I see how the salary database looks as a metafinite structure. The primary part is the set of employees, the secondary part is rational arithmetic, and the unique weight function assigns employees their salaries. But how do you compute the average salary?

A: In this example, the secondary part would be an extended rational arithmetic, where the arithmetical operations are applied to finite multisets. In particular, one can obtain the total salary of all employees and the number of employees.

Q: How is this related to finite model theory?

A: Quantification is allowed over the primary part only.

We investigated the theory of metafinite structures, with Erich doing most of the mathematical work. We found out that many results of finite model theory survive, sometimes in more than one form, in metafinite model theory.

Q: Do you view metafinite model theory as a separate research area?

A: No. In my view, it belongs to the research area known as finite model theory.

7 Genericity

Q: I presume that tailoring logics to complexity, at least to polynomial time complexity, has been achieved.

A: Not really. A complication arose. The traditional model of computation used to define complexity classes is the Turing machine model. Inputs for Turing machines are strings. Your algorithm may work with graphs or other structures, but you have to encode your input structure as a string to produce input for a Turing machine. We need a more general computation model that works directly with structures, and replacing your input structure with an isomorphic one should not affect the computation.

Q: This looks to me like a generalization that only mathematicians can come with. What on earth has it to do with query languages?

A: In database theory parlance, query languages should be *generic* which means that they should abstract from the physical layer of databases. In this connection, let us call more general computation models that work directly with structures *generic*; also let us call polynomial time defined according to a generic computation model *generic*.

Q: What does it mean to abstract from the physical layer?

A: For simplicity, consider a database that holds just one relation. That relation is a set of tuples. There is no order on the set. Accordingly, a query asking for the first tuple is meaningless. On the other hand, in a memory of a computer hosting the database the tuples are stored in some order.

Q: Fine, take advantage of that order and increase the expressivity of your query language. Specifically, the first-tuple query will return a particular tuple.

A: But the order is accidental. It varies from one database host to another. The first-tuple query may return different tuples on different hosts.

In [9], Chandra and Harel raised an important question. Fix some standard encoding of structures by binary strings and call a Turing machine M *generic* if the set of structures S such that M accepts the standard string encoding of S is closed under isomorphisms. Chandra and Harel asked whether the set of generic Turing machines is recursive. If the answer is positive, then the good Turing machines constitute a generic computation model.

In [29], I conjectured that the answer to the Chandra-Harel question is negative and that no reasonable (satisfying some minimal conditions) logic captures generic polynomial time on structures. The question remains open.

Actually, the situation is a bit strange. In [6, 7] a logic is defined — it is called BGS (after the authors) — such that every BGS definable property of structures is generic polynomial time computable and we don't know any generic polynomial time property that is not BGS definable.

Q: Do you believe that BGS captures generic polynomial time?

A: No, most probably it does not. In [40], Ben Rossman came close to exhibiting a set of structures that is computable in generic polynomial time but is not BGS definable. He constructed a generic polynomial time computable function that (a) given a finite vector space, constructs its dual and (b) is not BGS definable.

8 Postlude

Q: Have you now covered all DBT/FMT developments that you witnessed or participated in?

A: No. One issue is logic programming languages, specifically Prolog and Datalog. That issue is involved and richly deserves a separate conversation.

Q: Did finite model theory cut the umbilical cord to database management?

A: Yes. "DBT was a huge motivator for FMT," wrote Moshe Vardi to me on Nov. 24 2022, "but then FMT moved from PODS [the Symposium on Principles of Database Systems, the premier international conference on the theoretical aspects of database systems] to LICS [ACM/IEEE Symposium on Logic in Computer Science], which is a conference about mathematical foundations."

Q: Is finite model theory still a separate research area?

A: Research areas are social groups to an extent. My impression is that finite model theory becomes less of a separate research area in spite of strong internal social/communal connections. But I am not the right person to answer this question as I had moved on to other research areas quite a while ago.

Acknowledgement

As I worked on this little paper, I exchanged letters with Anuj Dawar, Ron Fagin, Erich Grädel, Neil Immerman, Mark Liogonky, Ben Rossman, and Moshe Vardi. Their timely replies were most helpful. And I much benefited from discussing all aspects of this paper with Andreas Blass. Finally, the photo of Boris Trakhtenbrot with his students comes courtesy of the Trakhtenbrot family archives, the photos of Dana Scott and William Tait come courtesy of Scott and Tait respectively, the photo of Patrick Suppes is from the Guggenheim Foundation, and the photos of Yuri Glebsky and Edgar Codd are from Wikipedia.

References

[1] Serge Abiteboul, Richard Hall, and Victor Vianu, "Foundations of databases," Addison-Wesley 1995

[2] Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, "Boris A. Trakhtenbrot: Academic Genealogy and Publications," in "Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday," *Lecture Notes in Computer Science* 4800 46–57 Springer 2008

[3] Alfred V. Aho and Jeffrey D. Ullman, "Universality of data retrieval languages," *ACM Symposium on Principles of Programming Languages* 6 110–117 1979

[4] Miklos Ajtai and Yuri Gurevich, "Monotone versus positive," *Journal of Association for Computing Machinery* 34 1004–1015 1987

[5] J. Richard Büchi, "Weak second order arithmetic and finite automata," *Zeitschrift für mathematische Logik and Grundlagen der Mathematik* (now *Mathematical Logic Quarterly)* 6 66–92 1960

[6] Andreas Blass, Yuri Gurevich, and Saharon Shelah, "Choiceless polynomial time," *Annals of Pure and Applied Logic* 100 141–187 1999

[7] Andreas Blass, Yuri Gurevich, and Saharon Shelah, "On polynomial time computation over unordered structures," *Journal of Symbolic Logic* 67:3 1093–1125 2002

[8] Egon Börger, Erich Grädel, and Yuri Gurevich, "The classical decision problem," Springer 1997

[9] Ashok Chandra and David Harel, "Structure and complexity of relational queries," *Journal of Computer and System Sciences* 25 99–128 1982

[10] Chen Chung Chang and H. Jerome Keisler, "Model theory," 3rd edition, Elsevier 1990

[11] Calvin C. Elgot, "Decision problems of finite automata design and related arithmetics," *Transactions of the American Mathematical Society* 98 21–52 1961

[12] Louise H. Chin and Alfred Tarski, "Remarks on projective algebras," *Bulletin of the American Mathematical Society* 54:1 80–81 1948

[13] Edgar F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM* 13:6 377–387 1970

[14] ———, "Relational completeness of database sublanguages," in R. Rustin(editor), "Courant Computer Science Symposium 6: Data Base Systems," 65–98 Prentice-Hall 1972

[15] Ronald Fagin, "Probabilities on finite models," *Notices of American Mathematical Society* A714 Abstract 72T-E90 1972

[16] ———, "Contributions to the model theory of finite structures," University of California Berkeley June 1973

[17] ———, "Generalized first-order spectra and polynomial-time recognizable sets," in Richard M Karp (editor), "Complexity of Computation," *SIAM–AMS Proceedings* 7 43–73 American Mathematical Society 1974

[18] ———, "Monadic generalized spectra," *Zeitschrift für mathematische Logik and Grundlagen der Mathematik* (now *Mathematical Logic Quarterly)* 21 89–96 1975

[19] ———, "A two-cardinal characterization of double spectra," ibid. p. 121–122

[20] ———, "A spectrum hierarchy," ibid. p. 123–134

[21] ———, "Probabilities on finite models," *Journal of Symbolic Logic* 41:1 50–58 1976

[22] ———, "Finite-model theory: A personal perspective," *Theoretical Computer Science* 116 3–31 1993

[23] Yuri V. Glebsky, "Quantitative estimates of the satisfiability of predicate formulas," *Abstracts of short scientific reports of the 1966 International Congress of Mathematicians* Secton 1 p. 32 Moscow 1966 (in Russian)

[24] Yuri V. Glebsky, Dmitry I. Kogan, Mark I. Liogonky, Vladimir A. Talanov, "Range and degree of satisfiability of formulas in the retricted predicate calculus," *Cybernetics (now Cybernetics and Systems Analysis)* 5:2 142–154, Russian version 1969, English translation 1972

[25] Yuri Gurevich, "Elementary properties of ordered abelian groups," *Algebra and Logic* 3:1 5–39 1964 (in Russian), *American Mathematical Society Translations* 46 165–192 1965

[26] ———, "Expanded theory of ordered abelian groups," *Annals of Mathematical Logic* 12 193–228 1977

[27] ———, "Algebras of feasible functions," in FOCS 1982, *IEEE Symposium on Foundations of Computer Science* 24 210–214 1982

[28] ———, "Toward logic tailored for computational complexity," in M. Richter et al. (editors), "Computation and Proof Theory," *Springer Lecture Notes in Mathematics* 1104 175–216 1984

[29] ———, "Logic and the challenge of computer science," in E. Börger (editor), "Current trends in theoretical computer sciene" 1–57, Computer Science Press 1988

[30] ———, "On finite model theory," in S.R. Buss and P.J. Scott (editors), "Feasible Mathematics" 211–219 Birkhäuser 1990

[31] ———, "Metafinite model theory," *Information and computation* 140:1 26–81 1998

[32] Tomasz Imieliński and Witold Lipski, Jr., "The relational model of data and cylindric algebras ," *Journal of Computer and System Sciences* 28 80–102 1984

[33] Neil Immerman, "First order expressibility as a new complexity measure," PhD thesis Cornell 1980

[34] ———, "Number of quantifiers is better than number of tape cell," *Journal of Computer and System Sciences* 22 384–406 1981

[35] ———, "Upper and lower bounds for first order expressibility," ibid 25 76–98 1982

[36] ———, "Relational Queries Computable in Polynomial Time," in STOC 1982, *Symposium on Theory of Computation* 14 147–152 1982

[37] ———, "Descriptive complexity," Springer 1999

[38] Roger C. Lyndon, "An interpolation theorem in the predicate calculus," *Pacific Journal of Mathematics* 9 155–164 1959

[39] Mark I. Liogonky and Vladimir A. Talanov, "On the 0-1 law, discovered by Yuri V. Glebsky, and related results obtained in the Mathematical Logic and Algebra Department of the Nizhny Novgorod State University," *Mathematics in Higher Education* 12 93–102 2014 (in Russian), there seems to be no English translation

[40] Benjamin Rossman, "Choiceless computation and symmetry," in A. Blass, N. Dershowitz, and W. Reisig (editors), "Fields of logic and computation: Essays dedicated to Yuri Gurevich on the occasion of his 70th birthday," *Springer Lecture Notes in Computer Science* 6300 565–580 2010

[41] Dana Scott and Patrick Suppes, "Foundational aspects of theories of measurement," *Journal of Symbolic Logic* 23:2 113–128 1958

[42] William W. Tait, "A Counterexample to a Conjecture of Scott and Suppes", *Journal of Symbolic Logic* 24:1 15–16 1959

[43] Alfred Tarski, "Der Wahrheitsbegriff in den formalisierten Sprachen," *Studia Philosophica* I 261–405 1935. English Translation in "Logic, Semantics and Metamathematics" 152–279 Oxford University Press 1956

[44] Alfred Tarski and Frederick B. Thompson, "Some general properties of cylindric algebras," *Bulletin of the American Mathematical Society* 58:1 p. 65 1952

[45] Boris A. Trakhtenbrot, "The impossibility of an algorithm for the decidability problem on finite classes," *Proceedings of the USSR Academy of Sciences* 70:4 569–572 1950 (in Russian), *American Mathematical Society Translations* Series 2 23 1–6 1963

[46] ———, "On recursive separability," *Proceedings of the USSR Academy of Sciences* 88 953—956 1953 in Russian. No English translation is available, but there is an informative review by Andrzej Mostowski in *Journal of Symbolic Logic* 19:1 p. 60 1954

[47] ———, "Finite automata and the logic of one-place predicates," *Siberian Mathematical Journal* 3 103–131 1962 (in Russian), *American Mathematical Society Translations* Series 2 59 23—55 1966

[48] Moshe Vardi, "Complexity of relational query languages," in STOC 1982, *Symposium on Theory of Computation* 14 137–146 1982

[49] ———, "On decomposition of relational databases," in FOCS 1982, *IEEE Symposium on Foundations of Computer Science* 23 176–185 1982

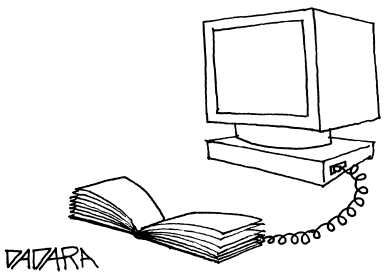
[50] Wikipedia contributors, "Akademgorodok,"
`https://en.wikipedia.org/w/index.php?&oldid=1122095807`

[51] ———, "Edgar F. Codd,"
`https://en.wikipedia.org/w/index.php?&oldid=1125454029`

[52] ———, "Codd's theorem,"
`https://en.wikipedia.org/w/index.php?&oldid=1068162856`

[53] Moshe M. Zloof, "Query-by-example: a database language,"
IBM Systems Journal 16 324–343 1977

News and Conference Reports



REPORT ON ICALP 2022

49th EATCS International Colloquium on Automata, Languages and Programming

Anca Muscholl[1]

The *49th EATCS International Colloquium on Automata, Languages and Programming (**ICALP 2022**)*, the flagship conference and annual meeting of the *European Association for Theoretical Computer Science (EATCS)*, took place in Paris, on July 4-8, 2022. The event was held in hybrid format, and it was hosted by the Research Institute on the Foundations of Computer Science (IRIF) at the Université Paris Cité.

The main conference was preceded by a series of workshops on July 4, 2022. The following workshops were held as satellite events of ICALP 2022:

- Parameterized Approximation Algorithms Workshop (PAAW),
- Combinatorial Reconfiguration,
- Recent Advances on Total Search Problems,
- LearnAut: 4th edition of the Learning and Automata workshop,
- Algorithmic Aspects of Temporal Graphs V (AATG),
- Trends in Arithmetic Theories,
- Structure Meets Power,
- Straight-Line Programs, Word Equations and their Interplay,
- Graph Width Parameters: from Structure to Algorithms (GWP).

The scientific programme of ICALP 2022 consisted of 3 unifying lectures and 3 invited lectures of the two tracks of ICALP, the presentation of 127 contributed papers (which were selected by the Program Committees out of 433 submissions) and award sessions with the lectures of the Gödel Prize 2022, the EATCS Award 2022, and the Presburger Award 2022 recipients.

[1]LaBRI, Université Bordeaux. Email: anca.muscholl@u-bordeaux.fr.

Organisation of ICALP 2022, Environment Sustainability, and EATCS'50 special event. ICALP 2022 had two special features: it aimed toward a more sustainable model of conference, and it presented an exhibition for the 50 years of ICALP and the EATCS.

Towards environmental sustainability several steps were taken:

ICALP 2022 signed the *TCS4F manifesto* (`http://www.tcs4f.org`), a TCS initiative pledging for a reduction of the carbon footprint of our research practices.

The conference ran in *full hybrid mode*. The call for papers explicitly allowed remote participation for authors who would have to take the plane to reach Paris. The hybrid conference used the HOPIN conference platform and the WHY.VISION company took care of the local technical support. The regular talks were 25 minutes long including questions, both for onsite and online speakers. The distant talks (less than one fourth) were presented live, and were smoothly alternating with onsite speakers during the sessions. The virtual platform streamed all the talks, and a replay of all presentations was accessible for several weeks to all participants. Interaction with other participants and speakers was possible through a chat.

An *extended stay support scheme* was organised. It advertised eight participating research laboratories accessible by train from Paris, that were offering to fund collaborations between their members and conference attendees, under the condition that no plane would be taken between the conference and the collaboration.

A *colocated conference* "Highlights of Logic, Games and Automata 2022" (on Track B topics) was scheduled the week before ICALP in Paris.

An *environmental chair*, Hugo Férée, was appointed, with as goal to evaluate the carbon footprint of the conference. The conference emitted 205 tons of CO_2-equivalent (CO_2-e): 172 for ICALP itself, and 30 more for workshop-only participants. Some 33 extra tons would have been emitted if all ICALP speakers would have come to Paris. 6 other tons would have been produced if the Highlights participants did go back home between the two events.

2022 was the *50 year anniversary* of both the EATCS and its flagship conference ICALP. A special *exhibition* was conceived and displayed at IRIF, first during the cocktail, and then for the rest of the conference. The scientific advisory board was chaired by Sylvain Schmitz (IRIF), and consisted of Luca Aceto, Susanne Albers, Giorgio Ausiello, Gilles Dowek, Phokion G. Kolaitis, and Mike Paterson. Extra contributors were solicited: Simon Apers, Alex Bredariol Grilo, Ioannis Chatzigiannakis, Geoffroy Couteau, Pierluigi Crescenzi, Irène Guessarian, Hugo Herbelin, Ce Jin, Jean-Jacques Levy, Kurt Melhorn, Subhasree Patro, Sylvain Périfel, and Jean-Éric Pin. Sandrine Cadet and Sylvain Schmitz were coordinators of the exhibition.

The registration costs were 540€ for regular participants (390€ for students) attending onsite, and 340€ for virtual participants (40€ for students). An extra fee of 100€ was charged for paper authors for covering the LIPIcs proceedings, and organising the event. The workshop day unique price was 50€. This choice was taken for mitigating uncertainties concerning remote participation, and running the hybrid mode.

The organisers and the attendees committed to the ICALP code of conduct, and two SafeToC counsellors, Sophie Laplante and Daniela Petrisan, were appointed.

On behalf of the entire community, we warmly thank the organizers in Paris for their fantastic efforts that helped to make ICALP 2022 and the 50th anniversary of EATCS a great success.

Paper selection and work of the Program Committee. After 15 years running the ICALP conference with three tracks, ICALP returned in 2020 to the original two-track format. The ICALP 2022 program had the following two tracks:

- Track A: Algorithms, Complexity, and Games.
- Track B: Automata, Logic, Semantics, and Theory of Programming.

The PC chairs for the two tracks of ICALP 2022 were David Woodruff (Track A) and Mikołaj Bojańczyk (Track B). Their efforts were supported by 71 members of the Program Committees (43 in track A and 28 in track B). In response to the call for papers, a total of 433 submissions were received: 350 for Track A and 83 for Track B. Each submission was assigned to at least three Program Committee members, aided by more than 600 external subreviewers. The Program Committees decided to accept 127 papers for inclusion in the scientific program: 103 papers for Track A and 24 for Track B. This gives the acceptance rate for the entire conference to be 29.3%. The selection was made by the Program Committees based on originality, quality, and relevance to theoretical computer science. The quality of the submitted manuscripts was very high, and unfortunately many strong papers could not be selected. We take this opportunity of thanking both the Program Committees and the subreviewers for doing an exceptional selection job.

Statistical information about the number of papers submitted and accepted for the last several editions of the ICALP conference, as well as acceptance rates, are available in Tables 1–3.

Invited presentations. In addition to the contributed talks, ICALP 2022 featured three invited presentations by Leslie Ann Goldberg (Track A), Stéphan Thomassé (Track B), and Santosh Vempala (Track A), and three unifying talks by Albert Atserias, Constantin Daskalakis and Madhu Sudan:

	2022	2021	2020	2019	2018	2017	2016	2015	2014	2013
Total	433	462	470	490	502	459	515	507	477	423
Track A	350	361	347	316	346	296	319	328	312	249
Track B	83	101	123	103	96	108	121	114	106	114
Track C	—	—	—	71	60	55	75	65	59	60

Table 1: Number of submitted papers at ICALP 2013–2022.

	2022	2021	2020	2019	2018	2017	2016	2015	2014	2013
Total	127	137	138	146	147	137	146	143	136	124
Track A	103	108	102	94	98	88	89	89	87	71
Track B	24	29	36	31	30	32	36	34	31	33
Track C	—	—	—	21	19	17	21	20	18	20

Table 2: Number of accepted papers at ICALP 2013–2022.

- Albert Atserias (Universitat Politècnica de Catalunya): *Towards a Theory of Algorithmic Proof Complexity*
- Constantin Daskalakis (MIT): *Equilibrium Computation, Deep Learning, and Multi-Agent Reinforcement Learning*
- Leslie Ann Goldberg (Oxford): *Some New (And Old) Results on Contention Resolution*
- Santosh S. Vempala (Georgia Tech.): *The Manifold Joys of Sampling*
- Madhu Sudan (Harvard): *Streaming and Sketching Complexity of CSPs: A Survey*
- Stéphan Thomassé (ENS Lyon): *A Brief Tour in Twin-Width*

ICALP Proceedings. As it has been the tradition since 2016, ICALP proceedings were published with *LIPIcs*. *LIPIcs – Leibniz International Proceedings in Informatics* is a series of high-quality conference proceedings across all fields in informatics established in cooperation with Schloss Dagstuhl–Leibniz Center for Informatics. All 127 ICALP 2022 contributed papers presented at the conference, papers or abstracts accompanying the invited talks of Albert Atserias, Constantin Daskalakis, Leslie Ann Goldberg, Madhu Sudan, Stéphan Thomassé, Santosh

	2022	2021	2020	2019	2018	2017	2016	2015	2014	2013
Total	29.3	29.6	29.4	29.8	29.3	29.8	28.3	28.2	28.5	29.3
Track A	29.4	29.9	29.4	29.7	28.3	29.7	27.9	27.1	27.9	28.5
Track B	28.9	28.7	29.3	30.1	31.3	29.6	29.8	29.8	29.2	28.9
Track C	—	—	—	29.6	31.7	30.9	28.0	30.8	30.5	33.3

Table 3: Acceptance rates (in %) for ICALP 2013–2022.

Vempala were published according to the principle of Open Access in LIPIcs Proceedings volume 229, and made available online and free of charge at `https://drops.dagstuhl.de/opus/portals/lipics/index.php?semnr=16238`. The proceedings editors are Mikołaj Bojańczyk, Emanuela Merelli and David P. Woodruff.

ICALP and EATCS Awards. The EATCS sponsors awards for both a best paper and a best student paper in each of the two tracks at ICALP, as selected by the Program Committees. During the general assembly, the ICALP Best Paper and Best Student Paper Awards were presented to the authors of the following papers:

Best Paper in Track A: Ian Newman and Nithin Varma. *Strongly Sublinear Algorithms for Testing Pattern Freeness*.

Best Paper in Track B: Jakub Gajarsky, Michał Pilipczuk, Wojciech Przybyszewski and Szymon Toruńczyk. *Twin-width and types*.

Best Student Papers in Track A: Joakim Blikstad. *Sublinear-round Parallel Matroid Intersection*, and Jakub Tětek. *Approximate Triangle Counting via Sampling and Fast Matrix Multiplication*.

Best Student Paper in Track B: Gaëtan Douéneau-Tabot. *Hiding pebbles when the output alphabet is unary*.

The program of ICALP 2022 included presentations of several prestigious scientific awards sponsored or co-sponsored by EATCS:

- The **Gödel Prize 2022** was awarded to the following papers

  - **Zvika Brakerski, Vinod Vaikuntanathan**: *Efficient Fully Homomorphic Encryption from (Standard) LWE*. FOCS 2011: 97-106. SIAM Journal of Computing 43(2): 831-871 (2014), and

– **Zvika Brakerski, Craig Gentry, Vinod Vaikuntanathan**: *(Leveled) fully homomorphic encryption without bootstrapping*. ITCS 2012: 309-325. ACM Transactions on Computation Theory 6(3): 13:1-13:36 (2014)

The above papers made transformative contributions to cryptography by constructing efficient fully homomorphic encryption (FHE) schemes.[2]

- The **EATCS Award 2022**, the annual EATCS Distinguished Achievements Award given to acknowledge extensive and widely recognized contributions to theoretical computer science over a life long scientific career, was awarded to **Patrick Cousot** (New York University) for introducing and developing, together with his late wife Radhia Cousot, the framework of abstract interpretation for program analysis.[3]

- The **Presburger Award 2022** for Young Scientists was awarded to **Dor Minzer** (MIT) for for his deep technical contributions towards resolving the 2-to-2 Games Conjecture.[4]

- The **EATCS Distinguished Dissertation Awards 2022**, to promote and recognize outstanding dissertations in theoretical computer science were awarded to three researchers:

  – **Alexandros Hollender** for his dissertation *Structural Results for Total Search Complexity Classes with Applications to Game Theory and Optimisation*, advised by Paul W. Goldberg at the University of Oxford,

  – **Jason Li** for his dissertation *Preconditioning and Locality in Algorithm Design*, advised by Anupam Gupta and Bernhard Haeupler at the Carnegie Mellon University,

  – **Jan van den Brand** for his dissertation *Dynamic Matrix Algorithms and Applications in Convex and Combinatorial Optimization*, advised by Danupon Nanongkai at the KTH Royal Institute of Technology.

[2]The laudatio for the Gödel Prize 2022 is available at `https://eatcs.org/index.php/component/content/article/1-news/2917-2022-05-21-19-57-34`.

[3]The laudatio for the EATCS Award 2022 is available at `https://eatcs.org/index.php/component/content/article/1-news/2918-the-eatcs-award-2022-laudatio-for-patrick-cousot`.

[4]The laudatio for the Presburger Award 2022 is available at `https://eatcs.org/index.php/component/content/article/1-news/2914-presburger-award-2022-laudatio-for-dor-minzer`.

- The EATCS has recognized two of its members for their outstanding contributions to theoretical computer science by naming them **EATCS Fellows** class of 2022:
  - **Samson Abramsky** (University College London) for fundamental contributions to logic in computer science, including domain theory in logical form, game semantics, and category-theoretic foundations of quantum computing.
  - **Orna Kupferman** (Hebrew University) for fundamental contributions to automata- and game-theoretic techniques aiming at the formal verification and reactive synthesis of computing systems.

The recipients of the Gödel Prize 2022 (Zvika Brakerski, Craig Gentry and Vinod Vaikuntanathan), of the EATCS Award 2022 (Patrick Cousot), and of the Presburger Award 2022 (Dor Minzer) gave presentations in the Award Sessions at ICALP 2022.

We congratulate all the winners and we hope their achievements will put the highlights of research in theoretical computer science in the spotlight, and will serve as inspirations to young researchers in the years to come.

We hope that this conference report gives you a glimpse of the rich scientific and social programme that made the 49th ICALP an unforgettable conference. Everyone involved in the organization of ICALP 2022 deserves the warmest thanks from the TCS community.

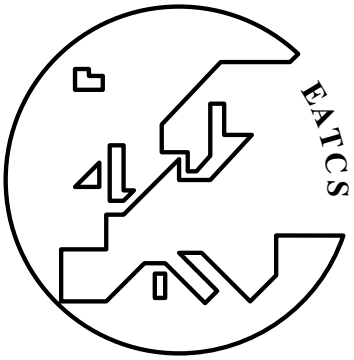
We wish to thank all authors who submitted extended abstracts for consideration, the Program Committees for their effort, and all the referees who assisted the Program Committees in the evaluation process. We are also grateful to the Conference Chair Thomas Colcombet and all the support staff of the Organizing Committee for organizing ICALP 2022. We also acknowledge support by FSMP, Université Paris cité, CNRS, Inria, IRIF and NOMADIC LABS.

The General Assembly of the EATCS was informed about the progress of the organization of the *50th EATCS International Colloquium on Automata, Languages and Programming*, ICALP 2023, that will take place in Paderborn, Germany, on July 10–14, 2023, with Sevag Gharibian as the general chair of the conference. The ICALP 2023 Program Committee Chairs are Uriel Feige (Track A) and Kousha Etessami (Track B). The ICALP 2022 call for papers is available at `https://icalp2023.cs.upb.de/call-for-papers/`.

We hope that you will make plans to submit your best work to ICALP 2023 and be able to go to Paderborn for the conference. We look forward to seeing you there.

BEATCS no 139

European Association for Theoretical Computer Science



E A T C S

124

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

(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 for-

mats, 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`,